DOI: 10.16452/j.cnki.sdkjzk.2026.02.010 文章编号:1672-3767(2026)02-0106-11

采样算子融合与内存优化协同驱动的图神经网络并行训练加速

李欣嵘[1],韩承磊[2],于建志[1,3],胡克坤[4],梁建国[5],董文博[6]

(1. 山东科技大学 计算机科学与工程学院,山东 青岛 266590; 2. 山东大学 集成电路学院,山东 济南 250101;
3. 浪潮计算机科技有限公司,山东 济南 250101; 4. 浪潮电子信息产业股份有限公司,山东 济南 250101;
5. 曲阜师范大学 计算机学院,山东 日照 276826; 6. 青岛市黄岛区中心医院,山东 青岛 266555)

摘 要:为了缓解大规模图神经网络(GNN)训练面临的内存占用和计算开销过大问题,基于采样技术的 GNN 训练得到广泛应用,但其采样和特征传输阶段效率低下,限制了模型的训练速度和扩展性。为此,将 GNN 点采样训练过程进一步分为采样、特征提取、前向传播和反向传播四个阶段,通过评测发现采样和特征提取是主要性能瓶颈。本研究在多图形处理器(GPU)环境下提出一系列优化方法:在采样阶段,设计 toBlockFast 算子,并与 SampleNeighbors 算子融合,提升采样效率;在特征传输阶段,提出一种基于固定锁页内存的优化方法,结合多进程与多流的并行传输技术,实现特征传输与前向计算的高效并行化。实验结果表明,在单 GPU 环境下,与现有的 DGL 方法相比,采样阶段可实现 1.52 倍的加速;进一步优化特征传输阶段后,总体训练效率提升至 1.80 倍。此外,在 2、4 和 8 个 GPU 环境下,本研究分别获得了 1.10、1.16 和 1.12 倍的性能加速。

关键词:图神经网络训练;点采样;算子融合;特征传输;性能优化

中图分类号:TP311 **文献标志码**:A

Synergizing operator fusion and memory optimization for parallel GNN training acceleration

LI Xinrong[1], HAN Chenglei[2], YU Jianzhi[1,3], HU Kekun[4], LIANG Jianguo[5], DONG Wenbo[6]

(1. College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China; 2. School of Integrated Circuits, Shandong University, Jinan 250101, China;
3. Inspur Computer Technology Co.,Ltd., Jinan 250101, China; 4. IEIT SYSTEMS Co., Ltd., Jinan 250101, China;
5. School of Computer Science, Qufu Normal University, Rizhao 276826, China;
6. Qingdao Huangdao District Central Hospital, Qingdao 266555, China)

Abstract: To alleviate the problems of large memory occupation and high computational overhead in graph neural networks (GNN) training on large-scale graphs, sampling-based GNN training methods have been widely applied. However, the inefficiencies in the sampling and feature transfer stages limit the training speed and scalability of the models. In this paper, the sampling training process was divided into four stages: sampling, feature transfer, forward propagation, and backward propagation. After evaluation, the sampling and feature transfer stages were found to be the primary performance bottlenecks. In multi-GPU environments, this study proposed a series of optimization methods. In the sampling stage, the toBlockFast operator was designed and integrated with the SampleNeighbors operator to enhance the sampling efficiency. In the feature transfer stage,

收稿日期:2025-04-11

基金项目:山东省自然科学基金创新发展联合基金项目(ZR2023LZH009);山东省先进计算重点实验室开放课题资助项目(2025LCJSJ0004)

作者简介:李欣嵘(1998—),女,河南南阳人,硕士研究生,主要从事高性能计算研究.
梁建国(1981—),男,山西霍州人,副教授,博士,主要从事高性能计算研究,本文通信作者.
E-mail:183549746@qq.com

an optimization scheme based on pinned memory was proposed and combined with multi-process and multi-stream parallel transfer techniques to achieve efficient parallelization of feature transfer and forward computation. Experimental results show that, in a single GPU enviroment, compared to the existing deep graph library (DGL) methods, the proposed approach achieves a 1.52 times speedup in the sampling stage, and that the overall training efficiency improves by 1.80 times after the further optimization of the feature transfer stage, In a multi-GPU environment, our solution achieves performance improvements of 1.10, 1.16, and 1.12 times on 2, 4, and 8 GPU setups, respectively.

Key words: graph neural network training; node sampling; operator fusion; feature transfer; performance optimization

图神经网络(graph neural networks,GNN)[1-4]将深度学习从传统的非结构化数据处理(如图像识别)拓展至结构化数据领域。图神经网络在顶点分类[1-3]、链接预测和关系提取[5-6]等图结构数据处理方面具有独特优势。然而,随着现实应用中图数据规模的急剧增长(如 Facebook 好友图包含 7 亿个顶点和超过 1 000 亿条边)以及节点和边的日益高维度化(特征维度通常达到 300～600 维),传统全图训练方法面临严峻挑战,在有限的时间和计算资源约束下,直接对整个大规模图进行端到端的训练不仅计算开销巨大,甚至可能因内存限制而无法实现,如何高效训练大规模 GNN 模型成为当前亟待解决的问题。

为了训练大规模 GNN,研究者提出了基于图采样的高效训练方法[7-9],其核心思想是通过从原始大规模图中动态生成子图,并将其作为小批量输入 GNN 模型进行训练,将原本难以处理的全图训练任务转化为可并行计算的子图训练任务。基于图采样的训练方法在降低单次计算资源消耗的同时,可确保模型训练的有效收敛。

根据采样策略的不同,图采样算法主要分为三类:点采样、层采样和子图采样。目前的大多数研究主要关注点采样算法和训练方法的并行优化。在点采样方面,GraphSAGE[10](graph sample and aggregated)对目标节点随机采样固定数目的邻居节点,确保每批次计算量的稳定;VR-GCN[11](vectorized relational graph convolution network)则利用节点历史信息减少采样的邻居数量,从而进一步降低计算开销。在训练方法优化方面,DGL[12](deep graph library)通过数据并行采样实现了多图像处理器(graphics processing unit,GPU)上的图神经网络训练;PaGraph[13]提出了计算感知分区算法,减少了跨分区数据访问的开销;DistDGL[14]利用 METIS 图划分算法提高了数据的局部性,从而减少网络通信的成本。此外,GNNLab[15]通过多 GPU 分解设计,解决了特征缓存和采样之间的资源竞争问题;SCGraph[16]采用 NVLink 优化了 GPU 间的数据传输效率;Song 等[17]基于性能模型提出了图数据的多 GPU 分配方案,并通过局部感知采样技术进一步缩短了采样耗时。

综上,现有研究主要从两个维度展开:一是图采样算法的高效性研究,重点解决大规模图数据的子图提取问题;二是训练方法的优化研究,着重提升多 GPU 和分布式环境下的计算性能。然而,这些工作对 GNN 训练过程中各阶段的性能特征缺乏深入分析,特别是在采样和特征处理环节仍存在优化空间。

在众多点采样模型中,GraphSAGE 凭借其简单有效的邻居采样和特征聚合机制,在不同类型的图数据集上取得较好的实验效果,并且其模型结构易于理解和实现,便于在多 GPU 环境下进行优化实验。如在处理大规模社交网络图时,与其他模型相比,GraphSAGE 能够更高效地生成节点表示,为后续的分类、链接预测等任务提供更优质的特征输入。同时,与其他复杂模型相比,在计算资源有限的情况下,仍能保持较好的性能。因此,本研究选择 GraphSAGE 模型作为实验对象,验证所提优化方法的有效性。

基于上述原因,首先对 GNN 训练流程进行细粒度划分,将其分解为采样、特征提取、前向传播和反向传播四个关键阶段。通过系统地性能分析发现,采样和特征提取阶段在实际训练中占据了主要时间开销。针对 DGL 框架下采样阶段存在的 toBlock 算子耗时高及不同算子间数据转换开销大的问题,设计 toBlockFast 算子并与 SampleNeighbors 算子融合,以提升采样效率。针对特征提取阶段存在的特征传输耗时大问题,提出一种基于固定锁页内存的优化方法,同时结合多进程和多流技术实现特征提取与前向计算的高效并行化。

1 相关技术

1.1 图采样训练

GNN 采样训练主要分为采样、特征提取、前向传播和反向传播四个阶段。

1) 采样:首先随机选取一定数量的训练节点,然后针对这些节点执行采样算法,生成采样子图。

2) 特征提取:首先根据采样子图中的节点索引,从节点特征中提取对应的节点特征,然后将采样子图拓扑数据、节点特征和标签和一起通过 PCIe(peripheral component interconnect express)总线传输至 GPU。

3) 前向传播:前向传播分为聚合和更新两个操作。在聚合操作中,节点需要将其邻居特征聚合成一个特征向量;在更新操作中,将聚合后的邻居特征表示与当前节点进行拼接,再经过非线性激活函数更新节点的特征表示。

4) 反向传播:首先根据节点前向传播的结果及其标签值计算损失,然后根据链式法则计算出模型参数的梯度,最后更新模型参数。

GNN 采样训练流程如图 1 所示。本研究对 GNN 采样训练的采样阶段和特征提取阶段进行性能分析与优化,将前向传播与反向传播合并为模型训练,并将特征提取阶段细分为特征查找、特征传输以及子图传输操作。

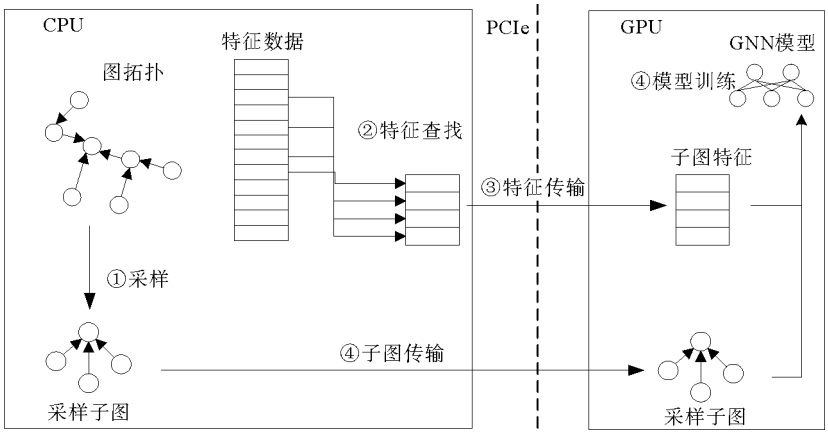


图 1 GNN 采样训练流程图

Fig. 1 Graph neural network sampling training process diagram

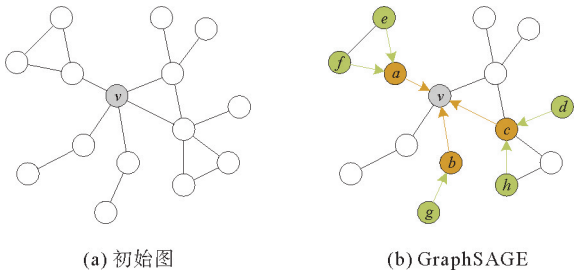
1.2 GraphSAGE 模型介绍

本研究基于点采样算法的 GraphSAGE 模型开展研究,该模型主要通过以下三种操作实现对节点特征的更新。

1) 邻居采样:对于每个节点 v,在每一 k 层中,从其邻居节点集合 $N(v)$ 中采样固定数量的节点。

2) 聚合:在每一 k 层中,对节点 v 采样得到的邻居节点 a、b、c 进行聚合,得到聚合后的邻居表示。具体聚合操作如图 2 所示。

3) 更新:将聚合后的邻居表示与当前顶点 v 进行拼接,再经过非线性激活函数更新节点 v 的表示。其具体算式为:



(a) 初始图 (b) GraphSAGE

图 2 GraphSAGE 模型中聚合操作

Fig. 2 Aggregation operations for GraphSAGE model

$$h_v^k=\sigma(\boldsymbol{W}^k\cdot \mathrm{Concat}(h_v^{k-1}\cdot \mathrm{Aggregate}(\{h_u^{k-1}\mid u\in N_k(v)\}))) 。 \tag{1}$$

式中:$N_k(v)$ 是第 k 层对节点 v 的邻居节点采样后的集合;Aggregate 表示聚合函数(如均值聚合、LSTM 聚合或池化聚合);Concat 表示拼接操作;$\boldsymbol{W}^k$ 是第 k 层的权重矩阵;$\sigma(\cdot)$ 表示非线性激活函数。

2 图神经网络采样训练性能瓶颈测试与分析

2.1 实验环境及设置

1) 硬件环境:CPU 为两颗 Intel Xeon E5-2690 CPU;GPU 为 8 块 NVIDIA K80 显卡,每块显卡内存为 12 GB。

2) 软件环境:Python 版本为 3.8.16;Pytorch 版本为 1.9.1;CUDA 驱动版本为 10.2;DGL 版本为 0.9.1。

3) GNN 模型:使用 GraphSAGE 模型,其中采用 MEAN 聚合器,模型优化器为 Adam[18],学习率设置为 0.01,采样层数的 fan_out 参数为[10,10,25]。

4) 数据集:实验数据集选取 Reddit、Products、Papers100M 和 Yelp 四个大规模数据集,表 1 展示了实验数据集的详细信息。

2.2 性能评测分析

表 1 实验数据集

Table 1 Experimental datasets

数据集	节点数	边数	平均度数	特征数
Reddit	232 965	114 848 857	985.98	602
Products	2 449 029	61 859 140	101.03	100
Papers100M	111 059 956	1 615 685 872	29.10	128
Yelp	537 635	7 949 403	29.57	300

本节在 1、2、4 和 8 个 GPU 时评测图神经网络采样训练的性能表现,分析其性能瓶颈。

1) 采样训练整体耗时分析实验。针对表 1 中不同数据集,在单 GPU 设置下,整个训练过程中采样阶段和特征提取阶段的耗时占比最多;随着 GPU 数量的增加,采样阶段与特征提取阶段耗时占总训练时间的比重始终保持较高。这表明采样阶段与特征提取阶段是采样训练的性能瓶颈。

2) 采样阶段耗时分析实验。采样阶段主要负责邻居采样和采样子图的生成,其中邻居采样主要通过 SampleNeighbors 算子完成,采样子图的生成则由 toBlock 算子实现。为探究采样阶段的性能瓶颈,对该阶段不同算子的耗时进行测量。针对表 1 中不同数据集,在单 GPU 设置下,toBlock 算子的耗时占采样阶段总耗时的比例最高,分别为 64%、80%、60%和 76%;当 GPU 数量增加到 2、4 和 8 时,toBlock 算子的耗时占比分别为 70%~80%、54%~80%和 40%~71%。同时,SampleNeighbors 算子的耗时占比也随着 GPU 数量增加略有上升,在 1、2、4 和 8 个 GPU 的实验设置下,其耗时占采样阶段总耗时的 20%~30%。这表明,在优化 toBlock 算子的同时,减少 SampleNeighbors 算子的耗时对于提升采样阶段性能同样至关重要。

3) 特征提取阶段耗时分析实验。为探究特征提取阶段的性能瓶颈,对该阶段的特征查找、特征传输以及子图传输操作的耗时进行测量。针对表 1 中不同数据集,在使用 1、2、4 和 8 个 GPU 的实验下,特征提取阶段的时间主要集中在特征传输操作上,其耗时在特征提取阶段总耗时的占比分别为 58%~76%、57%~75%、58%~80%和 59%~84%。因此,在采样训练中提升特征提取阶段性能的关键在于优化特征传输操作。

3 图神经网络采样训练性能优化方案

根据上节分析,GNN 采样训练的性能瓶颈主要集中在采样和特征传输阶段。为提升多 GPU 的训练效率,对上述两个阶段进行优化:优化采样阶段的 toBlock 算子,设计 toBlockFast 算子并与 SampleNeighbors 算子融合;在特征传输阶段,采用固定锁页内存提高传输效率,并结合多进程与多流并行传输技术,实现高效并行化。

3.1 基于数据复用的 toBlockFast 算子

3.1.1 toBlock 算子分析

toBlock 算子主要负责将 SampleNeighbors 算子进行邻居采样后生成的稀疏子图转化为采样子图,生成的采样子图主要包含稀疏子图中的连接边以及边对应的节点。toBlock 算子主要包括以下四个操作。

1) HashMap 构建:根据输入节点序列构建两个相同的 HashMap,HashMap1 中存放源节点,Hash-

Map2 中存放目标节点;HashMap 的键为节点编号,值为键值对的个数。

2) 更新 HashMap2:将边数据中的目标节点对 HashMap2 进行插入操作,当目标节点不存在时将该节点插入 HashMap2 中。

3) 节点映射:根据边数据中源节点获取其在 HashMap2 中对应值作为源节点的新编号,根据边数据中目标节点编号获取其在 HashMap1 中对应值作为目标节点的新编号。

4) 生成子图:根据节点编号的映射关系生成 COO 格式(coordinate format)子图拓扑数据。

toBlock 算子耗时主要在构建、更新以及映射三个操作中。构建本质上是对一个空的 HashMap 序列进行更新操作,因此构建与更新操作原理相同。更新操作主要通过查询 HashMap 来确定当前节点是否在 HashMap 中。在 DGL 中使用开放定址法处理哈希冲突,需要进行多次索引计算获取元素实际的存储位置。因此,提高 toBlock 算子效率的关键是减少查询操作的额外开销。

3.1.2 toBlock 算子优化

通过引入位图(BitMap)和双位图(double BitMap)来提高稀疏子图到采样子图的转换效率,减少转换过程中的冗余开销,设计 toBlockFast 算子,对 toBlock 算子进行优化。toBlockFast 算子的伪代码如算法 1 所示,具体步骤为:

1) 引入 BitMap 快速判断节点是否存在,避免在更新 HashMap 时查询操作的高开销。引入 BitMap 后,节点的存在性检查可以通过直接访问位图中的对应位实现,节点判断的复杂度降低到 $O(1)$。

2) 利用双位图优化 toBlock 算子中 HashMap 的构建与更新。引入双位图 BitMap1 和 BitMap2 分别记录源节点和目标节点,HashMap 的具体构建和更新过程分别如图 3(b)和图 3(c)所示,此时 toBlock 算子中 HashMap 的数量由两个减少至一个,相应减少了 HashMap 的创建和查询开销。

3) 映射操作和更新操作的并行。toBlock 算子的更新操作和映射操作存在数据依赖,引入双位图后,目标节点的映射仅需查询 BitMap1,源节点的更新修改 BitMap2 和 HashMap,两个操作之间的数据依赖消除,可以并行执行。将整个执行域划分为两个并行域,一个并行域使用 BitMap1 执行目标节点的映射操作,另一个使用 BitMap2 执行更新操作。

4) 合并源节点的更新和映射过程,减少对源节点数组和 HashMap 的重复访存开销。当节点已存在 BitMap2 中时,可直接从 HashMap 获取节点的值,否则先计算节点的映射值,再更新 BitMap2 和 HashMap。

算法 1 toBlockFast 算子

输入:输入节点 input_node,稀疏子图拓扑 graph,初始化标识 flag
输出:采样子图 s_graph,输出节点 output_node

```
if flag then                                  // 获取 HashMap
    map← data_reuse()
else
    map← create_HashMap(input_node)           // 构建时首先通过 BitMap1 判断节点是否存在并将 BitMapl 的数据复制到 BitMap2,
                                              最后将节点插入 HashMap 中
end if
edges ← graph.get_edges()
for node in get_dst(edges) do                 // 当节点已存在于 BitMap2 中时,直接从 HashMap 获取节点对应的值;当节点不存在
                                              时,先计算节点的映射值,再同时更新 BitMap2 和 HashMap
    new_dst.append(get_value(map, node))
end for
for node in get_src(edges) do                 // 如果节点不存在,则更新 BitMap,中对应位置的 bit 值,然后对 HashMap 执行更新操作
    new_src.append(map.Update(node))
end for
s_graph← create_graph(new_src, new_dst)
for node in map.value() do
    output_node.append(node)
end for
```

3.1.3 SampleNeighbors 算子与 toBlockFast 算子融合

在图神经网络的采样训练过程中,SampleNeighbors 算子负责邻居采样,而 toBlockFast 算子用于构建子图 block。首先,在数据从 SampleNeighbors 算子到 toBlockFast 算子的转换过程中,大量冗余的条件判断占据了 80%的耗时;其次,采样过程生成的稀疏数据的存储格式为 COO 格式,而非子图构建所需格式,频繁的格式转换不仅增加了计算复杂度,也增加了内存占用;最后,算子之间 Python 和 C++多次调用导致额外的序列化开销,进一步降低采样训练效率。

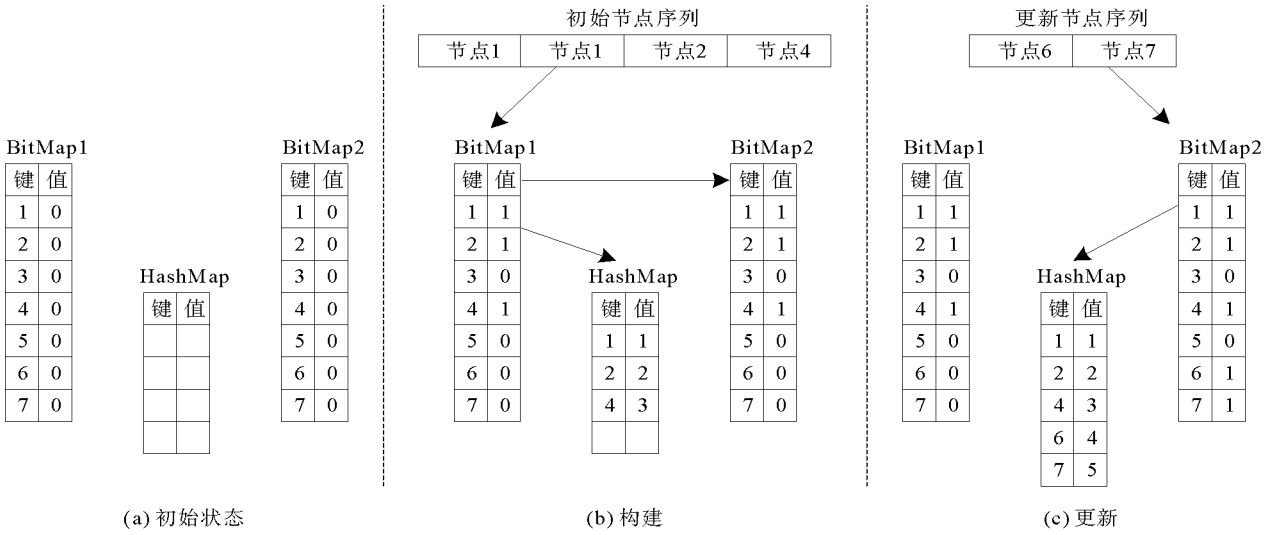


图 3 使用双位图构建和更新 HashMap

Fig. 3 Building and updating HashMap with double bitmaps

为了解决上述问题,将两个算子融合为一个算子,使采样操作直接生成符合 toBlockFast 算子要求的子图数据格式,避免冗余的条件判断和数据格式转换;在 C++层完成所有逻辑,减少 Python 和 C++之间的调用开销。两个算子的融合不仅能降低 SampleNeighbors 算子的写回开销,还能减少 toBlock 算子调用开销及访存开销,提高采样训练效率。

3.2 利用锁页内存提高特征传输效率

在使用非锁页内存的 CPU-GPU 数据传输中,CUDA(compute unified device architecture)驱动程序需要首先在主存中创建临时的锁页内存,然后将数据复制到这些页面中,最后通过直接内存访问(direct memory access,DMA)将数据传输到 GPU 显存。在图神经网络的采样训练过程中,每个批次的数据都需要 CPU 进行采样并发送相应的节点特征到 GPU 显存。频繁的数据传输导致在主存中需要不断创建临时锁页内存,影响整体训练效率。为此,本研究提出一种使用固定锁定内存页面进行数据传输的方法,具体是:在训练开始前,在内存中预留一块固定大小的锁页内存区域,每次特征提取时,先将节点特征和标签复制到该锁页内存中,然后通过 DMA 将数据从锁页内存传输到 GPU 显存,减少频繁的内存分配和释放操作,从而提高数据传输效率。

使用固定锁定内存页面进行数据传输需解决两个问题:

1) 确定内存大小。根据训练超参数和数据集特征(如节点特征维度和节点平均度数)来预估训练过程中所需的锁页内存大小。具体的估算公式为

$$M_S = B_S \times F_S \times \prod_{i=1}^{n} \min(f_i, D_{ED}) \text{。} \tag{2}$$

式中:M_S 为锁页内存大小,B_S 是每个迭代的小批量数据大小,F_S 是数据集节点特征的维度,f_i 是扇出系数的第 $i(i=1,2,\cdots,n)$ 项,D_{ED} 是数据集的平均节点度数。

2) 异常情况处理。尽管在计算锁页内存大小时已考虑了采样超参数 B_S 的影响,但在某些情况下,特征数据的大小仍可能超出分配的锁页内存。当特征数据量超出锁页内存大小时,可将数据分割成多个部分,每个部分的数据量等于锁页内存大小,通过异步传输方式发送至 GPU 内存,确保训练流程不受影响,减少训练等待时间。

3.3 基于多进程和多流数据的并行传输方法

由于 Python 的全局解释器锁(global interpreter lock,GIL)限制,无法将特征提取阶段与其他计算阶段并行。针对此问题,提出一种基于多进程和多流数据方法:为充分利用多核处理器资源,为每个进程创建独立的 GIL,并使用多流技术实现 GPU 训练和数据传输操作的并行。

多进程和多流数据传输方法采用生产者-消费者模型,具体如图 4 所示。在训练过程中,任务被分配给生产者进程和消费者进程。生产者进程负责处理输入的图数据,执行图采样,并将采样得到的特征数据发送到进程间共享内存区,同时将训练所需的信息传递给消费者进程。消费者进程负责接收来自生产者进程的信息,并从共享内存区提取训练数据,随后进行模型的计算处理。

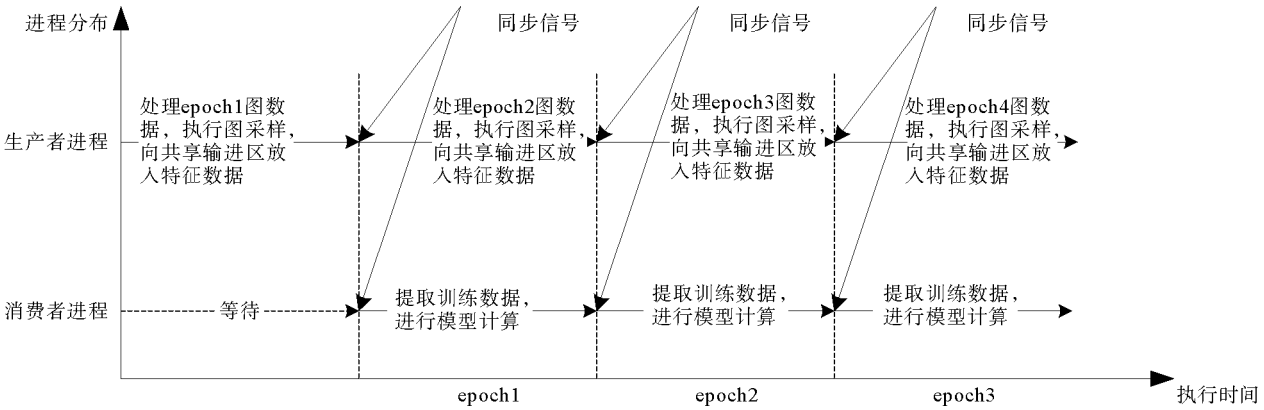


图 4 生产者-消费者模型

Fig. 4 Producer-consumer model

本方法的关键在于进程间数据传输和同步机制。进程间传输的数据包括节点特征及其标签和采样子图拓扑数据。生产进程和训练进程使用不同的 CUDA 流,对共享数据区的读写是并发进行的,再加上 GPU 显存资源有限,需要对共享数据区进行多次读写操作,这可能导致训练进程尚未完成对旧数据的训练时,生产进程的新数据就覆盖了旧数据。为保障训练流程正常进行,必须对共享数据区的读写操作进行同步控制。数据读写在 GPU 端进行,而数据生成在 CPU 端,因此进程间的同步信号需涵盖 CPU 和 GPU。CPU 端同步通过管道传输实现,GPU 端的同步利用 CUDA 事件来实现。

进程间共享内存区数据、节点特征和标签的数据格式均为张量格式,可以直接存储在进程间共享内存区中;而采样子图使用的是 DGL 自定义的数据格式,无法直接存放于共享内存区,需要通过队列实现进程间传输。进程间数据传输的实现流程如算法 2 所示。为了便于训练进程访问节点特征和标签,还需通过队列传输节点特征和标签在共享内存区中的位置索引。位置索引的数据量较小,进程间传输的主要开销来自于采样子图的传输过程。

进程间同步是确保训练正常进行的关键。CPU 端通过管道传输启动同步信号,生产进程通过阻塞式接收管道中的数据来实现同步。相对于主机,GPU 端内核的启动是异步的,强制 CPU 和 GPU 同步的操作会降低 GPU 的计算效率,因此采用 CUDA 事件来实现多流间的高效同步。进程间的同步如算法 3 所示。

综上,经过上述两方面优化后,可以实现特征提取和模型计算的并行处理,优化前后时空图如图 5 所示。假设图采样与特征提取的时间是 ∇t,模型计算的时间是 $2\nabla t$,批次数量为 n。优化前的总流水线耗时为 $(3n+1)\nabla t$,而优化后的总流水线耗时为 $(2n+4)\nabla t$。理论上,通过优化可以将整体训练时间缩短至原

来的 2/3。但由于各阶段的实际开销存在差异,并且加了进程间通信的额外开销,只有当 n 足够大时,优化效果才能接近理论值。

算法 2　进程间数据传输

```
输入:节点特征 features,节点标签 labels,队列 queue,采样子图 blocks,CUDAEvent_t event
输出:无
1) send_data ← [ ]
2) for i ← 0 to labels.shape[0] do
3)     event.wait()
4)    shape_1 ← features[i].shape[0]          //将 features[i]放入进程间的共享数据区
5)    shape_2 ← labels[i].shape[0]            //将 labels[i]放入进程间的共享数据区
6)    send_data.append([shape_1,shape_2,blocks])
7)    if 共享数据区已存满 do
8)        queue.put(send_data)
9)        send_data ← []
10)    end if
11) end for
```

算法 3　进程间的同步

```
输入:CUDA 事件 event,数据队列 queue
输出:无
1) model ← get_model()
2) train_data ← []
3) while 训练没有完成 do
4)     train_data ← queue.get()
5)    for i ← 0 to train_data.shape[0] do
6)        shape_1 ← train_data[i][0]          //根据 shape_1 从进程间的共享数据区获取 features
7)        shape_2 ← train_data[i][1]          //根据 shape_2 从进程间的共享数据区获取 labels
8)        blocks ← train_data[i][2]
9)        进行模型计算
10)       event.record()
11)    end for
12) end while
```

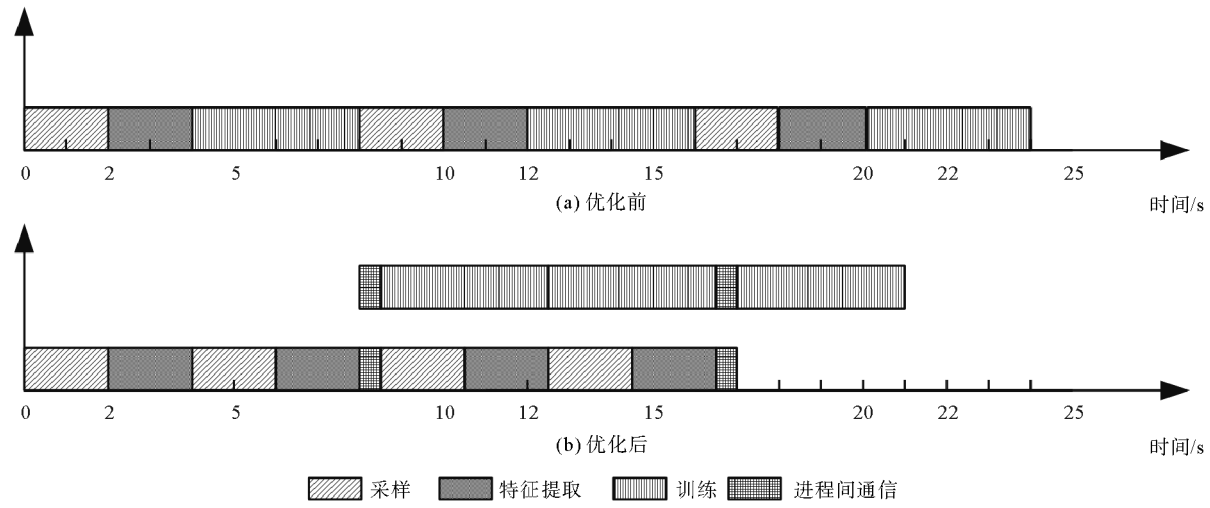


图 5　特征提取和训练阶段优化前后时空图

Fig. 5　Spatiotemporal diagram of feature extraction and training stage before and after optimization

4 实验评估与分析

实验设置与 2.1 节一致。

4.1 toBlock 算子优化实验

图 6 中展示了使用 toBlock 算子、toBlockFast 算子和融合算子在采样阶段的时间消耗对比。相较于 toBlock 算子，toBlockFast 算子提升了采样阶段效率，在四个数据集上的加速比分别达 1.59、1.75、2.00 和 1.31。融合算子在 toBlockFast 算子基础上进一步提升了采样阶段效率，在四个数据集上的加速比分别达 1.60、1.98、2.35 和 1.52。

4.2 特征传输优化实验

数据传输优化实验在采样阶段优化的基础上进行测试，图 7 展示了使用特征优化传输方案后的测试结果。优化数据传输后，对节点特征维度较大的数据集，如 Reddit 数据集和 Yelp 数据集，加速比分别最高可达 1.10 和 1.06；节点特征维度较小的数据集，如 Products 数据集和 Paper100M 数据集，加速比分别最高可达 1.79 和 1.58。

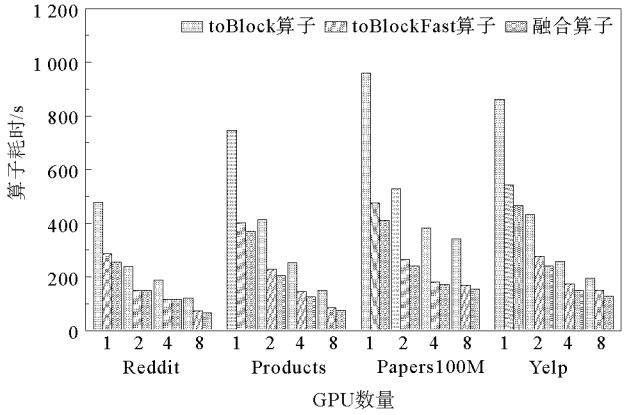


图 6 采样阶段的耗时对比

Fig. 6 Time consumption comparison in the sampling phase

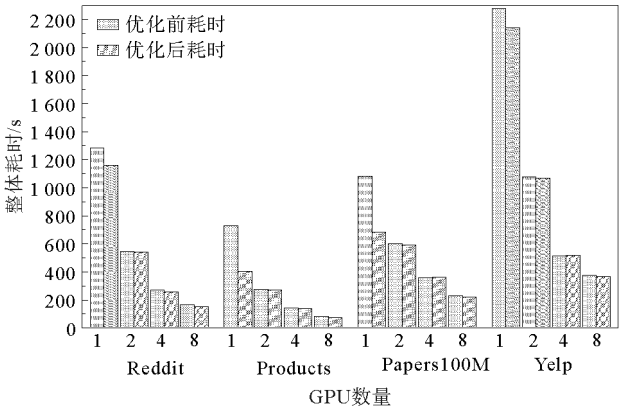


图 7 数据加载优化后加速比

Fig. 7 Acceleration ratio after data loading optimization

4.3 整体性能与精度测试

为了更全面地验证本研究方法的优势，将其与 DGL[12]、PyG[19] 和 Graph-Learn[20] 等主流 GNN 库进行实验对比。PyG 基于 PyTorch 后端，擅长动态计算和小批量处理；Graph-Learn 专为大规模图设计，优化了分布式训练和图分区。图采样优化方案与二者的采样训练加速比如表 2 所示。结果表明，本研究的图采样阶段优化方案在不同数据集和GPU配置下大多优于PyG和Graph-Learn，尤其是多GPU配置

表 2 图优化方案与 DGL、PyG、Graph-Learn 框架采样训练加速比对比

Table 2 Comparative acceleration ratios of the proposed graph optimization scheme against DGL、PyG and Graph-Learn

数据集	GPU 数量	DGL	PyG	Graph-Learn	数据集	GPU 数量	DGL	PyG	Graph-Learn
Reddit	1	1.11	1.08	0.96	Papers100M	1	1.58	1.67	1.27
	2	1.03	1.05	0.96		2	1.05	1.21	0.95
	4	1.06	1.24	0.95		4	1.16	1.54	1.12
	8	1.12	1.39	1.17		8	1.12	1.78	1.18
Products	1	1.80	1.44	1.36	Yelp	1	1.06	0.97	1.01
	2	1.10	1.00	0.91		2	1.08	1.03	1.05
	4	1.12	1.18	1.07		4	1.05	1.09	1.03
	8	1.10	1.25	1.09		8	1.04	1.16	1.09

下。对比 PyG 与采样阶段优化方案,本研究的图采样优化方案加速效果更明显。因 PyG 消息传递机制产生额外开销,底层数据结构及图切分后处理方式不如 DGL 优化方案高效。与 Graph-Learn 相比,采样阶段优化方案在多 GPU 设置下加速优势显著。Graph-Learn 虽在单卡环境下有一定加速,但多 GPU 时因通信延迟和资源竞争问题加速效果降低。而本研究方案在不同 GPU 数量下均取得较好加速效果。

Reddit 数据集下不同 batchsize 时,DGL、PyG 和图采样优化方案耗时对比如表 3 所示。在不同批采样大小下,图采样优化方案相比 DGL 和 PyG 均展现出更优的性能。例如,当 batchsize 为 1 000 时,图采样优化方案耗时仅 4.1 s,而 DGL 和 PyG 分别耗时 29.35 和 22.1 s。随着 batchsize 增大至 4 000 和 8 000,优化方案依旧保持性能领先。由此可见,所提出的优化方案在不同 batchsize 设置下均能有效提升模型性能。

表 3 Reddit 数据集下不同 batchsize 时 DGL、PyG 和图采样优化方案耗时对比

Table 3 Comparative time of DGL, PyG, and optimization scheme for Reddit dataset across batch sizes s

batchsize	DGL	PyG	图采样优化方案
1 000	29.35	22.14	4.10
4 000	13.79	11.51	1.98
8 000	9.39	7.44	1.29

最后评估优化方案对采样训练精度的影响。GraphSAGE 模型在经过 100 轮训练后,在不同数据集上的训练精度如表 4 所示。由测试结果可知,模型训练精度波动非常小,因此图采样加速方案并未改变采样结果。

表 4 采样优化前后训练精度对比

Table 4 Comparison of training accuracy before and after sampling training optimization

数据集	GPU 数量	优化前	优化后	数据集	GPU 数量	优化前	优化后
Reddit	1	0.972 3	0.977 5	Papers100M	1	0.495 3	0.481 9
	2	0.985 5	0.968 1		2	0.510 5	0.498 6
	4	0.986 5	0.969 4		4	0.509 6	0.499 3
	8	0.987 0	0.971 4		8	0.510 3	0.497 3
Products	1	0.757 4	0.759 1	Yelp	1	0.499 4	0.488 2
	2	0.775 7	0.762 1		2	0.358 9	0.336 2
	4	0.774 1	0.759 4		4	0.337 2	0.336 6
	8	0.778 1	0.754 4		8	0.359 9	0.342 8

5 结论

本研究针对图神经网络采样训练的主要性能瓶颈——采样阶段和特征提取阶段进行优化。对采样阶段的 toBlock 算子进行改进并与 SampleNeighbors 算子融合;在特征提取阶段,通过固定锁页内存提高数据传输效率,设计基于多进程和多流的数据传输方案,实现特征提取与模型计算的高效并行。实验验证了所提方法在多个数据集上展现出显著的性能提升,可有效缓解大规模 GNN 训练中的计算与存储瓶颈,为提升 GNN 在大规模图数据上的训练效率提供了可行的解决方案。

参考文献:

[1] CAO H,LI M,NIE L,et al. Vertex-based graph neural network classification model considering structural topological features for structural optimization[J/OL]. Computers and Structures,2024,305. DOI:10.1016/j.compsture.2024.107542.

[2] GENG H Y,CHEN C,HE Y X,et al. Pyramid graph neural network:A graph sampling and filtering approach for multi-scale disentangled representations[C]//Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. New York:Association for Computing Machinery,2023:518-530.

[3] LI W Z,WANG C D,XIONG H,et al. GraphSHA:Synthesizing harder samples for class-imbalanced node classification [C]//Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. New York:Association for Computing Machinery,2023:1328-1340.

[4] 张丽英,孙海航,孙玉发. 基于图卷积神经网络的节点分类方法研究综述[J]. 计算机科学,2024,51(4):95-105.
ZHANG Liying,SUN Haihang,SUN Yufa. Review of node classification methods based on graph convolutional neural networks[J]. Computer Science,2024,51(4):95-105.

[5] ZHANG M H,CHEN Y X. Link prediction based on graph neural networks[C]//Proceedings of the 32nd International Conference on Neural Information Processing Systems. Red Hook:Curran Associates Inc. ,2018:5171-5181.

[6] TAN T,YANG W. Research on knowledge graph entity recognition and relation extraction algorithm based on deep learning[C]//2024 IEEE 2nd International Conference on Electrical,Automation and Computer Engineering. Changchun:IEEE,2024:1521-1525.

[7] MA L,SHENG Z A,LI X K,et al. Acceleration algorithms in GNNs:A survey[J]. IEEE Transactions on Knowledge and Data Engineering,2025,37(6):3173-3192.

[8] SHEN Y Y,CHEN L,FANG J Z,et al. Efficient training of graph neural networks on large graphs[J]. Proceedings of the VLDB Endowment,2024,17(12):4237-4240.

[9] KHEMANI B,PATIL S,KOTECHA K,et al. A review of graph neural networks:Concepts,architectures,techniques,challenges,datasets,applications,and future directions[J]. Journal of Big Data,2024,11(1):18-61.

[10] HAMILTON W L,YING R,LESKOVEC J. Inductive representation learning on large graphs[C]//Proceedings of the 31st International Conference on Neural Information Processing Systems. Red Hook:Curran Associates Inc. ,2017:1025-1035.

[11] YE R,LI X,FANG Y J,et al. A vectorized relational graph convolutional network for multi-relational network alignment[C]//28th International Joint Conference on Artificial Intelligence. Macau:IJCAI 2019,2019:4052-4058.

[12] WANG M Y,ZHENG D,YE Z,et al. Deep graph library: Towards efficient and scalable deep learning on graphs[PP/OL]. arXiv (2019-09-03)[2025-04-20]. https://arxiv.org/pdf/1412.6980.

[13] LIN Z Q,LI C,MIAO Y S,et al. PaGraph:Scaling GNN training on large graphs via computation-aware caching[C]//Proceedings of the 11th ACM Symposium on Cloud Computing. New York:Association for Computing Machinery,2020:401-415.

[14] ZHENG D,MA C,WANG M J,et al. DistDGL:Distributed graph neural network training for billion-scale graphs[C]//2020 IEEE/ACM 10th Workshop on Irregular Applications:Architectures and Algorithms (IA3). Piscataway:IEEE,2020:36-44.

[15] YANG J B,TANG D H,SONG X Y,et al. GNNLab:A factored system for sample-based GNN training over GPUs[C]//Proceedings of the 17th European Conference on Computer Systems. New York:Association for Computing Machinery,2022:417-434.

[16] YIN Q J,LIU Q,FU Z R,et al. scGraph:A graph neural network-based approach to automatically identify cell types [J]. Bioinformatics,2022,38(11):2996-3003.

[17] SONG S,JIANG P. Rethinking graph data placement for graph neural network training on multiple GPUs[C]//Proceedings of the 36th ACM International Conference on Supercomputing. New York:ACM,2022:1-10.

[18] KINGMA D P,BA J. Adam:A method for stochastic optimization[PP/OL]. arXiv(2017-01-30)[2025-04-20]. https://arxiv.org/pdf/1412.6980.

[19] FEY M,LENSSEN J E. Fast graph representation learning with PyTorch geometric[PP/OL]. arXiv(2019-04-25)[2025-04-20]. https://arxiv.org/pdf/1903.02428.

[20] ZHU R,ZHAO K,YANG H X,et al. AliGraph: A comprehensive graph neural network platform[J]. Proceedings of the VLDB Endowment,2019,12(12):2094-2105.

(责任编辑:傅 游)