arXiv:2303.01568v1 [cs.DC] 2 Mar 2023

HitGNN：面向中央处理器+多 FPGA异构平台的高吞吐量GNN训练框架

Yi-Chien Lin、Bingyi Zhang、Viktor Prasanna

摘要——随着现实世界中的图规模不断增大，训练图神经网络（GNN）已变得耗时，因此需要加速。尽管以往的工作已证明利用现场可编程门阵列加速GNN训练的潜力，但由于需要硬件专业知识且开发工作量巨大，利用多个现场可编程门阵列加速GNN训练的研究仍然很少。为此，我们提出HitGNN，这是一种框架，使用户能够轻松地将GNN训练工作负载映射到CPU-多FPGA平台上进行加速。具体而言，HitGNN以用户定义的同步GNN训练算法、GNN 模型和平台元数据为输入，根据平台元数据确定设计参数，并自动将硬件映射到CPU+多 FPGA平台。HitGNN由以下构建模块组成：（1）高级应用程序编程接口（API），用户只需编写少量代码，即可指定各种同步GNN训练算法和GNN 模型；（2）软件生成器，用于生成主机程序，该程序执行小批量采样、管理CPU-FPGA通信，并处理各FPGA之间的工作负载均衡；（3）加速器生成器，用于生成具有优化数据通路和存储器组织的GNN内核。我们展示了使用该框架，可以轻松地将DistDGL和PaGraph等现有同步GNN训练算法部署到CPU+多 FPGA平台上，同时实现高训练吞吐量。与在多GPU平台上加速同步GNN训练的最先进框架相比，HitGNN的带宽效率最高可达27.21×，加速比最高可达4.26×，且所需计算能力和内存带宽远低于GPU。此外，HitGNN在CPU+多 FPGA平台上展现出良好的可扩展性，可支持多达16个FPGA。

索引词—图神经网络、中央处理器+多 FPGA、硬件加速

1 引言

图 RAPH 神经网络（GNN）已成为图表示学习领域的先进模型，推动了众多应用的发展，例如社交推荐系统 [1], [2], 分子性质预测[3], [4] 和交通预测 [5], 等。最初，GNN 在图形处理器 [6], [7], [8] 或 FPGA 平台[9], [10], [11]上进行计算；然而，随着图规模不断增大，在单个图形处理器或 FPGA 平台上计算 GNN 会变得耗时。因此，许多研究 [12], [13], [14],[15], [16]提出在多CPU或多GPU平台上加速 GNN 训练，因为这类平台能提供更大的内存带宽和更多计算资源。为了在多个设备上并行训练 GNN，这些研究采用同步 *GNN* 训练；我们将在第 2.3 节详细介绍同步 GNN 训练。

与 CPU 和 GPU 等通用处理器相比，中央处理器+现场可编程门阵列异构平台在图神经网络训练加速方面颇具潜力：中央处理器可以灵活支持各种图预处理和小批量采样算法；而现场可编程门阵列能够高效执行 GNN 运算，因为现场可编程门阵列支持定制化的数据访问模式和内存组织，可以有效减少 GNN训练中的大量内存流量和随机内存访问。尽管在中央处理器+现场可编程门阵列平台上可以部署各种优化，但训练 GNN 仍然——可能由于计算能力和存储器资源有限而耗时较长；因此，在中央处理器+多 FPGA 异构平台上加速 GNN训练是十分必要的。然而，在中央处理器+多 FPGA 平台上加速 GNN训练面临诸多挑战。首先，在中央处理器+多 FPGA 平台上训练 GNN 会受到工作负载不平衡以及 FPGA 之间大量数据通信开销的影响，从而导致性能低下且可扩展性较差。其次，在中央处理器+多 FPGA 平台上开发优化内核需要硬件专业知识，并且探索复杂的硬件设计空间也需要耗费大量时间。

受这些挑战的启发，我们提出了 HitGNN，这是一种通用框架，用于将同步 GNN 训练算法映射到中央处理器+多 FPGA 异构平台上。我们首先对同步 GNN 训练算法进行高级抽象，然后基于该抽象开发 HitGNN；因此，我们的框架能够支持各种可用所定义抽象描述的训练算法。为了实现高吞吐量并自动化实现流程，我们开发了一种硬件设计空间探索（DSE）引擎。给定平台元数据后，该设计空间探索引擎会确定能够优化训练吞吐量的加速器配置。为减少开发工作量，HitGNN 配备了内核库，其中包含可直接使用的优化 GNN 内核。为屏蔽硬件实现细节，HitGNN 提供了应用程序编程接口（API），使开发者能够轻松实现

• *Y. Lin*、*B. Zhang* 和 *V. K. Prasanna* 任职于南加州大学电气与计算机工程系，地址为加利福尼亚州洛杉矶市 *90089*。电子邮件：{*yichienl, bingyizh, prasanna*}*@usc.edu*

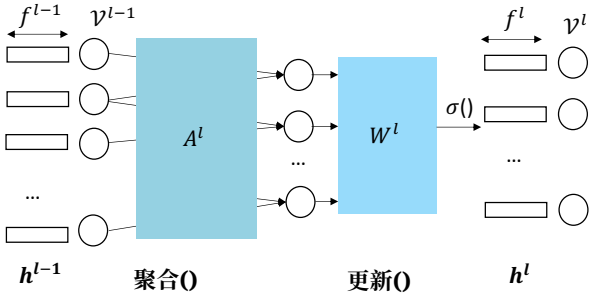

图 1：GNN 层的计算抽象

各种训练算法和 GNN 模型，只需编写少量代码。最后，HitGNN 的主机程序负责小批量采样、处理工作负载不平衡，并减少 FPGA 之间的数据通信。我们的贡献总结如下：

- 我们提出了 HitGNN，这是一种通用框架，可将多种同步 GNN 训练算法（如 P3、DistDGL 和 PaGraph）以及多种 GNN 模型（如 GCN 和 Graph-SAGE）自动映射到中央处理器+多 FPGA 异构平台上以实现加速。
- 为隐藏硬件实现细节，HitGNN 提供易于使用的编程 API，用户只需编写少量代码，即可指定各种同步 GNN 训练算法和 GNN 模型。
- 为减少开发工作量，我们设计了一个图神经网络内核库，其中包含各种知名图神经网络模型的参数化优化内核，以及一个可自动确定加速器配置的设计空间探索引擎。
- 为实现可扩展设计，我们开发了多种优化方法，以减少 FPGA 间通信开销，并平衡各 FPGA 的工作负载。
- 与多GPU平台上最先进的同步 GNN 训练实现相比，HitGNN 使用更少的计算资源和内存带宽，实现了最高 27.21× 的带宽效率和 4.26× 的加速比。

2 背景

2.1 图神经网络模型

给定输入图 $\mathcal{G}(\mathcal{V},\mathcal{E},\boldsymbol{X})$，其中 $\mathcal{V}$、$\mathcal{E}$ 和 $\boldsymbol{X}$ 分别表示图的顶点集合、边集合和特征矩阵，一个 GNN 模型定义如下：

- L：层数。
- $\mathcal{V}^t$：待推理的目标顶点集合。
- f^l：第 l （$1 \leqslant l \leqslant L$） 层的隐藏维度。
- 一种用于构建小批次的机制，包括：
  - 构建 $\mathcal{V}^l$的机制：第 l 层中的顶点集合($0 \leqslant l \leqslant L$)。 $|\mathcal{V}^l|$ 表示第 l层中的顶点数量。此外，$\mathcal{V}^L = \mathcal{V}^t$。- 构建 $\boldsymbol{A}^l \in \mathbb{R}^{|\mathcal{V}^{l-1}|\times|\mathcal{V}^l|}$的机制：用于特征聚合的邻接矩阵

算法 1 图神经网络计算抽象

1: **对于** $l = 1...L$，**执行**2: **对于**顶点 $v \in \mathcal{V}^l$，**执行**3: $\boldsymbol{a}_v^l =$ **聚合(**$\boldsymbol{h}_u^{l-1}$: $u \in \mathcal{N}(v)$ 和 $u \in \mathcal{V}^{l-1}$**)**4: $\boldsymbol{h}_v^l =$ **更新(**$\boldsymbol{a}_i^l, \boldsymbol{W}^l$, $\sigma()$**)**5: **结束循环**6: **结束循环**

层 l （$1 \leqslant l \leqslant L$）。 $\boldsymbol{A}^l$ 定义了 $\mathcal{V}^{l-1}$ 和 $\mathcal{V}^l$之间的层间连接（边）。

- **聚合()** 函数，用于每个顶点从其邻居聚合信息。
- **更新()** 函数包含一个多层感知器（MLP）和激活函数 $\sigma()$，用于执行特征更新。- $\boldsymbol{W}^l \in \mathbb{R}^{f^{l-1}\times f^l}$：层 l ($1 \leqslant l \leqslant L$) 的权重矩阵，在更新函数中用于对顶点特征执行线性变换。
- $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}|\times f^l}$：输入特征矩阵，其中每一行表示一个顶点的特征向量。- $\boldsymbol{h}^l \in \mathbb{R}^{|\mathcal{V}^l|\times f^l}$：层 l ($0 \leqslant l \leqslant L$) 中的顶点特征矩阵。此外，输入层的特征矩阵就是输入特征矩阵，即 $\boldsymbol{h}^0 = \boldsymbol{X}$；最后一层 $\boldsymbol{h}^L$ 的特征矩阵是目标顶点 $\mathcal{V}^t$的节点嵌入。

图神经网络通过迭代聚合和更新目标顶点 $\mathcal{V}^t$的 L–跳邻居的顶点特征，学习为一组目标顶点生成低维向量表示（即节点嵌入）。我们在图 1 中描述了 GNN 层的计算抽象。从层 1开始，GNN 模型通过聚合和更新 $\mathcal{V}^1$ 中每个顶点的邻居顶点在 $\mathcal{V}^0$中的特征向量，计算每个顶点的特征向量；该过程重复 L 次，直到得到目标顶点 $\mathcal{V}^t$ 的节点嵌入（即 $\mathcal{V}^L$）。GNN 模型的计算过程如算法 1 所示，也称为聚合-更新范式 [17]。 $\boldsymbol{a}_v^l \in \mathbb{R}^{f^l}$ 是 $v \in \mathcal{V}^l$的中间结果，而 $\mathcal{N}_s(v)$ 表示 v 在 $\mathcal{V}^{l-1}$中的采样邻居。

2.2 小批次GNN训练

GNN 模型可以采用小批次方式进行训练，训练过程包括五个阶段 [3], [6], [8]: 采样、前向传播、损失计算、反向传播和权重更新。在采样阶段，从输入图拓扑 $\mathcal{G}$（$\mathcal{V}$，$\mathcal{E}$）中采样一组顶点和邻接矩阵。我们使用 $\mathcal{V}^l$ 表示从 $\mathcal{V}$ 的第 l层中采样得到的顶点。 $\boldsymbol{A}^l$ 表示采样得到的邻接矩阵，用于描述小批次中 $\mathcal{V}^{l-1}$ 与 $\mathcal{V}^l$ 之间的层间连接（边）。一个小批次由目标顶点 $\mathcal{V}^L$、每一层的采样顶点 $\{\mathcal{V}^l: 0 \leqslant l \leqslant L-1\}$以及采样得到的邻接矩阵（边） $\{\boldsymbol{A}^l: 1 \leqslant l \leqslant L-1\}$组成。

在前向传播阶段，小批次按层处理；最后一层的输出是目标顶点 $\{\boldsymbol{h}_v^L: v \in \mathcal{V}^L\}$ 的节点嵌入，随后将其与真实值进行比较，以计算损失。计算得到的损失用作

表 1：同步 GNN 训练算法

同步 GNN 训练算法	图划分	特征存储策略
DistDGL [13]	采用多重约束的 METIS	基于图划分
PaGraph [15]	一种旨在均衡各分区中训练顶点数量的贪心方法	存储出度较高的顶点的特征向量
P^3 [12]	沿特征维度分区	基于图划分

算法2 小批量图神经网络训练算法

对于每次迭代**执行**
  采样($\mathcal{G}(\mathcal{V},\mathcal{E})$**)** ▷ 推导小批次
  对于 $l = 1...L$，**执行** ▷前向传播
    对于 顶点 $v \in \mathcal{V}^l$，**执行**
      $\boldsymbol{a}_v^l =$ **聚合(**$\boldsymbol{h}_u^{l-1}$ ： $u \in \mathcal{N}_s(v)$， $u \in \mathcal{V}^{l-1}$**)**
      $\boldsymbol{h}_v^l =$ **更新(**$\boldsymbol{a}_i^l$， $\boldsymbol{W}^l$， $\sigma()$**)**
    结束循环
  结束循环
  计算损失($\{\boldsymbol{h}_i^L: v_i \in \mathcal{V}^L\}$**)**
  反向传播( ) ▷ 推导 W^l 的梯度
  权重更新( )
结束循环

反向传播的输入，反向传播执行与前向传播类似的计算，但方向相反。最后，推导出 $\boldsymbol{W}^l$ 每层的梯度，用于权重更新。我们在算法 2 中展示了 GNN训练的步骤。

图神经网络模型也可以采用全图方式进行训练，这种方法无需采样阶段；然而，全图训练会导致较大的内存占用 [18], [19]，可能无法装入设备内存（例如 FPGA本地DDR）。因此，HitGNN 专注于加速小批量 GNN 训练，因为它在准确率、大图上的可扩展性方面具有优势，且已被许多最先进的 GNN 框架采用 [6], [8], [15], [20]。

2.3 同步 GNN 训练

现有工作 [12], [13], [15] 采用同步随机梯度下降（SGD）[21]，在多中央处理器或多图形处理器平台上训练 GNN。在本文其余部分，我们使用同步 *GNN* 训练 来指代在多个设备上通过同步随机梯度下降并行训练 GNN。

同步 GNN 训练与算法2类似，但多出两个阶段：图预处理和梯度同步。第一个阶段是图预处理。在此阶段，输入图 $\mathcal{G}(\mathcal{V},\mathcal{E})$ 会被划分并分配到各个设备（如图形处理器或现场可编程门阵列）上进行并行训练。除了图划分，图预处理阶段还会进行特征存储，将特征向量存储在设备本地内存中（例如图形处理器全局内存或 FPGA 本地 DDR）。对于图形处理器或现场可编程门阵列等设备，整个特征矩阵

$\boldsymbol{X}$ 大型图的特征矩阵可能过大，无法装入本地存储器；因此，现有研究 [12], [13], [15] 提出了各种特征存储策略，仅将部分特征矩阵存储在设备本地内存中。我们用 $\boldsymbol{X}_i$ 表示存储在设备 i本地存储器中的所选特征向量。对于 DistDGL 等工作，特征存储策略基于

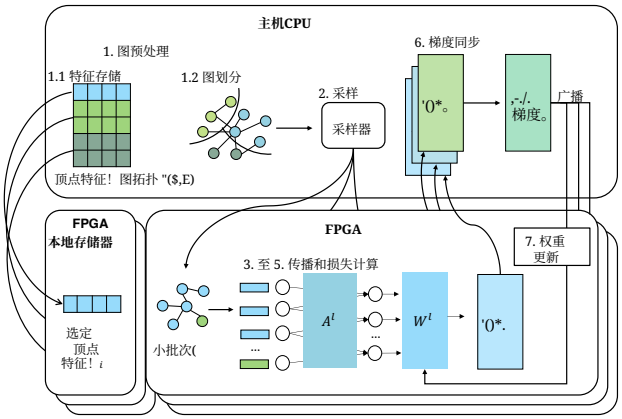

图2：在中央处理器+多 FPGA 平台上进行同步 GNN 训练

图划分的结果；也就是说，如果 v_j 属于分区i，那么 $v_j \in X_i$的特征向量。PaGraph 等其他工作则提出缓存策略，用于存储频繁访问顶点的特征向量，这与图划分无关。图预处理完成后，每个设备并行执行前向传播、损失计算和反向传播。随后执行梯度同步，对从各设备收集到的梯度求平均。然后，将平均梯度广播到每个设备，以更新模型权重。图 2 展示了中央处理器+多 FPGA 平台上的同步 GNN 训练工作流程。中央处理器+多 FPGA 平台由单个中央处理器或多个中央处理器（具体数量取决于机器中的插槽数）以及多个 FPGA 组成。中央处理器通过 PCIe 连接到 FPGA，每个 FPGA 则连接到本地 DDR 内存。

表 1 列出了几种具有代表性的同步 GNN 训练算法。这些算法的差异在于图划分和特征存储策略。其他阶段（如前向传播、梯度同步等）则完全相同。因此，为简洁起见，我们只展示这两个不同的阶段。

2.4 相关工作

已有多项研究提出利用多块 FPGA 加速神经网络训练：[22], [23]利用 FPGA 集群加速卷积神经网络（CNN）训练。尽管这些工作在性能和能效方面展现出良好的成果，但 CNN 加速器无法直接用于 GNN 训练。这是因为 CNN 和 GNN 的计算特性截然不同：CNN 模型以结构化输入数据为特征，计算强度较高；而 GNN 模型

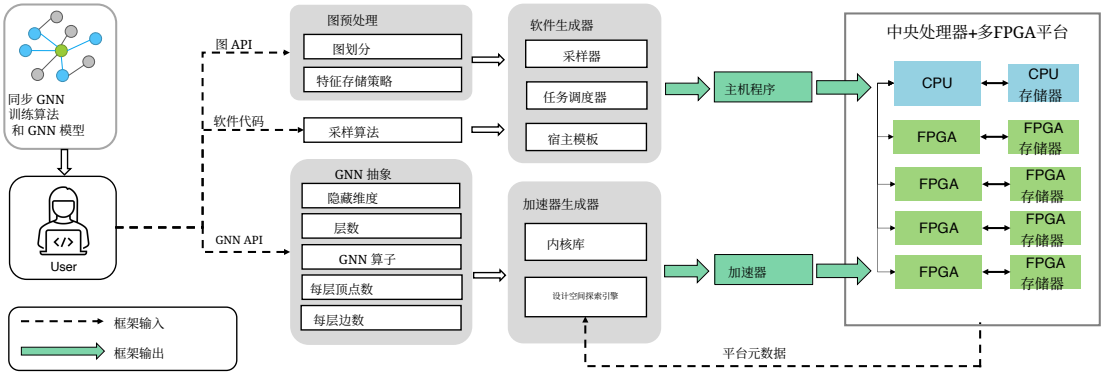

图 3：框架概述

以非结构化输入数据为特征，计算强度较低。也有研究利用多块 FPGA 加速 GNN 训练。 [18] 在分布式平台上加速 GNN 训练，其中图存储在多个节点上。在分布式平台上，训练性能受采样阶段的瓶颈限制。在这项工作中，我们聚焦于连接多块 FPGA 的单节点平台。在此类平台上，性能瓶颈出现在传播阶段。 [24]在单节点平台上加速 GNN 训练；然而，[24]仅加速一种特定的同步 GNN 训练算法，且未提供任何帮助用户降低开发工作量的工具。在这项工作中，我们提出一个通用框架，可在中央处理器+多 FPGA 平台上加速多种同步 GNN 训练算法。此外，我们的框架还提供软件 API、设计空间探索引擎和优化库，以降低在中央处理器+多 FPGA 平台上加速 GNN 训练的开发工作量。

3 挑战

我们识别出在中央处理器+多FPGA平台上加速同步图神经网络训练时面临的若干挑战。

挑战 1： 在中央处理器+多FPGA平台上开发图神经网络训练加速器耗时较长，并且需要硬件专业知识。具体而言，用户需要开发高度优化的内核，并探索硬件设计空间以充分利用可用资源；用户还需要开发主机程序，以管理 FPGA 之间的任务调度和数据通信。

挑战 2： 在中央处理器+多 FPGA 平台上实现可扩展加速。首先，图结构数据的数据依赖会在训练期间产生较高的 FPGA 间通信开销。其次，由于图划分，各 FPGA 上的工作负载可能不平衡。例如，在 DistDGL 中， [13],METIS 旨在通过最小化跨分区边连接来降低通信开销，但它无法为每个分区分配相同数量的顶点和边，因此会导致工作负载不平衡。

挑战 3： HitGNN 中的优化需要具有足够的通用性，以支持各种同步 GNN 训练算法；此外，这些优化不应改变算法本身，否则会影响模型准确率和收敛速度。

为应对挑战 1，HitGNN 通过软件编程 API 隐藏硬件实现细节，并借助软件生成器和加速器生成器自动完成设计步骤（第 4 节）；为应对挑战 2 和 3，我们针对中央处理器+多 FPGA 平台开发了多种优化，以提高可扩展性；这些优化适用于各种同步 GNN 训练算法。此外，尽管采用了多种优化，HitGNN 执行的计算与原始训练算法相同，因此准确率和收敛速度保持不变（第 5 节）。

4 框架

下面介绍 HitGNN 的工作流程：在设计阶段，用户指定同步 GNN 训练算法、GNN 模型和目标平台的元数据。随后，HitGNN 自动生成经过优化的加速器设计和主机程序（第 4.1 节）；在运行阶段，用户在中央处理器+多 FPGA 平台上启动 GNN 训练。我们将在第 4.2 节说明生成的设计如何映射到目标平台。

4.1 框架概述

我们在图 3 中展示了 HitGNN 的框架概述。HitGNN 以同步 GNN 训练算法、 GNN 模型和平台元数据为输入，生成高吞吐量设计，以加速在中央处理器+多 FPGA 平台上的训练。具体而言，该设计由以下部分组成：(1) 一个负责管理任务调度、数据通信和小批量采样的主机程序；(2) 在 FPGA 上运行的加速器设计，即经过优化的 GNN 内核。在输入程序中，用户通过两组 API 指定同步 GNN 训练算法：

- **图 API**：指定图预处理阶段（第 2.3 节中提及）的图划分和特征存储策略。
- **GNN API**：第 2.1 节中提到的用于定义 GNN 模型的参数。

表 2：HitGNN 的应用程序编程接口

API 类型	API 函数	描述
图 API	图分区( )	将一组顶点 $\mathcal{V}$ 和边 $\mathcal{E}$ 分配到一个现场可编程门阵列
	特征存储( )	将选定的顶点特征 $\boldsymbol{X}_i$ 传输到现场可编程门阵列的本地存储器
GNN API	GNN 参数( )	层数 L，以及特征长度 f^l
	图神经网络计算( )	GNN 模型中的层运算符。指定现成的 GNN 模型或“自定义”
	散射( )	用户定义函数，仅在指定自定义层算子时需要
	收集( )	仅在指定自定义层算子时需要用户定义函数
	更新( )	仅在指定自定义层算子时需要用户定义函数
	GNN 模型( )	使用 GNN 参数构建 GNN 模型
主机 API	FPGA 元数据( )	FPGA 内存带宽、数字信号处理器数量、查找表等。
	平台元数据( )	PCIe 内存带宽、FPGA 数量等。
	生成设计( )	生成硬件设计和软件设计
	加载输入图( )	读取输入图的文件路径
	开始训练( )	运行 GNN 训练
	保存模型( )	保存训练好的 GNN 模型

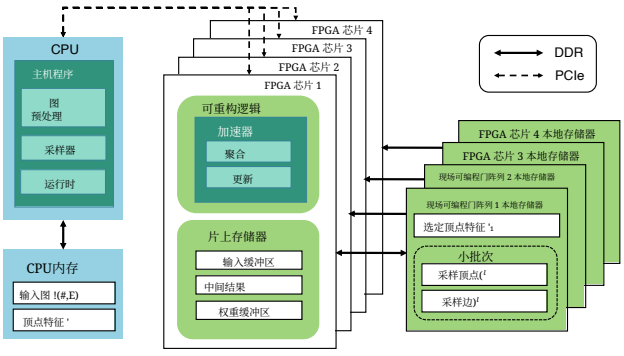

图 4：系统概览

随后，HitGNN 会解析输入程序，并提取同步 GNN 训练算法的抽象，作为软件和加速器生成器的中间表示，以生成可在中央处理器+多 FPGA 平台上运行的设计。下面分别介绍各个构建模块：

软件生成器：软件生成器根据用户指定的输入生成主机程序。在预处理阶段，主机程序执行图划分，并将顶点特征分发到每个 FPGA。在训练期间，主机程序执行小批次生成，并将小批次分发到每个 FPGA。主机程序还包含一个运行时系统，用于管理 FPGA 任务调度和数据通信。

加速器生成器：加速器生成器解析用户输入，并使用内核库生成参数化硬件设计；参数（即加速器配置）由设计空间探索引擎确定。随后，加速器生成器为目标 FPGA 生成可综合硬件。

- **内核库**：HitGNN 为多种广泛使用的图神经网络模型提供了采用高级综合（HLS）编写的优化硬件内核。用户可以在内核库中实现现有模型，也可以使用经过优化的内核模板定制自己的模型。
- **设计空间探索引擎**：设计空间探索引擎以平台元数据（例如 PCIe 带宽、FPGA 数量）为输入，探索硬件设计空间，并生成能够优化 GNN训练吞吐量的加速器配置（第 6 节）。

4.2 系统概览

图 4 展示了如何将同步 GNN 训练算法映射到 CPU+多 FPGA 平台上。运行期间，CPU 上的主机程序首先执行图预处理，并将选定顶点特征分发到每个 FPGA。随后，主机启动 GNN训练。采样器生成小批次，运行时系统将这些小批次分发到 FPGA 本地内存。同时，FPGA 执行 GNN 运算。如果计算所需的顶点特征不在 FPGA 本地内存中，FPGA 会向主机CPU发送请求，运行时系统则从 CPU内存读取数据并将其传输到 FPGA（第 5.2 节）。完成反向传播并得出梯度后，每个 FPGA 都会将梯度发送回主机以进行同步。运行时系统对收集到的梯度求平均，然后将平均后的梯度广播回每个 FPGA，以执行全局权重更新。

HitGNN 在主机CPU上执行图预处理和采样，并在 FPGA 上执行包括特征聚合和特征更新在内的图神经网络运算。根据这一任务分配，输入图拓扑 $\mathcal{G}(\mathcal{V},\ \mathcal{E})$ 和顶点特征 X 存储在主机内存中，由主机CPU执行图预处理和采样；小批次拓扑 $\mathcal{V}^l$ 和 $\boldsymbol{A}^l$ 以及选定顶点特征X_i 存储在 FPGA 本地内存中，由 FPGA 执行图神经网络运算。

4.3 高级 API

表 2总结了为用户提供的高级 API，用于编程同步 GNN 训练

Python。列表 1展示了如何使用 HitGNN 将异步GNN训练算法映射到目标中央处理器+多FPGA平台。

```
1 ### 设计阶段 ### 2 3 ’’’ 4 运行图预处理程序，生成 5 V[p], E[
```

列表 1：用户程序示例

5 中央处理器+多 FPGA 优化

在中央处理器+多 FPGA 平台上加速同步 GNN 训练会面临工作负载不平衡和较高的数据通信开销。我们将在第 5.1 节和第 5.2 节中介绍为应对这些挑战而采用的优化方法。此外，我们还将介绍 HitGNN 中的内核库，该内核库由经过优化以实现高训练吞吐量的 GNN 内核组成（第 5.3 节）。

5.1 工作负载均衡

在图预处理阶段（第 2.3 节），图被划分为 p 个分区，其中 p 表示目标平台上的 FPGA 数量。在采样阶段，采样器从每个图分区中采样小批次，并将这些小批次分发到各个 FPGA。如第 3 节所述，每个分区中的顶点数和边数各不相同；因此，每个图分区中的小批次数量也各不相同，从而导致工作负载不平衡。

我们提出了一种两阶段任务调度器，用于平衡各 FPGA 之间的工作负载。我们通过图5中的 $p = 3$ 说明这一思路。采样器对每个图分区进行采样并生成小批次（以圆圈表示）。小批次的颜色表示其所属的图分区
已采样。每个小批次上标注的数字表示其生成顺序。图5中标注的迭代对应于算法2中的迭代。

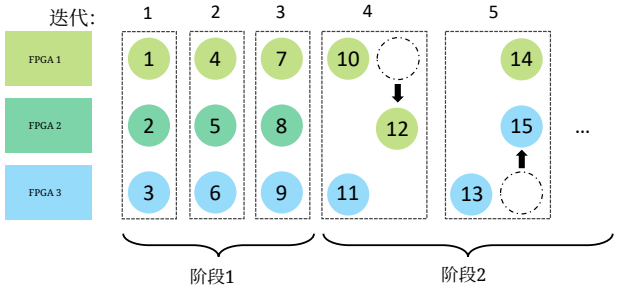

图5：两阶段任务调度

算法3 两阶段任务调度 输入：分区图拓扑 $\mathcal{V}$[i], $\mathcal{E}$[i],，其中 $0 \leq$ i $< p$输出：小批次 b 及其FPGA分配 1：阶段1 2：当每个分区中的小批次数量 > 0 时 3：对于 i 从 0 到 p 执行 4： b **=采样(**$\mathcal{V}$**[i],** $\mathcal{E}$**[i]) 5：分发(**b**, i)** ▷ **分发到FPGA** i **6：结束循环 7：结束循环 8：阶段2 9：cnt** $= 0$ ▷**用于轮询采样的计数器 10：当任一分区中的小批次数量** > 0 **时 11：对于 i 从 0 到 p-1 执行 12：如果分区** $i > 0$ **中的小批次数量 13：avail.append(i)** ▷**待采样分区列表 14：否则 15：idle.append(i)** ▷**空闲FPGA列表 16：结束条件判断 17：结束循环 18：对于 i 从 0 到 avail.length() 执行 19：j = avail[cnt % avail.length()]20：** b **=采样(**$\mathcal{V}$**[j],** $\mathcal{E}$**[j]) 21：分发(**b**, avail[i]) 22：结束循环 23：对于 i 从 0 到 idle.length() 执行 24：j = avail[cnt % avail.length()] 25：** b **=采样(**$\mathcal{V}$**[j],** $\mathcal{E}$**[j])** ▷ **采样额外的小批次 26：分发(**b**, idle[i])** ▷**分配给空闲FPGA 27：cnt++28：结束循环 29：avail.clear() 30：idle.clear() 31：结束循环**

阶段1：在阶段1中，采样器能够从每个图分区中采样小批次。任务调度器根据小批次的采样来源图分区，将这些小批次分配给各个现场可编程门阵列。

阶段2：在阶段2中，某些图分区中的小批次已全部执行完毕。图5展示了一个示例：分区2中的所有小批次在迭代3之后都已执行完毕。因此，在迭代4中，采样器会从分区1采样一个额外的小批次，以生成3个小批次来执行同步随机梯度下降。默认情况下，额外的小批次（即小批次12）会分配给现场可编程门阵列1

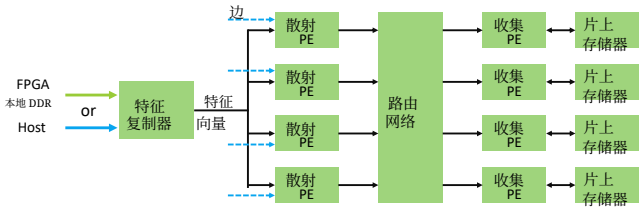

图6：聚合内核的硬件设计

，这会导致工作负载不平衡，因为现场可编程门阵列1需要在迭代4中执行2个小批次。为平衡工作负载，任务调度器会将额外的小批次分配给空闲的现场可编程门阵列；在此示例中，即现场可编程门阵列2。采样器以轮询方式对分区进行采样，因此在迭代5中，采样器会继续从分区3采样一个小批次，并生成小批次13。

我们在算法3中展示了适用于任意数量 FPGA 的通用两阶段任务调度。应用负载均衡优化后，执行的计算与原始算法相同。以图5为例，无论是否采用该优化，小批次10、11和12均在第4次迭代中执行。

5.2 数据通信

为了在多个 FPGA 上并行训练图神经网络，需要对输入图进行分区并分发到各个 FPGA。由于图结构数据的数据依赖复杂，当某个 FPGA 执行 GNN 训练时，所需数据可能不在本地分区中。在这种情况下，该 FPGA 需要从另一个 FPGA 获取数据，从而产生 FPGA 间通信。配备的 FPGA 越多，图就会被划分为越多分区，所需数据也就越有可能位于远程分区。以往研究表明，在多GPU平台上进行训练时，设备间通信开销很容易占据同步 GNN 训练时间的主要部分 [25]。

图 4 展示了 CPU+多 FPGA 平台的概览；FPGA间通信通过 CPU内存中的共享内存空间实现。具体而言，数据首先从 FPGA 存储器复制到共享内存空间，然后再传输到另一块 FPGA。与 CPU-FPGA通信相比，FPGA间通信速度慢得多，因为它需要将数据额外复制到 CPU内存 [26]。因此，我们提出直接从 CPU内存获取数据。具体而言，每当训练所需的顶点特征未存储在 FPGA 本地内存中时，FPGA 就会向主机 CPU发送请求以获取数据。从 CPU内存获取顶点特征是可行的，因为 CPU内存存储了整个图（第 4.2 节）。这种优化避免了从另一块 FPGA 读取顶点特征，从而减少了 FPGA间通信。

尽管现代 FPGA 支持通过以太网与另一块 FPGA 直接通信，但大多数云现场可编程门阵列平台（例如 Amazon Web Services（AWS） [27]和 Microsoft Azure [28]）尚不支持此功能。此外，随着平台配备的 FPGA 数量增加，从另一块 FPGA 读取数据的控制器逻辑会变得更加复杂因为控制器需要考虑 FPGA 之间连接的网络拓扑，并决定获取所需数据的路由。

5.3 优化内核库

图神经网络计算非常耗时，因为它会导致大量随机内存访问。在我们之前的工作中 [9]，我们开发了具有优化数据布局和内存组织的硬件模板，能够有效降低通信开销，从而实现较高的图神经网络训练吞吐量。尽管这些优化利用了数据并行性并提高了数据复用率，但它们不会改变图神经网络训练算法。硬件模板设计遵循算法 1 中提到的计算范式，因此能够支持多种图神经网络模型。聚合内核采用散射-聚集设计；图 6 中的聚合内核展示了一个包含 4 个散射 - 聚集处理单元（PE）的示例，每个处理单元都支持 SIMD向量并行，可在一个周期内处理多个数据。更新内核采用脉动阵列设计来执行多层感知机（MLP）。

为进一步减少开发工作量，我们提出了一个优化内核库，其中包含广泛使用的 GNN 模型（如 GraphSAGE [3]，GCN [29],等）的内核。该内核库使用户能够开箱即用地部署广泛使用的 GNN 模型。为支持不同的 FPGA 平台，GNN内核中的加速器配置采用参数化方式。用户提供平台元数据后，硬件设计空间探索引擎会自动确定这些参数（第6节）。

6 硬件设计空间探索

HitGNN 配备了一个设计空间探索引擎，可探索中央处理器+多 FPGA 平台的硬件设计空间，并自动确定加速器配置。具体而言，该引擎以小批次的配置（$\{|\mathcal{V}^l|: 0 \leqslant l \leqslant L\}$，$\{|\mathcal{A}^l|: 1 \leqslant l \leqslant L\}$）、GNN 隐藏维度 $\{f^l: 0 \leqslant l \leqslant L\}$以及平台元数据为输入，输出一组能够优化GNN训练吞吐量的参数。

6.1 资源利用率模型

资源利用率模型根据一组加速器配置，计算硬件资源消耗。在我们的内核设计中，在各类硬件资源中，数字信号处理器和查找表的使用量最大。因此，我们将查找表和数字信号处理器的使用量建模为约束：

$$\lambda_1 \times m + \lambda_2 \times n \leq N_{DSP} \tag{1}$$

$$\rho_1 \times m + \rho_2 \times n + \rho_3 \times n\log(n) \leq N_{LUT} \tag{2}$$

N_{DSP} 和 N_{LUT} 分别表示单个 FPGA 平台上可用的数字信号处理器和查找表。对于多 FPGA 平台，每个 FPGA 均独立受到公式 1 和 2 的约束。目标平台上的每个 FPGA 都必须满足所有约束，才能生成有效设计。我们用 m 表示更新内核中的处理单元（PE）数量，用 n表示聚合内核中的散射-聚集处理单元数量（第 5.3 节）。系数

λ_i $(1 \leqslant i \leqslant 2)$ 和 ρ_i $(1 \leqslant i \leqslant 3)$是表示每个处理单元资源消耗量的常数。随着实例化的处理单元数量增加，数字信号处理器的利用率呈线性增长；对于查找表，还会引入一个额外的 $n\log(n)$ 项，用于对聚合内核中路由网络的查找表开销进行建模（图 6）。

6.2 性能模型

我们将 GNN训练的吞吐量定义为每秒遍历顶点数 (NVTPS)：

$$\text{Throughput} = \frac{\sum_{i=0}^{p}\sum_{l=0}^{L}|\mathcal{V}^l|}{t_{\text{parallel}}} \tag{3}$$

分子表示一次迭代中遍历的顶点总数。对于一个多FPGA平台，p 个FPGA会并发计算 p 个小批次。分母 t_{parallel} 是一次训练迭代的并行执行时间，其中包括 p 个FPGA执行前向传播、损失计算等操作所需的时间（详情见算法2），以及梯度同步所需的时间。t_{parallel} 可建模为：

$$t_{\text{parallel}} = \max_{i \in p}\left(t^i_{\text{execution}}\right) + t_{\text{gradient_sync}} \tag{4}$$

由于有 p 个FPGA并行处理，并行执行时间受最慢的FPGA限制；同步会引入额外开销$t_{\text{gradient 同步}}$。

我们将采样阶段与图神经网络计算重叠，因此估算单个FPGA上的平均执行时间 $t_{\text{execution}}$ 为：

$$\begin{aligned} t_{\text{execution}} &= \max\left(t_{\text{sampling}}, t_{\text{GNN}}\right) \\ t_{\text{GNN}} &= t_{\text{FP}} + t_{\text{LC}} + t_{\text{BP}} \end{aligned} \tag{5}$$

其中，t_{GNN} 由前向传播 t_{FP}、损失计算 t_{LC}和反向传播 t_{BP}的执行时间组成。

总传播时间 t_{FP} 和 t_{BP} 是各层执行时间之和；由于聚合阶段和更新阶段采用流水线处理，每层的执行时间取决于完成时间较长的任务。聚合阶段包含两项任务：（1）顶点特征加载；（2）计算。由于这两项任务采用流水线处理，$t_{\text{aggregate}}$ 可建模为：

$$t^l_{\text{aggregate}} = \max(t^l_{\text{load}}, t^l_{\text{compute}}) \tag{6}$$

$$t^l_{\text{load}} = \frac{|\mathcal{V}^{l-1}|\times\beta\times f^l\times S_{\text{feat}}}{BW_{\text{DDR}}} + \frac{|\mathcal{V}^{l-1}|\times(1-\beta)\times f^l\times S_{\text{feat}}}{BW_{\text{PCIe}}} \tag{7}$$

$$t^l_{\text{compute}} = \frac{|\boldsymbol{A}^l|\times f^l}{n\times \text{PE}_{SIMD}\times \text{Freq.}} \tag{8}$$

我们将顶点特征加载时间 t^l_{load} 建模为（传输的数据量）/（有效带宽）。f^l 为特征长度，S_{feat} 为每个特征的数据大小。β 表示从存储在本地 DDR 内存中的本地图分区获取数据的比例（公式 7 的第一项）。如果数据位于远程分区，则通过 PCIe 从主机获取数据（公式 7 的第二项）。

我们将计算时间建模为（操作数）/（处理单元数 × 内核频率）。$|\boldsymbol{A}^l|$ 表示每层中的边数。n 表示实例化了 n 个散射-聚集处理单元用于聚合内核。每个处理单元都具有向量并行（第5.3节），每个周期可计算512位数据；对于单精度浮点数据，每个处理单元$_{SIMD}$ = 512/32 = 16。特征更新建模如下：

$$t_{\text{update}} = \frac{|V^l|\times f^l\times f^{l+1}}{m\times freq} \tag{9}$$

我们将 t_{update} 建模为（操作数）/（处理单元数 ×内核频率）。分子表示多层感知机的复杂度，m 表示在更新内核中实例化的处理单元数量。

算法4 硬件设计空间探索引擎

```
对于每个现场可编程门阵列执行
    对于每个骰子执行
        构建设计空间( )                    ▷ 推导 n_max, m_max
        最大值 = 0
        对于 n = 1...n_max 执行            ▷ 穷举搜索
            对于 m = 1...m_max 执行
                #使用公式（1）、（2）检查资源可用性
                有效 ←检查资源可用性(n,m)
                如果 有效且吞吐量(n, m) > max_val:
                    max_val ←吞吐量(n,m)
                    保存配置(n,m)
                结束 if
            结束 for
        结束 for
    结束 for
结束循环
```

6.3 硬件设计空间探索引擎

由于设计空间可能很大，因此探索中央处理器+多 FPGA 平台的设计空间颇具挑战。例如，我们可以分配一些 FPGA 执行特征聚合，其余 FPGA 执行特征更新；或者，也可以让每个 FPGA 都执行这两项任务，但在两个内核中实例化较低的并行度。我们的设计空间探索引擎采用后一种方案，原因有二：（1）图神经网络计算的瓶颈在于特征聚合期间的顶点特征加载，因此设计应尽可能充分利用内存带宽。也就是说，所有 FPGA（而非仅部分 FPGA）都应利用各自的内存带宽执行特征聚合；（2）如果 FPGA 能够独立执行特征聚合和特征更新，就可以直接复用中间结果，从而减少 FPGA 之间大量的数据通信。

DSE 引擎会在每个现场可编程门阵列上探索设计空间，并确定能够优化图神经网络训练吞吐量的加速器配置。此外，许多现代现场可编程门阵列由多个芯粒组成，各芯粒可用的资源可能有所不同。因此，该引擎会针对每个芯粒执行设计空间探索，以寻找最优配置。为简化起见，我们假设每个芯粒都连接到一个 DDR 通道（例如 Xilinx Alveo U250）。DSE 引擎首先根据公式（1）和（2），分别求出 n 和 m的最大值，以此构建搜索空间。随后，DSE 引擎遍历所有可能的配置组合，进行参数扫描。对于每组配置，DSE 引擎使用公式 3 评估其吞吐量，最终得出最优设计。算法4展示了 DSE 引擎执行的步骤。

7 实验

7.1 实验设置

运行环境：我们使用框架在中央处理器+多 FPGA平台上生成图神经网络训练实现，并将训练吞吐量与多GPU平台进行比较。多GPU平台和中央处理器+多 FPGA平台均构建在一台双路服务器上。多GPU平台配备了 4 个图形处理器，中央处理器+多 FPGA平台配备了 4 个 FPGA。图形处理器或 FPGA 通过 PCIe 连接到主机 CPU。表 3 列出了主机 CPU、图形处理器和 FPGA 的相关数据。请注意，FPGA 平台的峰值性能和内存带宽都远低于图形处理器平台；因此，中央处理器+多 FPGA平台上的图神经网络训练性能高度依赖于所提出的优化。多GPU基线使用 Python v3.6、PyTorch v1.11、CUDA v11.3 和 PyTorch-Geometric v2.0.3 实现。中央处理器+多 FPGA平台上的实现详见第 7.2 节。

表 3：平台规格

平台	CPU AMD EPYC 7763	GPU Nvidia RTX A5000	FPGA Xilinx Alveo U250
制程技术	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4：数据集和图神经网络层维度的统计信息

数据集	#顶点	#边	f_0	f_1	f_2
Reddit（RD）	232,965	23,213,838	602	128	41
Yelp（YP）	716,847	13,954,819	300	128	100
亚马逊 (AM)	1,569,960	264,339,468	200	128	107
ogbn-products (PR)	2,449,029	61,859,140	100	128	47

同步 GNN 训练算法：我们使用三种具有代表性的同步 GNN 训练算法来评估我们的框架：DistDGL [13], PaGraph [15], 和 P^3[12]。需要注意的是，我们仅遵循这些算法如何执行图预处理（详见第 2.3 节），并未实现原论文中的优化，因为部分优化依赖于平台，无法应用于中央处理器+多 FPGA 平台。此外，HitGNN 已经针对中央处理器+多 FPGA 平台实现了自身的优化。

GNN 模型和数据集：我们在两个广为人知的 GNN 模型上评估我们的框架：GraphSAGE [3] 和 GCN[29]。我们采用一个 2 层模型，隐藏特征维度为 128，每个小批次中的目标顶点数量 $|\mathcal{V}^t|$ 设为 1024，并将每层的邻居采样数量分别设为 25 和 10；用于评估的 GNN 参数已得到广泛应用，并在许多研究中展现出良好效果 [3], [6], [7], [29], [30]。我们选择了四个广泛使用且边数超过一千万的数据集进行评估：Reddit、Yelp、亚马逊 [6] 和 ogbn-products [30]。表4列出了数据集和 GNN 层维度的详细信息。

7.2 框架实现

HitGNN 由多个构建模块组成，用于生成可在中央处理器+多 FPGA 异构平台上运行的设计。程序解析器、设计空间探索引擎、软件生成器和硬件生成器使用 Python v3.6 实现，加速器模板则使用 Xilinx Vitis HLS v2021.2 实现。主机程序模板采用 C++14和 OpenCL 库编写。用户可通过所提供的 API（第 4.3 节）与 HitGNN 交互。在代码清单 2 中，我们给出了在实验中实现三种同步 GNN 训练算法的示例；在代码清单3中，我们展示了部分生成的主机程序和可综合加速器设计。 P^3 需要在第一个 GNN 层末尾额外执行一步全对全广播；我们将这一步视为特殊情况，为简化设计，不提供用于处理该步骤的 API。在代码清单3的第 14–19 行中，我们展示了如何使用 OpenCL 库的内置函数轻松实现这一步。

```
### DistDGL 预处理示例 ###
2.3 "" 4 在 DistDGL 中运行多约束 METIS，以生成 5 V [n], E [n]和
X [
8 图
例
度
配
每
for
i)#
samp = 打开('GraphSAGE_采样器')
图神经网络_comp = 图神经网络_计算('GraphSAGE')图神经网络_
```

清单 2：三个同步 GNN 训练算法的示例程序

```
//主机程序的一部分cl::Device devices =
for(int iter = 0; iter <层; iter++) { // 变量"层"由用户指定 buffer_
```

```
19 q.enqueueMigrateMemObjects(layer2in_ii): 20 } 21 } 22
q.f
果
str
inc
res
HL
reg
数
```

代码清单3：生成代码示例

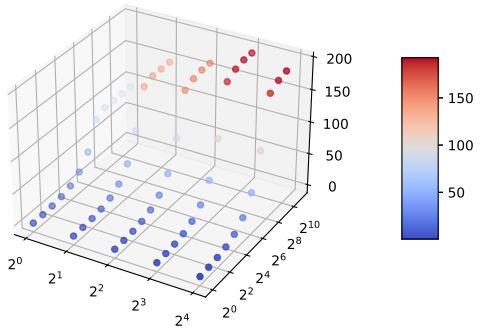

图7：GraphSAGE 的设计空间探索结果

7.3 DSE引擎评估

给定一个 GNN 模型、采样算法和中央处理器+多现场可编程门阵列平台元数据，设计空间探索引擎会自动确定加速器配置。我们用n 表示聚合内核中散射-聚集处理单元的数量，用 m 表示更新内核中的处理单元数量（第6节）。我们按照算法4进行参数扫描，以探索FPGA设计空间。对于每个设计点（n,m），设计空间探索引擎会估算四个数据集上的GNN训练吞吐量（表4），并报告平均吞吐量。图7展示了一个 GraphSAGE 模型的 DSE 结果示例。GCN 模型的 DSE 也呈现出类似的结果。

由于设计空间探索引擎会系统地搜索硬件设计空间，因此能够找到最优加速器配置。如表5所示，加速器配置 (8,2048) 和 (16,1024) 均已用尽可用硬件资源。由于特征聚合通常是 GNN训练的瓶颈，直观来看，人们可能会选择 (16,1024)，以最大化聚合内核的并行度。然而，正如设计空间探索引擎所示，配置 (8,2048) 反而能带来更高的训练吞吐量；这是因为我们优化后的内核有效降低了特征聚合的通信开销，并将瓶颈转移到了特征更新阶段。因此，配置 (8,2048) 将更多硬件资源投入更新内核，从而实现更高的吞吐量。

表5：资源利用率和并行度

并行度（n,m）	(8,2048)	(16,1024)
LUTs	72%	65%
DSPs	90%	56%
URAM	48%	34%
BRAM	40%	28%
估计吞吐量（每秒新词元数）	97.0 M	92.6 M

7.4 性能指标

- **周期时间**：训练一个周期所需的时间（秒）。
- **吞吐量**：我们将训练吞吐量定义为每秒遍历顶点数（NVTPS）。
- **带宽效率**：吞吐量除以目标平台可用的内存带宽（NVTPS/(GB/s)）。图神经网络训练吞吐量高度依赖平台可用的内存带宽；由于不同平台的内存带宽各不相同，使用可用带宽对吞吐量进行归一化，可以在不同平台之间进行公平比较。

7.5 跨平台比较

我们将 HitGNN 生成的、运行于中央处理器+多 FPGA 平台上的一组设计，与使用 PyTorch-Geometric [20]、运行于多GPU平台上的先进图神经网络训练实现进行比较。中央处理器+多 FPGA 平台包含四个 FPGA，而多GPU平台包含四个 GPU（第 7.1 节）。我们在表6中展示结果，并使用第 7.4 节定义的性能指标进行比较。我们使用*GPU* 表示多 GPU基线，使用我们的方法 表示 HitGNN 生成的设计；使用 *GSG* 表示 Graph-SAGE [3]模型，使用图卷积网络 表示 GCN [29] 模型。与多GPU基线相比，HitGNN 分别在 DistDGL、PaGraph 和 P^3 上实现了 2.11×、2.28× 和 2.34× 倍加速比。这是因为 HitGNN 采用了能够在各 FPGA 之间均衡工作负载并减少 FPGA间通信开销的优化。为评估这些优化的有效性，我们开展了消融研究。我们首先评估基线设计的性能，然后逐步向该设计中加入工作负载均衡（WB）优化和数据通信（DC）优化。我们在表7中展示评估结果，并以 DistDGL算法为例；两项优化总计可带来最高 66% 的吞吐量提升。此外，我们高度优化的图神经网络内核使 HitGNN 相较于多GPU平台，在几何平均值方面实现了 13.4×-14.9 × 倍的带宽效率。得益于卓越的带宽效率，HitGNN 仅使用多GPU平台 0.16× 的内存带宽，即可实现最高 4.26× 倍加速比。

7.6 可扩展性

我们构建了一个中央处理器+多FPGA平台模拟器，用于评估 HitGNN 的可扩展性。该模拟器根据同步 GNN 训练算法、GNN 模型和平台元数据来估算训练性能。为了验证

表6：跨平台比较

		数据集	Reddit		Yelp		亚马逊		ogbn-products		几何平均值
		模型	GCN	GSG	GCN	GSG	GCN	GSG	GCN	GSG	
DistDGL [13]	周期时间（秒）	GPU	1.29	1.34	1.75	1.79	4.19	4.34	5.03	5.38	-
		Ours	0.62	0.77	0.63	0.87	1.14	1.72	3.06	4.31	-
	吞吐量（每秒新词元数）	GPU	15.6 M	15.1 M	21.6 M	21.1 M	22.6 M	21.8 M	97.5 百万	91.2 百万	28.8 百万 (1.00×)
		Ours	32.5 百万	26.2 百万	59.9 百万	43.4 M	83.1 M	55.1 M	160 M	114 M	60.7 M (2.11×)
	带宽效率（每秒新词元数/(GB/s)）	GPU	4.77 K	4.59 K	6.58 K	6.44 K	6.90 K	6.66 K	29.8 K	27.8 K	8.78 K (1.00×)
		Ours	63.4 K	51.1 K	117 K	84.6 K	162 K	107 K	313 K	222 K	118 K (13.4×)
PaGraph [15]	周期时间（秒）	GPU	1.21	1.25	1.66	1.69	4.04	4.16	4.61	4.89	-
		Ours	0.53	0.67	0.54	0.77	0.98	1.53	2.64	3.82	-
	吞吐量（每秒新词元数）	GPU	16.7 M	16.1 M	22.7 M	22.3 M	23.4 M	22.8 M	106 M	100 M	30.6 M（1.00×）
		Ours	38.1 M	30.1 M	69.9 M	49.0 百万	96.6 百万	61.9 百万	186 百万	128 百万	69.8 百万 (2.28×)
	带宽效率（每秒新词元数/(GB/s)）	GPU	5.09 K	4.92 K	6.94 K	6.82 K	7.15 K	6.95 K	32.5 K	30.6 K	9.35 K (1.00×)
		Ours	74.2 K	58.7 千	136 千	95.6 千	188 千	121 千	362 千	250 K	136 K (14.6×)
P^3 [12]	周期时间（秒）	GPU	1.24	1.32	1.82	1.79	4.43	4.41	5.21	5.43	-
		Ours	0.53	0.70	0.55	0.82	1.04	1.70	2.71	4.13	-
	吞吐量（每秒新词元数）	GPU	16.3 百万	15.3 百万	20.7 百万	21.1 百万	21.4 百万	21.5 百万	94.2 百万	90.4 百万	28.4 百万 (1.00×)
		Ours	38.1 百万	28.8 百万	68.6 百万	46.0 M	91.1 M	55.7 M	181 M	119 M	66.4 M (2.34×)
	带宽效率（每秒新词元数/(GB/s)）	GPU	4.96 K	4.66 K	6.33 K	6.44 K	6.52 K	6.55 K	28.7 K	27.6 K	8.67 K (1.00×)
		Ours	74.2 K	56.2 K	134 K	89.7 K	178 K	109 K	353 K	232 K	129 K（14.9×）

表 7：优化带来的吞吐量提升

数据模型	基线	WB	WB+DC	加速比
RD-GCN	19.9 M	22.7 M	32.5 M	63%
RD-GSG	16.9 M	19.2 M	26.2 M	55%
YP-GCN	36.4 M	41.9 M	59.9 M	65%
YP-GSG	28.5 M	32.8 M	43.4 M	52%
AM-GCN	50.8 M	59.6 M	84.1 M	64%
AM-GSG	36.5 M	42.9 M	55.1 M	51%
PR-GCN	96.7 M	113 M	160 M	66%
PR-GSG	73.8 M	86.5 M	114 M	54%

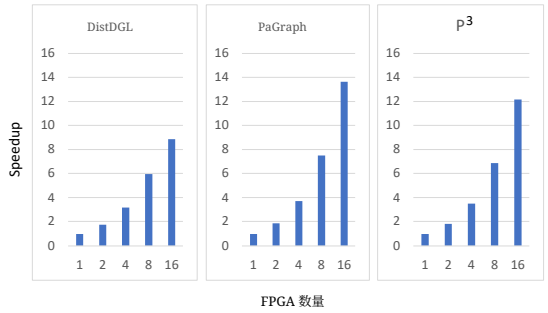

图 8：可扩展性评估

该模拟器，我们首先实现主机程序和硬件内核。然后，测量主机程序的执行时间以及综合后内核的执行时间，以微调模拟器。图 8 展示了各算法相较于单个FPGA的加速比。通过减少 FPGA 间通信开销并均衡工作负载，HitGNN 实现了可扩展加速。HitGNN 的可扩展性受中央处理器内存带宽限制。这是因为当 FPGA 本地内存中没有所需数据时，FPGA 会从中央处理器内存中获取数据（第 5.2 节）。以 EPYC 7763 中央处理器为例，其中央处理器内存（带宽 = 205GB/sec.）

在不使 CPU 内存带宽饱和的情况下，最多可支持 205/16 = 12.8 个 FPGA，其中分母表示通过 PCIe 实现的单个 CPU-FPGA连接的带宽；如果向平台添加更多 FPGA，由于 CPU 内存带宽逐渐饱和，加速比的可扩展性便会开始下降。图 8 展示了 HitGNN 的吞吐量几乎能够线性扩展至 16 个 FPGA。

8 结论

本文提出了 HitGNN，这是一种通用框架，可在给定的中央处理器+多 FPGA异构平台上生成高吞吐量的 GNN 训练实现。HitGNN 提供了多种优化，可加速中央处理器+多 FPGA平台上的同步 GNN 训练。HitGNN 的优化具有足够的通用性，可应用于各种同步 GNN 训练算法；此外，这些优化不会改变训练算法，因此不会影响模型精度或收敛速度。与最先进的多GPU基线相比，HitGNN 生成的实现可实现高达 27× 的带宽效率，因此使用远低于多GPU基线的计算能力和内存带宽，即可达到高达 4.26× 的吞吐量。我们展示了 HitGNN 可扩展至 16 个 FPGA 并实现可扩展加速，其限制因素是 CPU内存带宽。未来，我们计划利用数据预取技术和高速 FPGA互连网络（例如 SmartNIC）缓解 CPU内存带宽的压力，进一步提升 HitGNN 的可扩展性。

致谢

本研究得到美国国家科学基金会（NSF）SaTC-2104264 和 OAC-2209563 项目，以及 DEVCOM陆军研究实验室（ARL）W911NF2220159 项目的资助。

参考文献

[1] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, 《面向网络规模推荐系统的图卷积神经网络》，发表于第*24*届*ACM*知识发现与数据挖掘国际会议论文集，2018。

[2] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, 《Aligraph：一个综合性图神经网络平台》，《VLDB Endowment 会议录》，2019。[3] W. L. Hamilton, R. Ying, and J. Leskovec, 《大规模图上的归纳式表示学习》，发表于第31届神经信息处理系统进展国际会议论文集，2017。[4] C.-I. Yang and Y.-P. Li, 《基于深度学习的分子性质预测中的可解释不确定性量化》，《化学信息学杂志》，2023。[5] W. Jiang and J. Luo, 《用于交通流预测的图神经网络：综述》，arXiv预印本 arXiv:2101.11174，2021。[6] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, 《GraphSAINT：一种基于图采样的归纳学习方法》，发表于国际学习表征会议，2020。[7] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, 《解耦图神经网络的深度与范围》，发表于神经信息处理系统进展，2021。[8] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, 《Cluster-GCN：一种训练深层大规模图卷积网络的高效算法》，发表于ACM SIGKDD知识发现与数据挖掘国际会议，2019。[9] Y.-C. Lin, B. Zhang, and V. Prasanna, 《HP-GNN：在中央处理器-现场可编程门阵列异构平台上生成高吞吐量GNN训练实现》，发表于2022年ACM/SIGDA现场可编程门阵列国际研讨会论文集，2022。[10] Z. Que, M. Loo, H. Fan, M. Blott, M. Pierini, A. D. Tapper, and W. Luk, 《LL-GNN：用于粒子探测器的现场可编程门阵列低延迟图神经网络》，arXiv预印本 arXiv:2209.14065，2022。[11] Y. C. Lin, B. Zhang, and V. Prasanna, 《利用高级综合加速图卷积网络推理》，发表于2021年IEEE高性能极限计算会议（HPEC），2021。[12] S. Gandhi and A. P. Iyer, 《P3：大规模分布式深度图学习》，发表于第15届USENIX操作系统设计与实现研讨会（OSDI 21），2021。[13] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, 《DistDGL：面向十亿规模图的分布式图神经网络训练》，发表于IEEE/ACM非规则应用研讨会：架构与算法，2020。[14] L. Zhang, Z. Lai, S. Li, Y. Tang, F. Liu, and D. Li, 《2PGraph：在图形处理器集群上加速大规模图的GNN训练》，发表于2021年IEEE集群计算国际会议（CLUSTER），2021。[15] Z. Lin, C. Li, Y. Miao, Y. Liu, and Y. Xu, 《PaGraph：通过计算感知缓存扩展大规模图上的GNN训练》，发表于第11届ACM云计算研讨会论文集，2020。[16] J. Yang, D. Tang, X. Song, L. Wang, Q. Yin, R. Chen, W. Yu, and J. Zhou, 《GNNLab：面向图形处理器上基于采样的GNN训练的分解式系统》，发表于第17届欧洲计算机系统会议论文集，2022。[17] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, 《HyGCN：采用混合架构的图卷积网络加速器》，发表于2020年IEEE高性能计算机体系结构国际研讨会（HPCA），2020。[18] S. Li et al., 《面向大规模分布式图神经网络的超大规模FPGA即服务架构》，发表于第49届年度计算机体系结构国际研讨会论文集，2022。[19] H. Zhang, Z. Yu, G. Dai, G. Huang, Y. Ding, Y. Xie, and Y. Wang, 《理解GNN计算图：协同计算、I/O与存储器视角》，发表于《机器学习与系统论文集》，2022。[20] M. Fey and J. E. Lenssen, 《利用PyTorch Geometric快速进行图表示学习》，发表于ICLR图与流形表示学习研讨会，2019。[21] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, 《重新审视分布式同步随机梯度下降》，发表于国际学习表征会议研讨会，2016。[22] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Xu, R. Patel, and M. Herbordt, 《FPDeep：在FPGA集群上加速CNN训练并实现负载均衡》，发表于2018年IEEE第26届现场可编程定制计算机年度国际研讨会（FCCM），2018。[23] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Patel, and M. Herbordt, 《一种在深度流水线FPGA集群上加速CNN训练的框架：采用工作量与权重负载均衡》，发表于现场可编程逻辑与应用国际会议（FPL），2018。

[24] Y.-C. Lin、B. Zhang 和 V. Prasanna，“在中央处理器+多 FPGA 异构平台上加速 GNN 训练”，收录于《高性能计算》。Springer International Publishing，2022年。[25] Z. Cai、X. Yan、Y. Wu、K. Ma、J. Cheng 和 F. Yu，“Dgcl：用于分布式图神经网络训练的高效通信库”，收录于《第十六届欧洲计算机系统会议论文集》，EuroSys ’21 系列，2021年。[26] Y.-K. Choi、J. Cong、Z. Fang、Y. Hao、G. Reinman 和 P. Wei，“现代异构 CPU-FPGA 平台微体系结构的深入分析”，*ACM* 重构技术系统汇刊 [27] “亚马逊 EC2 f1 [在线],” https://aws.amazon.com/tw/ec2/inst ance-types/f1/, 访问日期：2022-06-23。[28] “Azure NP 系列 [在线],” https://docs.microsoft.com/en-us/ azure/virtual-machines/np-series, 访问日期：2022-06-23。[29] T. N. Kipf 和 M. Welling，“利用图卷积网络进行半监督分类”，收录于《国际学习表征会议》，2017年。[30] W. Hu、M. Fey、M. Zitnik、Y. Dong、H. Ren、B. Liu、M. Catasta 和 J. Leskovec，“开放图基准：图上的机器学习数据集”，*arXiv* 预印本 *arXiv:2005.00687*，2020年。



Yi-Chien Lin于2020年获得国立台湾大学电气工程理学学士学位，目前是南加州大学电气工程专业的博士生。他获得了谢明电气与计算机工程系维特比工程学院研究生奖学金。他的研究兴趣包括异构计算、硬件加速和图机器学习。



Bingyi Zhang 于2017年获得复旦大学微电子学理学学士学位，并获得复旦大学集成电路工程理学硕士学位。他目前在南加州大学（USC）攻读计算机工程博士学位。他的研究兴趣包括并行计算、数字信号处理和数字电路设计，主要研究如何利用现场可编程门阵列平台加速基于图的机器学习。



Viktor K.Prasanna 获得班加罗尔大学电子工程理学学士学位、印度科学研究所自动化学院理学硕士学位，以及宾夕法尼亚州立大学计算机科学博士学位。他是南加州大学（USC）谢明电气与计算机工程系查尔斯·李·鲍威尔工程学讲席教授，同时担任计算机科学教授。他的研究兴趣包括高性能计算、并行

行和分布式系统、可重构计算以及嵌入式系统。他是南加州大学—印孚瑟斯高级软件技术中心（CAST）的执行主任，曾任南加州大学雪佛龙交互式智能油田技术卓越研究与学术培训中心（Cisoft）副主任。他还担任南加州大学能源信息学中心主任。2003年至2006年，他担任《IEEE计算机汇刊》主编。目前，他担任《并行与分布式计算杂志》主编。他是IEEE计算机学会并行处理技术委员会的创始主席。他担任IEEE国际并行与分布式处理研讨会（IPDPS）指导委员会主席，以及IEEE高性能计算国际会议（HiPC）指导委员会主席。他荣获宾夕法尼亚州立大学颁发的2009年杰出工程校友奖。2015年，他因在可重构计算方面的贡献荣获IEEE计算机学会颁发的W. Wallace McDowell奖。他在正则表达式匹配方面的研究成果于2013年获评FCCM创办20年来最具影响力的论文之一。他是IEEE、ACM和美国科学促进会（AAAS）会士，也是欧洲科学院成员。