软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(9):4407−4420 [doi: 10.13328/j.cnki.jos.006647]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向多 GPU 的图神经网络训练加速*

苗旭鹏[1], 王驭捷[1], 沈 佳[1], 邵蓥侠[2], 崔 斌[1]

[1](北京大学 计算机学院, 北京 100871)

[2](北京邮电大学 计算机学院, 北京 100876)

通信作者: 崔斌, E-mail: bin.cui@pku.edu.cn

摘 要: 图神经网络由于其强大的表示能力和灵活性最近取得了广泛的关注. 随着图数据规模的增长和显存容量的限制, 基于传统的通用深度学习系统进行图神经网络训练已经难以满足要求, 无法充分发挥 GPU 设备的性能. 如何高效利用 GPU 硬件进行图神经网络的训练已经成为该领域重要的研究问题之一. 传统做法是基于稀疏矩阵乘法, 完成图神经网络中的计算过程, 当面对 GPU 显存容量限制时, 通过分布式矩阵乘法, 把计算任务分发到每个设备上, 这类方法的主要不足有: (1) 稀疏矩阵乘法忽视了图数据本身的稀疏分布特性, 计算效率不高; (2) 忽视了 GPU 本身的计算和访存特性, 无法充分利用 GPU 硬件. 为了提高训练效率, 现有一些研究通过图采样方法, 减少每轮迭代的计算带价和存储需求, 同时也可以支持灵活的分布式拓展, 但是由于采样随机性和方差, 它们往往会影响训练的模型精度. 为此, 提出了一套面向多 GPU 的高性能图神经网络训练框架, 为了保证模型精度, 基于全量图进行训练, 探索了不同的多 GPU 图神经网络切分方案, 研究了 GPU 上不同的图数据排布对图神经网络计算过程中 GPU 性能的影响, 并提出了稀疏块感知的 GPU 访存优化技术. 基于 C++和 CuDNN 实现了该原型系统, 在 4 个不同的大规模 GNN 数据集上的实验表明: (1) 通过图重排优化, 提高了 GPU 约 40% 的缓存命中率, 计算加速比可达 2 倍; (2) 相比于现有系统 DGL, 取得了 5.8 倍的整体加速比.

关键词: 图神经网络; 分布式计算; 内存优化; GPU 加速

中图法分类号: TP303

中文引用格式: 苗旭鹏, 王驭捷, 沈佳, 邵蓥侠, 崔斌. 面向多GPU的图神经网络训练加速. 软件学报, 2023, 34(9): 4407–4420. http://www.jos.org.cn/1000-9825/6647.htm

英文引用格式: Miao XP, Wang YJ, Shen J, Shao YX, Cui B. Graph Neural Network Training Acceleration for Multi-GPUs. Ruan Jian Xue Bao/Journal of Software, 2023, 34(9): 4407–4420 (in Chinese). http://www.jos.org.cn/1000-9825/6647.htm

Graph Neural Network Training Acceleration for Multi-GPUs

MIAO Xu-Peng[1], WANG Yu-Jie[1], SHEN Jia[1], SHAO Ying-Xia[2], CUI Bin[1]

[1](School of Computer Science, Peking University, Beijing 100871, China)

[2](School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: In recent years, graph neural networks (GNNs) have attracted wide attention due to their powerful and flexible representation ability. Considering the increasing scale of graph data and the limitation of the video memory capacity, it becomes more challenging to train GNNs with traditional general deep learning systems, and such training cannot give full play to the performance of GPU devices. To achieve efficient use of GPU hardware for GNN training is one of the important research issues in this field. Traditional approaches employ sparse matrix multiplication for the calculation process of GNNs. When the video memory capacity of GPU devices is limited, the computation tasks are distributed to each device by distributed matrix multiplication. Their shortcomings are mainly as follows: (1) Sparse matrix multiplication ignores the sparse distribution of the graph data, which results in low computation efficiency. (2) These methods

* 基金项目: 国家重点研发计划 (2018YFB1004403); 国家自然科学基金 (61832001, U1936104); 北京大学-腾讯协同创新实验室项目; CCF-百度松果基金
收稿时间: 2021-08-02; 修改时间: 2021-09-26; 采用时间: 2022-01-23; jos 在线出版时间: 2023-01-04
CNKI 网络首发时间: 2023-01-05

ignore the computation and memory access characteristics of GPU and fail to utilize the hardware resources. To improve the training efficiency, some studies propose to reduce the costs of each iteration and storage requirements through graph sampling techniques, which also support flexible distributed scaling. Due to the stochastics and variance, however, these methods often affect the model accuracy. Therefore, this study proposes a high-performance GNN training framework for multi-GPUs. Different GNN partition strategies for multi-GPUs are explored, and the influence of different graph ordering patterns on the GPU performance during the calculation process of GNNs is investigated to ensure the accuracy of the model. Moreover, block-sparsity-aware optimization methods are put forward for GPU memory access. The prototype system is achieved using C++ and CuDNN. The experiments on four large-scale GNN datasets demonstrate that (1) the graph re-ordering method improves the cache hit rate of GPU by around 40% and doubles the computation speedup; (2) compared to the existing system DGL, the proposed system achieves a total speedup of 5.8x.

Key words: graph neural network (GNN); distributed computation; memory optimization; GPU acceleration

图是一种具有强大表示能力的数据结构, 能够表达许多真实世界中具有复杂关系结构的数据, 例如分子结构、社交网络、知识图谱、文献引用网络等. 随着机器学习技术的成熟, 为了处理复杂图数据, 研究者们将图嵌入算法与卷积神经网络结合, 提出了图神经网络 (graph neural network, GNN) [1–4]. GNN 使得机器学习算法从欧式空间数据的特征提取拓展到非欧式空间, 其已在多种与图相关的任务中取得了目前最佳的结果, 例如节点分类[1]、图分类[5]、链接预测[6]等.

GNN 训练算法可大致分为两种: (1) 基于全量图的训练算法, 称为 Full-batch 算法; (2) 基于采样子图的训练算法, 称为 Mini-batch 算法. Full-batch 算法中, 图中所有节点都参与运算和梯度更新, 其优点是没有精度损失, 缺点是训练内存开销大. 典型工作有 GCN[1], SGC[7], N-GCN[8], LightGCN[9]等. 基于采样的 Mini-batch 算法中, 每次只选取一批节点构成子图参与 GNN 的运算和梯度更新. 这类训练算法的优点是训练内存开销小, 且易于并行拓展; 缺点是由于采样算法存在随机性, 子图具有方差, 难以完全反映完整的数据分布, 从而使得训练的模型精度有所下降. 这类算法典型工作有 GraphSAGE[5], FastGCN[10], GraphSAINT[11]等.

为了利用 GPU 实现高效训练, GNN 通常使用传统的深度学习系统如 TensorFlow, PyTorch 等完成计算. 在这类通用系统中, GNN 的主要计算过程通过预置算子库如 CuSparse 中的稀疏矩阵乘法 (SpMM) 来实现. 由于现实世界中的图数据往往非常庞大复杂, 这类系统逐渐难以满足 GNN 对计算性能的需求. 为此, 有些工作在深度学习系统的基础上, 提供了统一的 GNN 模型加速库, 并进行了底层算子优化, 代表性系统包括 DGL[12], PyG[13]等, 但是这些系统只支持单卡训练. 随着图规模的日益增大, 数据难以存储到单一设备当中, 分布式图表示学习系统开始出现. 早期的分布式 GNN 系统如 AliGraph[14]采用 METIS 算法把图数据的切分到不同设备上, 每个设备基于当前子图进行采样和训练. CAGNET[15]是首个采用全量图进行 Full Batch 的 GNN 训练的分布式系统, 它将传统的分布式矩阵乘法技术应用到 GNN 训练当中, 提出了一系列 GNN 分布式训练的切分方案, 并分析了不同切分方案的计算负载和通信代价. 相比于基于采样的方法, 基于全量图的方法虽然有精度保证, 但是现有解决方案存在着以下挑战.

(1) 数据访问量、计算量更大, 面临着严重的硬件执行效率问题.

(2) 缺乏对图数据内容的感知, 没有充分利用图数据本身的稀疏分布性质.

针对以上问题, 本文提出了一套面向全量图的多 GPU 图神经网络训练框架. 具体来说, 首先针对多 GPU 实现了包括 1D、1.5D 在内的不同 GNN 分布式切分算法; 在此基础上, 研究了图数据排布对 GPU 训练图神经网络的影响, 发现通过图重排技术可以聚集图数据访问, 从而显著提高 GPU 的执行速度; 进一步, 作者把图重排方案拓展到多 GPU 情况, 保证不同 GPU 间的负载均衡; 最后, 在图重排基础上进行 2D 切分, 提出了稀疏块感知的 GPU 访存优化方法, 对于图中不同密度的块自适应地采用不同的计算方式, 大大提高了 GPU 访存性能. 实验表明: (1) 图重排技术提高了 GPU 约 40% 的缓存命中率, 从而加速 GNN 训练时的显存访问; (2) 在多 GPU 场景下, 整体取得了至少 2 倍的计算加速比; (3) 相比于现有系统 DGL, 取得了 5.8 倍的整体加速比.

本文的主要贡献可概括如下.

(1) 提出了一套面向多 GPU 的高效分布式 GNN 训练算法, 支持分布式 GNN 的多种切分方式, 包括 1D 和 1.5D 算法等.

(2) 提出了通过图重排技术提高 GPU 上 GNN 执行效率的优化方法, 探究了不同的图重排算法的优化效果, 并拓展到多 GPU 场景.

(3) 提出了一套稀疏块感知的 GPU 访存优化方法, 引入 GPU 共享内存, 进一步提高 GNN 计算时的访存性能.

(4) 实现了一套基于 C++和 CuDNN 的多 GPU 的 GNN 训练框架, 在 4 个大规模图数据集上进行了实验, 结果充分证明了以上优化算法的有效性.

1 相关工作

本节简要介绍常见的图神经网络算法和现有的图神经网络训练系统.

1.1 图神经网络算法

图神经网络把卷积神经网络中的卷积操作拓展到图数据, 每个节点通过聚合自身和邻居节点的信息来学习节点的表示, 通用公式可用公式 (1) 和公式 (2) 表示:

$$z_v^l = AGGREGATE\left(\left\{h_u^{l-1} | u \in N(v)\right\}\right) \tag{1}$$

$$h_v^l = UPDATE\left(z_v^l, h_v^{l-1}\right) \tag{2}$$

其中, h_v^l 是图节点 v 在第 l 层的特征 (feature), h_v^0 是输入特征, z_v^l 代表聚合后的邻居信息, $N(v)$ 是节点 v 的邻居点集. $AGGREGATE$ 通过求和等方式聚合邻居信息, $UPDATE$ 用邻居和本节点特征更新下一层节点表示. 典型的基于全量图的 Full-batch 算法的 GNN 模型: GCN 首次使用图卷积算子聚合邻居信息; SGC[7]通过反复消除 GCN 层间的非线性变换来降低 GCN 复杂度; N-GCN[8]将多跳邻接矩阵作为 GCN 的聚合方式, 把不同阶邻居的信息结合在一起; LightGCN[9]对 GCN 进行了简化, 只保留其中邻居聚合这一基本结构, 使其更适用于推荐系统.

1.2 图神经网络训练系统

现有的 GNN 训练系统底层主要基于主流的深度学习系统搭建, 例如 PyTorch[16]等, 此类系统主要针对单卡场景. 当面对大规模的图数据需要拓展到多卡分布式环境时, 他们主要通过类似 AliGraph 的基于采样的 GNN 算法进行支持. ROC[17]是一个针对大图在 GPU 上进行全量图 GNN 训练的系统, 通过将 CPU 内存作为 GPU 显存的拓展, 可以帮助解决大图训练内存开销过大的问题, 但是由于 PCIe 带宽的限制, 显存和内存的数据交换会带来严重的性能损失. 由此可见, 如何在全量图上进行 GNN 训练仍然具有重要研究意义, 因此本工作主要着眼于 Full-batch 的分布式 GNN 算法的优化加速.

2 面向多 GPU 的分布式 Full-batch 图神经网络训练算法

2.1 算法描述

单个 GPU 显存有限, 难以支撑大图的 Full-batch 的 GNN 训练, 可以用多 GPU 并行计算来支持和加速训练. Full-batch GNN 的计算过程本质上是稀疏矩阵乘法 (sparse matrix multiplication, SpMM), 本文使用 GCN 作为基本模型进行优化, 其他 Full-batch GNN 模型如 SGC、N-GCN、LightGCN 等由于具有相似的计算特性, 也依然适用. GCN 的正向和反向核心运算如公式 (3)–公式 (6) 所示:

正向传播:

$$Z^l = AH^{l-1}W^l \tag{3}$$

$$H^l = \sigma\left(Z^l\right) \tag{4}$$

反向传播:

$$G^{l-1} = A^{\mathrm{T}}G^l\left(W^l\right)^{\mathrm{T}} \odot \sigma'\left(Z^{l-1}\right) \tag{5}$$

$$Y^{l-1} = \left(H^{l-1}\right)^{\mathrm{T}}A^{\mathrm{T}}G^l \tag{6}$$

其中, AH^{l-1}, $A^{\mathrm{T}}G^l$ 均为稀疏矩阵乘法. 作为示例, 图 1 展示了公式 (3) 的计算过程, 图数据以邻接矩阵的形式组织, 每层 GCN 会将邻接矩阵 A 和特征矩阵 H^{l-1} 相乘, 从而实现公式 (1) 中的邻居信息聚合. 之后会将计算结果与本层参数矩阵 W^l 相乘进行非线性激活, 成为下一层的特征矩阵.

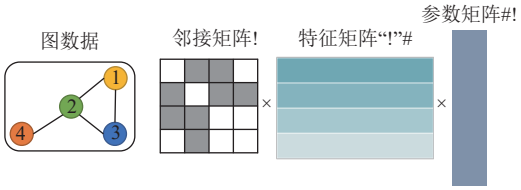


图 1 图卷积神经网络计算过程示例 (第 l 层)

算法执行前会把输入数据进行合理切分并分配给每个 GPU, 进行稀疏矩阵乘法的计算, 再借助通信把各个 GPU 上的计算结果相结合, 得到完整的矩阵乘积. 分布式矩阵乘法按照进程之间的网络拓扑可以划分为 1D 算法、2D 算法以及 3D 算法, 这些算法会按照相应的维度 (dimension) 对矩阵进行切分, 在此基础上, 通过引入进程组的概念, 又出现了 1.5D 算法以及 2.5D 算法. 在 GCN 训练中通常只对矩阵 A, H 进行切分, 矩阵 W 完整存储在每一个进程中. 接下来以 $T=AH$ 为例, 分别介绍基于 1D 和 1.5D 切分的 GCN 训练. 表 1 展示了本文所用到的标记与符号.

表 1 标记与符号

符号	描述
n	图的节点数量
f^l	第 l 层节点特征长度
nnz	稀疏邻接矩阵的非零元素数
L	GNN层数
σ	激活函数
p	参与运算的进程总数
A	规范化后的稀疏邻接矩阵 ($n\times n$)
H^l	第 l 层的密集特征矩阵 ($n\times f^l$)
W^l	第 l 层的密集参数矩阵 ($f^{l-1}\times f^l$)
Z^l	第 l 层激活函数的输入矩阵 ($n\times f^l$)
G^l	$Loss$ 对 Z^l 的梯度矩阵 ($n\times f^l$)
Y^l	$Loss$ 对 W^l 的梯度矩阵 ($f^{l-1}\times f^l$)

2.1.1 基于 1D 切分的 GCN 训练

1D: 按行均匀切分 A 和 H, 每个进程负责存储 A 和 H 的一个行块, 计算部分 A 和整个 H 的乘积. 假定共有 p 个进程, 则在每个进程内, 将存储的矩阵 A 按列划分, 每次计算一个列块与矩阵 H 一个行块的乘法. 在计算前, 拥有该矩阵 H 块的进程需要把该块广播给所有进程. 图 2 展示了一个 1D 切分 (p=2) GCN 中 SpMM 计算的例子, 两个进程 ($P0$ 和 $P1$) 的数据块分别用不同深浅度的颜色表示.

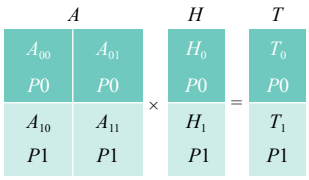


图 2 矩阵 1D 切分示例 (p=2)

2.1.2 基于 1.5D 切分的 GCN 训练

1.5D: 在如 1D 按行切分 A 和 H 的基础上, 加入重复度 r. 假定进程组数为 p/r, 每个进程组中含有 r 个进程. 则进程组 i 中的所有 r 个进程均拥有矩阵 A 的 $A_{i,0}-A_{i,p/r-1}$, 矩阵 H 的 H_i. 每个进程组合作计算一个行块, 即 $T_i=\sum_{j=0}^{j=p/r-1}A_{i,j}H_j$. 其核心思想是利用多倍的进程资源重复存储、合作计算来提高并行度. 在进程组内, 所有 GPU 并

行计算, 比 1D 算法的计算速度更快. 在进程组间, 计算对应列的进程需要进行矩阵 H 行块的广播, 同 1D 算法类似. 此外, 1.5D 算法的额外代价是进程组内需要通过 AllReduce 通信[18]来对求得的 H 进行加和. 图 3 中展示了 1.5D 算法 (p=4, r=2) 的一次迭代计算步骤.

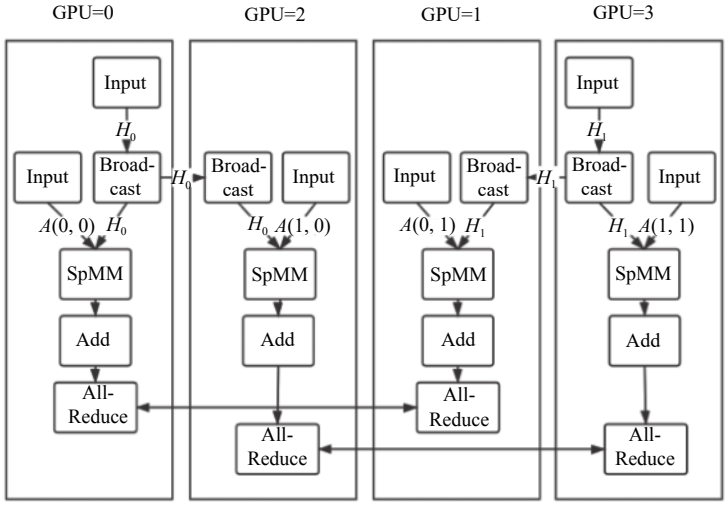


图 3 1.5D 切分下 GCN 执行数据流图 (p=4, r=2)

2.2 时间复杂度分析

计算的内容: 计算部分耗时最多的是矩阵乘法运算. 由于稀疏矩阵 A 的维度 n 远大于节点特征维度 f^l, 稀疏矩阵 A 参与的稀疏矩阵乘法开销远大于其他密集矩阵乘法, 故 SpMM 计算是本文关注的重点. 在以上分布式 GNN 基本方案中, SpMM 出现在正向传播中的 AH^{l-1} 以及反向传播中的 $A^{\mathrm{T}}G^l$, 根据分布式切分和通信方案的不同, 此处的 A 、 H 、 G 会被切分存放在多个 GPU 中, 由多个 GPU 共同并行完成分块计算.

SpMM 时间复杂度: 在 SpMM 计算中, 由稀疏矩阵 $S\,(d_1\times d_2)$ 与密集矩阵 $D\,(d_2\times d_3)$ 相乘得到密集矩阵 $M\,(d_1\times d_3)$. 本文中的稀疏矩阵均以标准的 CSR (compressed sparse row) 格式存储, 包括: indptr, indices, data. 其中, data 表示稀疏矩阵中所有非零元素的值, indptr 表示每一行首元素在 data 数组中的索引值, indices 表示 data 数组中对应元素的列索引值. SpMM 计算流程如算法 1 所示.

算法 1. 顺序的 SpMM 计算流程.

输入: CSR 格式稀疏矩阵 $S\,(d_1\times d_2)$, 密集矩阵 $D\,(d_2\times d_3)$;
输出: 密集矩阵 $M\,(d_1\times d_3)$.

```
1 for i = 0→ d_1 − 1 do
2     for j = S.indptr[i] → S.indptr[i+1] − 1 do
3         for k = 0 → d_3 − 1 do
4             M[i][k] += S.data[j] * D[S.indices[j]][k]
```

循环 S 每一行, 将其中每一个非零元素 (设列号为 c) 与 D 每一列中对应位置的元素 (行号为 c) 相乘并累加, 时间复杂度是 $\mathrm{O}(nnz\times d_3)$, nnz 是 CSR 矩阵 S 中非零元素的数量, d_3 是密集矩阵 D 的宽度. 在分布式 GNN 方案中, AH^{l-1} 的复杂度为 $\mathrm{O}(nnz(A)\cdot f^{l-1})$, $A^{\mathrm{T}}G^l$ 的复杂度为 $\mathrm{O}\left(nnz(A)\times f^l\right)$. 对于切分的 GCN 训练, 每个进程在单层 GCN 的 SpMM 乘法时间复杂度为:

$$C_{\mathrm{SpMM}}=O\left(\frac{1}{k}\cdot nnz(A)\cdot(f^{l-1}+f^l)\right),$$

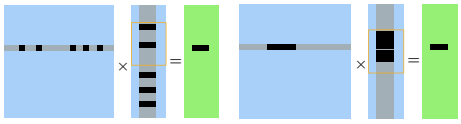
其中, 并行比例系数 k ($k>1$) 是一个常数, 由分布式切分方案确定. 注意到任何 GNN 算法本质上都可以化归到稀疏矩阵乘法 SpMM 上, 故以上复杂度分析对任何 GNN 算法均有效.

3 基于图重排列的分布式训练算法优化

3.1 GNN 计算内存访问分析

当前基本方案中的分布式 SpMM 算法并没有利用到矩阵 A 的稀疏特性, 其实质上也是适用于密集矩阵乘法的分布式方案. 图数据结构带来的矩阵稀疏性意味着数据访问空间局部性较差, Cache 命中率较低, 这为通过提高内存访问连续性来加速计算提供了空间.

在 GPU 中, CSR 稀疏矩阵的多线程 SpMM 机制如图 4(a) 所示: 每个线程计算一行乘一列, 根据矩阵 A 行中非零元素位置读取矩阵 H 列中对应位置乘子, 相乘并累加, 一个 warp 中的多线程并行计算连续的多个位置. GPU 的 Cache 中缓存着密集矩阵 H 的部分连续行 (图 4 黄色框表示 Cache 缓存的行), 矩阵 A 的稀疏性使得 A 行中的非零元素相隔较远, 对应需要读取的 H 列中的乘子相隔也较远, 由于容量有限 Cache 每次只能缓存连续的若干行, 命中率较低. 例如图 4(a) 中, 当计算第 1 个元素乘法时, Cache 只能缓存两行有效的 H 数据, 当计算第 3 个元素乘法时又需要重新访问主存读取数据.



(a) 图重排前 (b) 图重排后

图 4 多线程 SpMM 与缓存图示

以上现象是图数据访问的空间局部性较差导致的. SpMM 中的一行乘一列在图中含义为每个节点聚集邻居节点的信息 (如公式 (1), 公式 (2) 所示), 节点的特征向量 (H 矩阵的一行) 在主存中依照节点编号排布, 当这些邻居节点的编号较分散时, 对应的特征向量在主存中也会分散排布, 则聚集信息时内存访问局部性较差, Cache 命中率较低.

算法的优化目标是提高 Cache 命中率, 减少内存拷贝数据到 Cache 的次数, 或减少低级 Cache 拷贝数据到高级 Cache 的次数, 从而加速 SpMM 计算过程. 实现该目标的优化方法是图重排 (graph reordering), 通过对图节点重新编号使图中相邻节点具有邻近编号, 则相邻节点的特征在主存中连续存储, 从而提高 Cache 命中率. 相邻节点连续编号体现在邻接矩阵上的就是矩阵 A 非零元素连续排布.

3.2 图重排算法

图重排算法将图节点进行重新编号, 其输入为无向图的邻接矩阵, 将一个置换映射 ($f:\{1,\ldots,N\}\to\{1,\ldots,N\}$) 作用在图数据的邻接矩阵、特征、标签上即得到重排后的图数据.

已有一些工作利用图重排算法来优化图算法效率, 本文实现了 4 种算法, 并通过单机 GNN 算法上的实验结果挑选计算加速比最大的算法, 用于设计针对分布式 GNN 算法的优化方案. 4 种图重排算法分别如下: (1) DegSort: 根据节点的度数对节点进行降序排序作为重排映射; (2) SlashBurn[19]: 该图重排算法被用于图压缩, 能够加速稀疏矩阵计算; (3) RCM[20], 该算法是逆 Cuthill-McKee 算法, 用于减少图带宽, 是基于树的图重排算法, 通过图上的广度优先搜索获得重排编号; (4) METIS[21]重排, METIS 是经典的图切分算法, 将图节点切分为若干个簇, 最小化边割 (edge-cut), 即簇之间的边数, 使用 METIS 将图分为若干簇后, 将同一簇内的点连续编号即得重排映射. 其中, METIS 和 RCM 的效果均为使图中相邻的节点尽量获得连续编号, 故重排后的邻接矩阵会出现非零元素连续排布的情形.

3.3 分布式 GNN 的分块图重排优化算法

利用图重排算法可以通过增加内存访问的空间局部性来提高 Cache 命中率, 从而加速 SpMM 运算. 然而若直接将图重排应用到分布式 GNN 的方案中, 会破坏矩阵 A 各分块的稀疏性平衡, 使得不同进程之间负载失衡, 使加速效果大打折扣.

算法 2. 分布式 GNN 的分块图重排算法.

输入: A, n, $blocknum$, $Graph\_reorder\_alg$;

输出: 重排索引 $reorder\_index$ $(1\times n)$.

1 for $i=0\rightarrow blocknum-1$ do

2 $s,e=i*n/blocknum,\ (i+1)*n/blocknum$

3 $adj\_block=A[s:e,\ s:e]$

4 $reorder\_index[s:e]=Graph\_reorder\_alg(adj\_block)+s$

5 return $reorder\_index$

例如, 直接将 METIS 重排应用在 Reddit 全图上, 然后部署 1.5D (p=4, r=2) 分布式方案 (此时 4 个进程并行分别计算 4 个分块), SpMM 计算时间由最慢的分块决定. 重排前后每个分块的边数和 SpMM 计算时间如图 5(a) 和图 5(b) 所示. 可见, 在全图重排后, 对角块成为密集块, 非对角块成为稀疏块, 稀疏性失衡. 在对角块中, METIS 重排使得非零元素连续排布从而加速计算, 但由于计算密度过高, 加速幅度打折扣; 在非对角块中, 因其计算密度下降故由加速, 但并不出现非零元素连续排布, 无法利用内存访问连续性加速. 如此的结果便是各分块之间负载不均衡, 由于 1.5D 算法性质, 稀疏块需要等待密集块计算完成, 图 5(b) 中计算时间由最慢分块的 309.16 ms 决定, 加速幅度大打折扣.

(a) 原图 (b) 全图重排 (c) 分块图重排

图 5 METIS 重排前后每个分块的边数和对应的 SpMM 计算时间

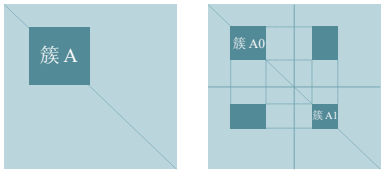
本文提出针对分布式 GNN 的分块图重排优化算法 (算法 2), 该算法的思想是根据不同的分布式 GNN 方案, 对原图进行对应的切块 (切块数如表 2 所示), 对每一个对角块 (可视为一个子图) 调用图重排算法, 并将所有重排索引拼接成完成重排映射.

该算法有以下两个性质: (1) 在原图稀疏性均衡的情况下, 分块图重排优化算法不改变各块稀疏性, 从而保证负载均衡; (2) 所有块均可通过内存访问连续性加速计算, 且加速程度相近.

具体原因如下: 性质 (1) 是由于对角块和非对角块的序号重排均只发生在块内, 因此不改变各块稀疏性. 性质 (2) 的原因如图 6 所示, 以 METIS 分块图重排 (blocknum=2) 为例, 在全图重排情形下, 设簇 A 为一个点簇, 其内部连通性强、连边密集, 该点簇的点连续编号故非零元素集中在对角线附近. 假设原图均匀分布, 则在分块重排中簇 A 被均匀分为簇 A0 和簇 A1 并分别位于对角块的对角线附近. 由于同属于连通性强的簇 A, 故簇 A0 和簇 A1 间具有密集连边, 对应的即非对角块中非零元素连续排布. 可见, 所有块中均出现非零元素连续排布情形, 且连续排布的密集程度相仿, 因此有效地提高了每个分块 SpMM 计算的内存访问的空间局部性, 各块加速效果接近.

表 2 不同分布式 GNN 算法下的分块数

分块数 (blocknum)	分布式GNN方案
2	1D (p=2) / 1.5D (p=4, r=2)
4	1D (p=4) / 1.5D (p=8, r=2)
8	1D (p=8)



(a) 全图重排 (b) 分块图重排

图 6 性质 (2) 原因解释

另外值得一提的是: (1) 分块图重排算法是针对计算的加速优化, 其不改变分布式 GNN 算法通信总量; (2) 分块图重排算法的时间开销与大规模图上分布式 GNN 训练的时间相比, 可以忽略不计.

4 稀疏块自适应的 GPU 计算优化

传统的 SpMM 并不关注邻接矩阵内数据分布, 整体均采用稀疏方式进行计算. 图重排虽然一定程度上提高了邻接矩阵内密集排列的元素的 GPU 的空间访问局部性, 但是对于无法紧密排列的部分, GPU 仍然无法充分发挥性能. 事实上, 现实生活中的图数据通常具有聚集性, 即在拓扑上呈现出聚簇效果, 聚簇子图内部的边比较密集, 聚簇子图之间的边比较稀疏. 因此, 如果对邻接矩阵进行图重排, 也会使得邻接矩阵的非零元素呈现出明显的聚集分布 (如图 6 所示). 考虑到这种性质, 本文提出了一种稀疏块感知的 GPU 计算优化方法 (block sparse), 利用重排后邻接矩阵不同块之间的稀疏性差异, 自适应的选择稀疏/稠密矩阵乘法进行计算[22], 充分发挥 GPU 的访存特性来提高数据访问局部性, 从而加速 GNN 计算.

4.1 稀疏块自适应的 GNN 计算

GPU 作为一种单指令多线程 (SIMT) 的处理器, 通常被用来对数据密集型任务进行计算加速. GPU 拥有百上千的线程, 在逻辑上会以线程束 (32 个线程) 的方式进行组织. 同一个线程束中的线程会同时执行相同的指令. 为了利用图数据的聚集性, 可以预先对图进行 2D 切分, 把图重排后的邻接矩阵切割成 $n/32 \times n/32$ 个大小相同的块 (为了对应 GPU 中的线程束, 取块大小为 32×32). 之后, 根据每个小块的非零元素的密集程度, 选择对应的 GNN 计算模式. 具体算法如算法 2.

算法 3. 稀疏块自适应的 GNN 计算流程.

输入: A, H, n, $density\_threshold$

1 for $i = 0 \to n/32 - 1$ do
2 for $j = 0 \to n/32 - 1$ do
3 $block = A[i*32:(i+1)*32; j*32:(j+1)*32]$
4 $nnz_{block} = count\_nonzero\_items(block)$
5 if $nnz_{block} > density\_threshold$
6 $DenseMM(block, H)$
7 else
8 $SpMM(block, H)$

通过统计每个小块非零元素个数, 对不同密集度的块适配不同的计算方法, 如果是稀疏块, 则采用 CSR 格式存储该块邻接矩阵, 用算法 1 中的标准 SpMM 方式进行计算, 如果是稠密块, 则采用传统的稠密矩阵乘法完成计算.

4.2 GPU 实现与优化

GPU 线程分配: GPU 作为一种特殊的计算设备, 通常被用来借助其并行计算特性, 加速神经网络训练中的矩阵运算. GPU 的最小逻辑单元为线程束, 可以在同一周期中执行相同的指令. 因此在我们的 GNN 计算中, 会把每个 32×32 大小的邻接矩阵分配给一个线程束进行处理, 在后续的计算过程中, 该线程束执行的具体函数由算法 3 中的密集程度判断条件决定.

GPU 共享内存优化: 除了 GPU 显存以外, GPU 还拥有共享内存. GPU 共享内存本质上是一种可编程的缓存, 由于物理硬件上更靠近流处理器, GPU 的共享内存具有比显存更低的延迟和更高的带宽 (几十倍). 因此, 可以利用共享内存来存储邻接矩阵, 从而提高 GPU 访存效率, 加速 GNN 的计算过程. 但是, 共享内存的容量往往只有几十 KB, 并且会被同一个 GPU 流处理器下的所有线程共享, 所以并不能无限制的使用. 为了解决这个问题, 需要把每个邻接矩阵块进一步分解成规模更小的切片, 逐片将邻接矩阵数据从显存拷贝到共享内存, 再进行后续的计算.

5 实验结果

5.1 数据集介绍

本文共选用 4 个图数据集, 具体信息如表 3 所示. Reddit 是在线讨论论坛数据集, Proteins (源自 OGB, Open Graph Benchmark) 和 Archaea[23]表示生物之间蛋白质的互相关联, Products (源自 OGB) 表示 Amazon 商品共同购买网络. 其中, Reddit 和 Proteins 属于 GNN 常用的较大规模的图, Archaea 和 Products 属于超大规模的图. 部分数据集节点特征缺失, 由固定维度单位阵补足.

表 3 数据集信息

数据集	节点数	边数	特征数	类别数
Reddit	232 965	114 848 857	602	41
Proteins	132 534	79 255 038	602	8
Archaea	1 644 228	206 436 882	602	10
Products	2 449 029	126 167 181	100	47

所有实验均在 NVIDIA TITAN RTX 八卡 GPU 集群上进行, 分布式实验中使用的 GPU 数量等于分布式方案进程数 p 的值. GNN 结构为两层 GCN, 隐层节点数默认为 128, 对于数据集 Products, 隐层节点数为 64.

5.2 收敛性能测试

本系统底层的训练框架是河图深度学习系统 (https://github.com/PKU-DAIR/Hetu)[24], 基于 C++和 CuDNN 等底层通用计算库进行开发, 为了验证基本性能和正确性, 本节选取 Reddit 数据集对本系统和 PyTorch 进行端到端的收敛性能测试, 结果如图 7 所示. 首先, 两者训练均是 Full-batch 算法, 并且是基于相同的超参数 (如学习率) 配置, 可以看出最终收敛的测试集表现 (test loss) 几乎一致, 验证了我们实现的正确性. 其次, 可以发现我们实现的单卡训练会比 PyTorch 实现更快, 这主要是由于 PyTorch 本身的框架执行代价, 以及它调用 CuSparse 库中的 SpMM 乘法所需的额外处理[25]. 我们通过人工实现 GPU kernel 避免了这些额外代价, 同时还引入了图重排以及稀疏块等 GPU 访存优化技术. 最终, 通过拓展到多卡, 我们在收敛速度上相比于 PyTorch 最多可以取得 10.9 倍的加速, 相比于我们的单卡实现最多可以取得 6.51 倍的加速.

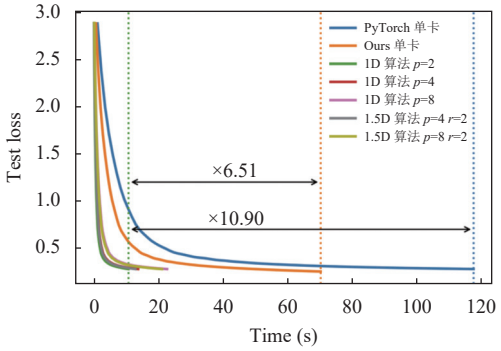
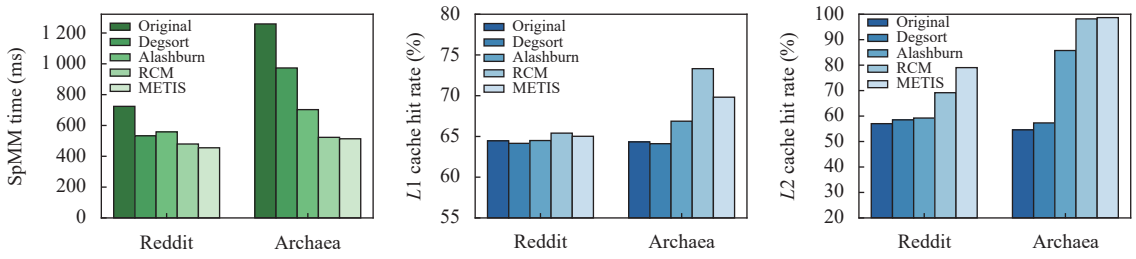


图 7 端到端收敛性能测试

5.3 图重排算法优化效果对比

本节旨在验证图重排算法对单卡 GNN 计算的加速有效性, 并挑选加速效果最佳的图重排算法, 作为分块图重排算法的组件算法, 拓展到第 5.4 节的分布式场景中. 本节选取较大规模的 Reddit 和超大规模的 Archaea, 使用 4 种重排算法 (Degsort, SlashBurn, METIS, RCM) 对全图进行重排, 比较 SpMM 算子 (AH) 的计算时间以及 GPU Cache 命中率. 其中, METIS 算法中我们固定簇大小为 200. 实验结果如图 8 所示: METIS 和 RCM 重排算法对于 SpMM 计算的 Cache 命中率提高有明显作用, 尤其是 L2 Cache 命中率, Reddit 上 METIS 和 RCM 分别达到近

80% 和 70%, 远高于未重排的命中率 57.03%; Archaea 上均达到近 99%, 远高于未重排的 55.71%, 可见 METIS 和 RCM 能够有效利用内存访问连续性提高 Cache 命中率, SpMM 计算时间也有显著提升, 在 Reddit 和 Archaea 上的计算加速比约为 1.5 倍和 2.4 倍. 相比之下, SlashBurn 和 Degsort 对于 Cache 命中率提升不大, 故带来的计算加速也有限. 因此, 本文选取 METIS 和 RCM 作为后续分布式 GNN 的基本分块图重排方案.



(a) SpMM 时间 (b) *L*1 缓存命中率 (c) *L*2 缓存命中率

图 8 图重排算法的选择

值得注意的是图重排方案的优化效果与图数据的具体分布相关. Reddit 作为一个真实的社交网络图, 具有幂律分布特性, 存在一些较大度数的节点, 即使通过 METIS 或 RCM 进行图重排之后, 这些节点的大量邻居节点也无法实现很好的内存连续访问效果. 如图 8 中实验所示, 蛋白质网络数据集 Archaea 可以取得比社交网络图 Reddit 更高的时间加速以及缓存命中率提升.

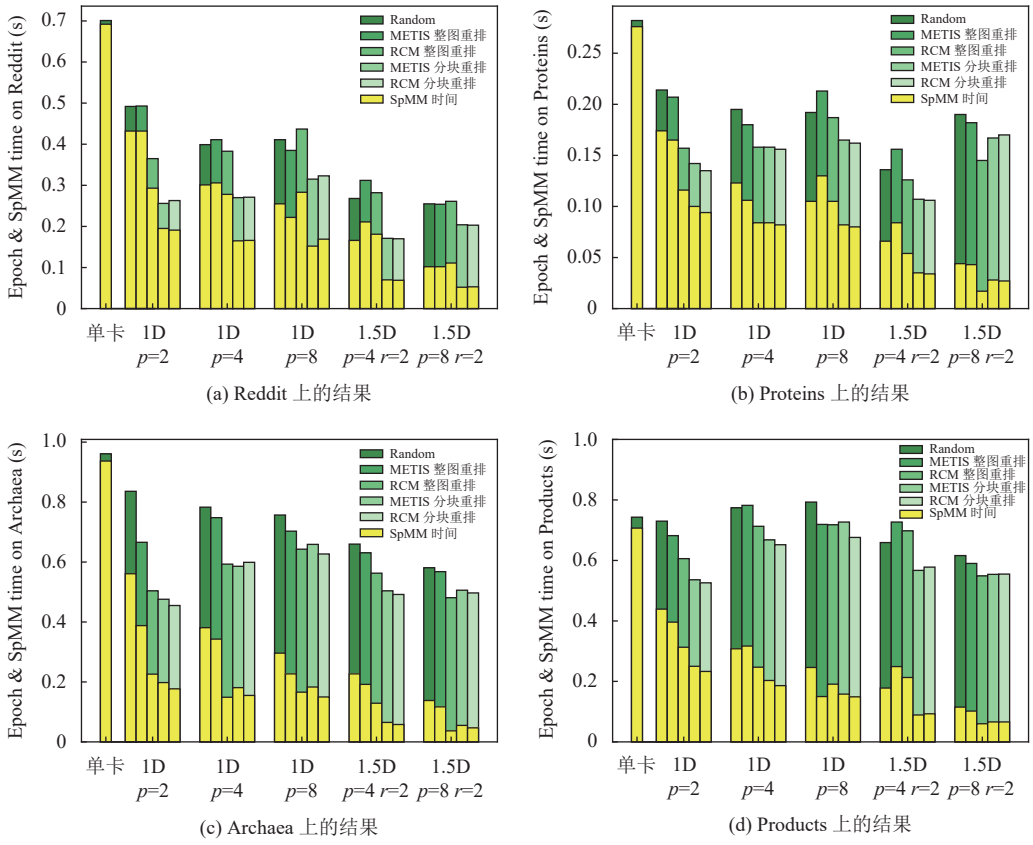
5.4 分块图重排算法优化结果

本节旨在验证分块图重排算法对分布式 GNN 训练计算加速的有效性. 本节中使用 METIS 和 RCM 分块图重排算法, 在 4 个数据集、5 种不同 GNN 切分方案 (见表 2) 下, 验证其相比于单卡执行的优化加速的有效性. 所有 GNN 模型均训练 100 轮, 记录其平均 Epoch 时间和 SpMM 时间. 实验结果如图 9 所示, 图重排的优化策略对于分布式 GNN 算法的计算时间有显著优化加速效果. 此外, 实验还分别测试了整图重排和分块图重排的效果, 可以看出分块图重排整体上比整图重排 SpMM 计算时间更少, 更高效. Reddit 上 METIS 和 RCM 分块图重排算法的计算加速比可达 2 倍以上, 其中 RCM 对 1.5D 算法 (p=4, r=2) 计算加速 2.41 倍; Proteins 和 Products 上最高计算加速比约为 2 倍; Archaea 上, 计算加速比可达 3–4 倍, 例如 RCM 对 1.5D 算法 (p=4, r=2) 计算加速 3.91 倍, 对 1D 算法 (p=4) 计算加速比 3.46 倍. 相应地, 由于计算的加速, 分块图重排算法能够有效缩短 Epoch 时间, 例如对于 1D 算法 (r=2), RCM 分块图重排在 Reddit 上将其整体加速 1.87 倍, Proteins 上加速 1.59 倍, Archaea 上加速 1.84 倍, Products 上加速 1.39 倍. 本文取得的主要加速来自每个 GPU 设备上计算速度的提升以及不同设备间的负载均衡. 另外, 我们也注意到, 在 Archaea 和 Products 这些超大规模数据集上通信时间已经远远超过计算时间, 未来可以通过采用 GPU 间 NVLink 总线来进行通信, 其理论传输带宽上限比我们实验环境配置的 PCIe3.0 总线带宽高 3 到 12 倍以上, 从而可以大幅缓解通信上的损耗, 提高 GPU 计算资源的利用率.

总体上来说, 在以上 4 个大规模的 GNN 数据集上, 分块图重排算法对分布式 GNN 的计算均有显著加速, 加速比可达两倍以上, Epoch 耗时也会显著缩短.

5.5 稀疏块自适应优化效果对比

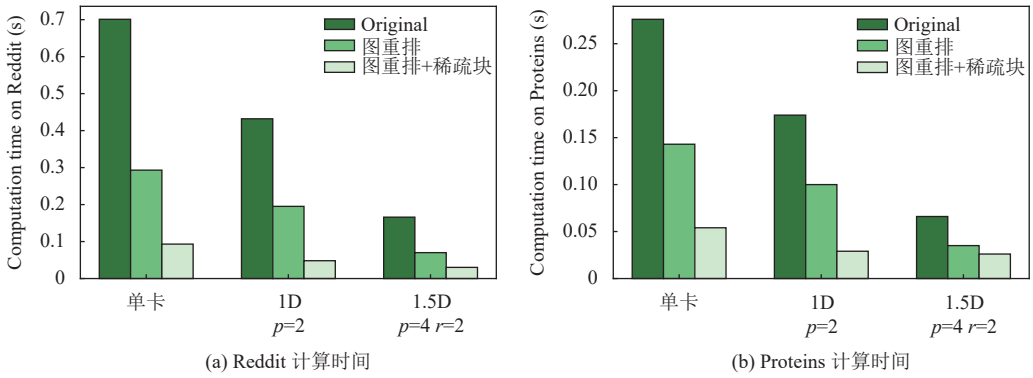
本节旨在验证稀疏块自适应优化技术对 GNN 训练加速的有效性, 本节在分块图重排技术的基础上, 进一步应用了稀疏块自适应优化技术. 实验选取了 Reddit 和 Proteins 数据集进行测试, 测量了 GNN 在单卡、1D (p=2) 切分, 1.5D (p=4, r=2) 切分情况下的计算时间. 从图 10 中可以看出, 在各种情况下, 稀疏块优化都可以显著地降低 GNN 计算所需的时间. 通过稀疏/稠密的自适应感知, 以及引入了高速低延迟的 GPU 共享内存, 大大提高了 GPU 的资源利用效果.



(a) Reddit 上的结果 (b) Proteins 上的结果

(c) Archaea 上的结果 (d) Products 上的结果

图 9 分块图重排的优化结果



(a) Reddit 计算时间 (b) Proteins 计算时间

图 10 稀疏块优化效果对比

5.6 端到端系统性能对比结果

为了对比本系统和现有 GNN 系统的性能差异, 本节对端到端的系统训练效率进行了测试. DGL 是目前性能最快的 GNN 训练系统之一, 但是目前只支持单卡训练, 所以本实验也只对单卡计算性能进行对比. 实验使用了 4 个测试数据集, 测量了系统每轮的执行时间, 如表 4 所示.

从表 4 中可以看出, DGL 相比于原生的 PyTorch 实现可以有 2–3.8 倍的性能提升, 这主要得益于 DGL 对 GNN 中消息传递机制的聚合优化以及人工实现的高性能内置函数, 这些定制化的优化技术使得 DGL 的系统效率

远高于普通的的 SpMM 实现. 因此, DGL 的训练性能会略优于本系统的基线版本. 但是, 在引入图重排技术后, 本系统的性能已经接近并超过了 DGL. 事实上, DGL 中的消息传递聚合优化技术也是在一定程度上对图中局部节点顺序进行了重新排布, 从而得到了更好的 GPU 访存效率. 进一步地, 本系统引入稀疏块自适应 GPU 优化技术后, 由于稠密/稀疏分离, 以及 GPU 上的访存性能优化, 本系统的性能得到了更显著的提升, 最终可以比 DGL 快 3.1–5.8 倍. 需要注意的是, 由于 Archaea 相对密集, 需要更多的显存资源, PyTorch 和 DGL 无法进行单卡训练.

表 4　端到端训练效率 (每轮执行时间) 对比 (s)

数据集	PyTorch	DGL	Ours (基线)	Ours+图重排	Ours+图重排+稀疏块
Reddit	1.187	0.587	0.710	0.302	**0.102**
Proteins	0.714	0.186	0.285	0.149	**0.060**
Archaea	OOM	OOM	0.980	0.306	**0.107**
Products	1.537	0.672	0.764	0.379	**0.136**

5.7　密集度阈值影响测试

上述实验中密集度阈值默认设置为 5%, 为了进一步探究密集度阈值设置对系统性能的影响, 本节测试了系统在密集度阈值分别调整为 2%, 5% 以及 10% 情况下的训练效率. 如表 5 所示, 在 Archaea 和 Products 上, 阈值设置为 5% 可以得到最优的执行时间, 而在 Reddit 和 Proteins 上即使把阈值提升到 10%, 仍然可以获得性能的提升. 这种差异主要来源于数据集本身的特性 (如表 3 所示), Archaea 和 Products 规模更大, 边更稀疏, 所以会对密集度阈值更敏感. 如果密集度阈值过小, 则无法充分发挥稀疏优化的效果, 如果阈值过大, 则可能会使得过多的分块采用 SpMM 计算, 从而导致负优化. 在实际使用中, 应当根据目标数据集对该阈值进行调整, 以达到更优的训练效率.

表 5　不同密集度阈值下训练效率 (每轮执行时间) 对比 (s)

数据集	2%	5%	10%
Reddit	0.147	0.102	**0.097**
Proteins	0.100	0.060	**0.057**
Archaea	0.205	**0.107**	0.161
Products	0.162	**0.136**	0.191

6　总　结

本文研究了面向多 GPU 的图神经网络训练的加速优化, 基于 1D 和 1.5D 的分布式 GNN 切分算法将稀疏矩阵乘法计算部署到多个 GPU 上. 针对目前 GPU 计算实现方案中忽视图数据稀疏分布的缺陷, 利用图重排技术, 提高内存访问连续性和 Cache 命中率, 加速稀疏矩阵乘法计算, 并在负载均衡的基础上拓展到多 GPU 切分的场景下. 更进一步, 提出了稀疏块感知的 GPU 访存优化技术, 引入 GPU 共享内存, 提高了 GPU 上 GNN 的执行效率. 在多个大规模数据集上进行了实验, 结果表明本文提出的优化方法可以使多卡 GNN 训练加速计算两倍以上, 原型系统可以比目前的 GNN 训练系统 DGL 快 5.8 倍. 未来我们还将探索通过 2D 切分等更多复杂的切分方式来提高并行执行效率.

References:

[1] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017. 1–14.

[2] Zhang WT, Miao XP, Shao YX, Jiang JW, Chen L, Ruas O, Cui B. Reliable data distillation on graph convolutional network. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 1399–1414. [doi: 10.1145/3318464.3389706]

[3] Miao XP, Gürel NM, Zhang WT, Han ZC, Li B, Min W, Rao SX, Ren HS, Shan YN, Shao YX, Wang YJ, Wu F, Xue H, Yang YM,

Zhang ZT, Zhao Y, Zhang S, Wang YJ, Cui B, Zhang C. DeGNN: Improving graph neural networks with graph decomposition. In: Proc. of the 27th ACM SIGKDD Conf. on Knowledge Discovery & Data Mining. ACM, 2021. 1223–1233. [doi: 10.1145/3447548.3467312]

[4] Miao XP, Zhang WT, Shao YX, Cui B, Chen L, Zhang C, Jiang JW. Lasagne: A multi-layer graph convolutional network framework via node-aware deep architecture. IEEE Trans. on Knowledge and Data Engineering, 2021. [doi: 10.1109/TKDE.2021.3103984]

[5] Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 1025–1035.

[6] Zhang MH, Li P, Xia YL, Wang K, Jin L. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In: Proc. of the 35th Conf. on Neural Information Processing Systems. 2021. 34.

[7] Wu F, Souza Jr AH, Zhang TY, Fifty C, Yu T, Weinberger KQ. Simplifying graph convolutional networks. In: Proc. of the 36th Int'l Conf. on Machine Learning. Long Beach: PMLR, 2019. 6861–6871.

[8] Abu-El-Haija S, Kapoor A, Perozzi B, Lee J. N-GCN: Multi-scale graph convolution for semi-supervised node classification. In: Proc. of the 35th Conf. on Uncertainty in Artificial Intelligence. Tel Aviv: PMLR, 2019. 841–851.

[9] He XN, Deng K, Wang X, Li Y, Zhang YD, Wang M. LightGCN: Simplifying and powering graph convolution network for recommendation. In: Proc. of the 43rd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. 2020. 639–648. [doi: 10.1145/3397271.3401063]

[10] Chen J, Ma TF, Xiao C. FastGCN: Fast learning with graph convolutional networks via importance sampling. In: Proc. of the 6th Int'l Conf. on Learning Representations. Vancouver: OpenReview.net, 2018. 1–15.

[11] Zeng HQ, Zhou HK, Srivastava A, Kannan R, Prasanna VK. GraphSAINT: Graph sampling based inductive learning method. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: OpenReview.net, 2020. 1–19.

[12] Wang M, Yu L, Zheng D, Gan, Q, Gai Y, Ye Z, Zhang Z. Deep graph library: Towards efficient and scalable deep learning on graphs. In: Proc. of the Int'l Conf. on Learning Representations. 2019. 1–18.

[13] Fey M, Lenssen JE. Fast graph representation learning with PyTorch Geometric. arXiv:1903.02428, 2019.

[14] Yang HX. Aligraph: A comprehensive graph neural network platform. In: Proc. of the 25th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. Anchorage: ACM, 2019. 3165–3166. [doi: 10.1145/3292500.3340404]

[15] Tripathy A, Yelick K, Buluç A. Reducing communication in graph neural network training. In: Proc. of the 2020 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. Atlanta: IEEE, 2020. 1–14. [doi: 10.1109/SC41405.2020.00074]

[16] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin ZM, Gimelshein N, Antiga L, Desmaison A, Köpf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai JJ, Chintala S. PyTorch: An imperative style, high-performance deep learning library. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc, 2019. 721.

[17] Jia ZH, Lin SN, Gao MY, Zaharia M, Aiken A. Improving the accuracy, scalability, and performance of graph neural networks with ROC. In: Proc. of 2020 Machine Learning and Systems. Austin: Mlsys.org, 2020. 187–198.

[18] Miao XP, Nie XN, Shao YX, Yang Z, Jiang JW, Ma LX, Cui B. Heterogeneity-aware distributed machine learning training via partial reduce. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 2262–2270. [doi: 10.1145/3448016.3452773]

[19] Lim Y, Kang U, Faloutsos C. SlashBurn: Graph compression and mining beyond caveman communities. IEEE Trans. on Knowledge and Data Engineering, 2014, 26(12): 3077–3089. [doi: 10.1109/TKDE.2014.2320716]

[20] George A, Liu JW. Computer Solution of Large Sparse Positive Definite. Englewood Cliffs: Prentice Hall Professional Technical Reference, 1981.

[21] Karypis G, Kumar V. Parallel multilevel series *k*-way partitioning scheme for irregular graphs. SIAM Review, 1999, 41(2): 278–300. [doi: 10.1137/S0036144598334138]

[22] Tian C, Ma LX, Yang Z, Dai YF. PCGCN: Partition-centric processing for accelerating graph convolutional network. In: Proc. of the 2020 IEEE Int'l Parallel and Distributed Processing Symp. New Orleans: IEEE, 2020. 936–945. [doi: 10.1109/IPDPS47924.2020.00100]

[23] Azad A, Pavlopoulos GA, Ouzounis CA, Kyrpides NC, Buluç A. HipMCL: A high-performance parallel implementation of the Markov clustering algorithm for large-scale networks. Nucleic Acids Research, 2018, 46(6): e33. [doi: 10.1093/nar/gkx1313]

[24] Miao XP, Zhang HL, Shi YN, Nie XN, Yang Z, Tao YY, Cui B. HET: Scaling out huge embedding model training via cache-enabled distributed framework. Proc. of the VLDB Endowment, 2021, 15(2): 312–320. [doi: 10.14778/3489496.3489511]

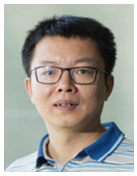
[25] Miao XP, Ma LX, Yang Z, Shao YX, Cui B, Yu LL, Jiang JW. CuWide: Towards efficient flow-based training for sparse wide models on gpus. IEEE Trans. on Knowledge and Data Engineering, 2020. [doi: 10.1109/TKDE.2020.3038109]

苗旭鹏(1995－), 男, 博士, CCF 学生会员, 主要研究领域为人工智能系统, 分布式系统, GPU 性能优化, 图表示学习.

王驭捷(1998－), 男, 博士, CCF 学生会员, 主要研究领域为分布式深度学习系统, 图神经网络.

沈佳(1997－), 女, 硕士, 主要研究领域为分布式深度学习系统, 图神经网络.

邵荃侠(1988－), 男, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为数据库, 知识图谱数据管理, 并行图计算.



崔斌(1975－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库, 人工智能系统, 大数据管理分析.