中国科学：信息科学

SCIENTIA SINICA Informationis

论文

面向 CPU-GPU 异构环境下的高性能超图神经网络加速系统

余辉[1, 2, 3, 4]，张宇[1, 2, 3, 4*]，李鑫滔[1, 2, 3, 4]，陈子康[1, 2, 3, 4]，赵英淇[1, 2, 3, 4]，赵进[1, 2, 3, 4]，齐豪[1, 2, 3, 4]，廖小飞[1, 2, 3, 4]，金海[1, 2, 3, 4]

1. 华中科技大学大数据技术与系统国家地方联合工程研究中心, 武汉 430074, 中国
2. 华中科技大学服务计算技术与系统教育部重点实验室, 武汉 430074, 中国
3. 华中科技大学集群与网格计算湖北省重点实验室, 武汉 430074, 中国
4. 华中科技大学计算机科学与技术学院, 武汉 430074, 中国

* 通信作者. E-mail: zhyu@hust.edu.cn

湖北省重点研发计划项目 (NO. 2023BAB078)，国家重点研发计划项目（NO. 2024YFB4504200），国家自然科学基金（NO. 62472183），本研究成果得到 CCF-蚂蚁科研基金资助（NO. CCF-AFSG RF20240204）

摘要 图神经网络因其在处理非欧几里得空间数据方面的卓越学习和推理能力而备受关注. 然而, 现实世界中的图结构通常包含顶点之间复杂的高阶关联关系, 这类图通常被定义为超图. 为了在超图上有效捕获这些高阶复杂特征和潜在语义信息, 众多超图神经网络模型被相继提出. 尽管目前已有多个利用 GPU 加速器的高并行计算能力来加速超图神经网络训练的系统软件, 但在加速模型训练过程中, 这些系统仍面临大量冗余计算和频繁的数据通信问题, 导致 GPU 利用率较低. 此外, 这些系统仅支持图规模较小的超图神经网络模型训练. 针对上述问题, 我们发现超图神经网络训练过程中, 由于超图中多条超边之间存在较强的拓扑重叠特性, 因此这些超边的计算会重复获取和计算相同图顶点的特征向量. 基于这一观察, 我们提出了一个面向 CPU-GPU 异构环境的高性能超图神经网络系统（Redundancy-elimination HyperGraph Neural Network，RHGNN）. 该系统充分利用超边间的重叠特性指导超边和顶点特征计算与更新, 有效减少 CPU-GPU 数据通信量. 具体而言, RHGNN 提出了一种新颖的以超边为中心的冗余消除执行方法, 使系统以最优方式完成顶点特征向量的通信加载, 实现一次加载服务多次超边和顶点特征计算. 同时, RHGNN 通过高效的超度感知层次数据缓存机制, 在 GPU 端优先缓存频繁访问的顶点和超边特征向量以及超边中共享顶点的中间结果值, 进一步减少数据通信开销. 为验证 RHGNN 的有效性, 我们将其与当前最先进的超图神经网络软件系统 DGL、PyG 以及 HyperGef 进行了性能对比. 实验结果表明, 在支持超图神经网络模型训练方面, RHGNN 相较于 PyG、DGL 和 HyperGef 的性能分别提升了 2.5-3.4 倍、2.1-3.1 倍和 1.4-2.3 倍.

关键词 超图神经网络, 训练加速, CPU-GPU 异构, 冗余消除, 拓扑相似性

1 引言

近年来, 图神经网络 (Graph Neural Network, GNN) 因其出色的图结构数据处理能力, 在药物制造 [1∼3]、知识图谱 [4∼6]、决策辅助 [7∼9] 等国民经济和国防军事领域得到了广泛应用. 尽管 GNN 能够高效地学习和训练图数据, 但它们主要集中在对传统二元边之间的关系进行建模, 即仅捕捉顶点之间的简单关联. 然而, 现实世界中的图往往具有更为复杂的关系结构. 例如, 在社交网络、分子结构或交通网络中, 多个顶点之间存在高阶关联关系 [10]. 这类复杂的多顶点关联关系难以被普通的 GNN 学习与捕捉, 从而限制了其在更复杂场景下的表现. 这种一条边可以连接任意数量顶点的图类型被称为超图 (Hypergraph) [11]. 为了针对超图结构进行高效学习, 超图神经网络 (Hypergraph Neural Network, HyperGNN) 应运而生 [12, 13]. HyperGNN 通过引入超边 (Hyperedge) 的概念, 能够直接对超图中的高阶关系进行建模和学习, 充分挖掘顶点之间的复杂交互信息. 例如, 在真实的生物网络中, 蛋白质复合体通常涉及多个蛋白质节点, 这种多元关系无法用传统的二元图关系表示。因此, 超图提供了更为丰富的建模能力, 能够更好地揭示复杂的多层次关联。与传统 GNN 相比, HyperGNN 在处理包含高阶关系的数据时展现出显著优势. 首先, HyperGNN 能够更准确地捕获节点之间的多样化联系, 提取更丰富的结构特征. 其次, 通过超边的引入, HyperGNN 能够有效缓解 GNN 中的过度平滑问题, 提高模型的表达能力. 此外, HyperGNN 在处理不平衡数据时也表现出色, 能够更好地适应小样本超图数据学习. 并且, 从定义上来看, 图神经网络实际上是超图神经网络的一种特殊形式, 后者能够涵盖前者无法表达的高阶关联. 综上所述, HyperGNN 凭借其强大的建模和推理能力, 在超图结构数据的学习和分析任务中展现出巨大的应用潜力 [12, 14, 15].

为了有效支持 HyperGNN 模型的训练, 一些软件系统被相继提出, 例如 DGL [16] 和 HyperGef [12]. 它们利用 GPU 的高性能并行计算能力对 HyperGNN 的训练过程进行加速, 但这些系统仅支持将超图拓扑数据和特征向量数据全部存储在 GPU 主存的训练方式. 对于较大规模的 HyperGNN 模型训练, 完整的超图拓扑和特征向量数据往往无法整体加载到 GPU 内存中 [12, 17], 严重制约了这些系统的可扩展性. 为了有效支持大规模普通图神经网络的训练, 也有一些基于 CPU-GPU 异构环境下的图神经网络系统被提出, 例如 NeuGraph [18] 和 ROC [19]. 这些方案都是将需要训练的目标顶点划分为多个批次 (batches), 然后将每个批次的数据作为最基本的训练单元. NeuGraph 提出了 SAGA-GNN 编程模型, 允许数据以更细粒度的方式传输到 GPU 内存中进行训练, 从而加快 CPU-GPU 的数据通信效率. 但是, 它们只支持普通图数据, 对于超边引入的高阶连接关系和更复杂的拓扑结构, 其性能会大幅下降. 虽然 CPU-GPU 异构资源可以有效拓展 HyperGNN 模型训练的规模, 但在 CPU-GPU 异构环境下支持大规模 HyperGNN 模型训练仍然面临以下两个挑战.

首先, HyperGNN 模型训练过程中存在大量的冗余计算[1)]. 这是因为现有解决方案需要对每条超边分别进行计算和更新. 具体来说, 在 HyperGNN 的超边聚合操作中, 每条超边都需要独立地获取其所连接的顶点的特征向量, 并执行相应的计算. 然而, 在现实的超图拓扑中, 不同的超边之间往往存在着拓扑重叠, 即多个超边可能连接着大量相同的顶点子集. 这意味着, 对于重叠的顶点, 其特征向量会被多个超边重复获取和计算, 导致了大量的冗余计算和访问.

其次, 与普通图相比, 超图中的每条超边包含多个图顶点, 这意味着每条超边可能涉及多个顶点特征向量的读取与计算. 由于这些特征向量通常需要在多个超边中被多次访问, 导致了大量的 CPU-

1) 冗余计算指的是在超图神经网络训练过程中，相同的顶点和超边特征向量被多次加载和计算

GPU 数据传输开销. 特别是, 当超边之间存在大量的拓扑重叠时, 这种重复访问加剧了数据通信的频繁性和复杂性. 这是因为图顶点的特征向量维度通常较高 [17], 这些高维度的特征向量往往无法完全驻留在 GPU 内存中, 因此必须频繁从 CPU 内存中加载. 这种频繁的数据传输不仅增加了通信延迟, 还加剧了由于图神经网络聚合操作带来的内存瓶颈, 进一步降低了 GPU 的利用效率, 使得系统无法充分发挥 GPU 并行计算的潜力.

为了有效解决上述两个问题, 我们对 HyperGNN 模型训练进行了详细分析, 得出了以下两个观察. 首先, 超图中存在着**超边重叠**特性, 即多条超边之间往往共享大量相同的顶点. 这种超边重叠的拓扑特性为我们提供了减少冗余数据计算和访问的机会, 通过对共享顶点的特征向量进行合并和一次性更新, 可以显著减少不必要的冗余计算和访问. 其次, 超图与普通图一样也符合幂律分布特性, 即少数顶点和超边的度值 (连接到该顶点的超边数量/该超边包含的顶点数量) 非常高, 而大多数顶点和超边的所关联的邻居数量较少. 通过对现实世界超图数据集的分析, 我们发现顶点/超边度值的分布通常呈现出明显的长尾特征 [20, 21]. 这意味着, 少数顶点/超边的特征向量会被频繁地访问和参与计算操作. 结合这两个特性, 我们可以充分挖掘 HyperGNN 模型的并行训练潜力, 从而高效地加速训练效率.

基于上述两个关键观察, 我们设计了一个面向 CPU-GPU 异构环境的高性能超图神经网络系统 (Redundancy-elimination HyperGraph Neural Network, RHGNN), 它能够通过减少模型训练的冗余数据通信和计算来提高计算效率. 具体来说, RHGNN 采用一种新颖的以超边为中心的冗余消除执行方法, 通过将具有高重叠度的超边划分到同一个计算块中, 实现对该块内的相同顶点特征向量的复用访问和计算. 该方法的核心思想是首先按照超边之间的相似度进行块划分, 即满足同一块内多条超边拥有尽可能多的共享顶点, 从而实现传输尽可能少的顶点特征向量来满足该块内尽可能多条超边的计算需求, 最大限度地减少超边和顶点的冗余计算. 同时, RHGNN 还提出一种超度感知的层次缓存策略, 即根据超边块的优先级和相邻超边块之间的顶点数据相似性, 优先将超边块间的高相似顶点[2)] 特征存储在 GPU 的共享内存中, 以减少跨块访问时的通信开销. 这是因为高相似度顶点会参与块内的多条超边的超边聚合计算, 因此可以进一步减少块内的数据访问开销。同时, 将高度顶点的特征存储在 GPU 全局内存中, 确保这些频繁访问的顶点能够被快速访问. 本文的主要贡献如下:

- 观察到现有的动态图神经网络系统的不足, 并揭示现有模型训练性能不佳的主要原因.
- 提出一个面向 CPU-GPU 异构环境下的高性能超图神经网络系统 RHGNN, 它能够有效地减少大规模超图神经网络模型训练的冗余数据传输和计算开销. 同时, RHGNN 还采用超度感知层次缓存策略, 进一步减少数据通信开销.
- 将 RHGNN 与目前最好的 HyperGNN 软件系统 PyG、DGL 和 HyperGef 进行对比. 实验结果表明, RHGNN 将其数据通信量分别减少 47.2%-62.3%、44.8%-67.8% 和 27.1%-42.2%, 性能分别提升 2.5-3.4 倍、2.1-3.1 倍和 1.4-2.3 倍.

本文的其余部分组织如下: 第 2节介绍超图神经网络的背景和现有方案的不足; 第3节介绍我们核心方法的细节; 第4节介绍系统的设计与实现; 第5节进行实验和结果分析; 第6节介绍论文相关工作; 第7 节总结本文工作.

2) 高相似度顶点被定义为在一个超边块内, 除共享顶点外, 具有最高度的顶点

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

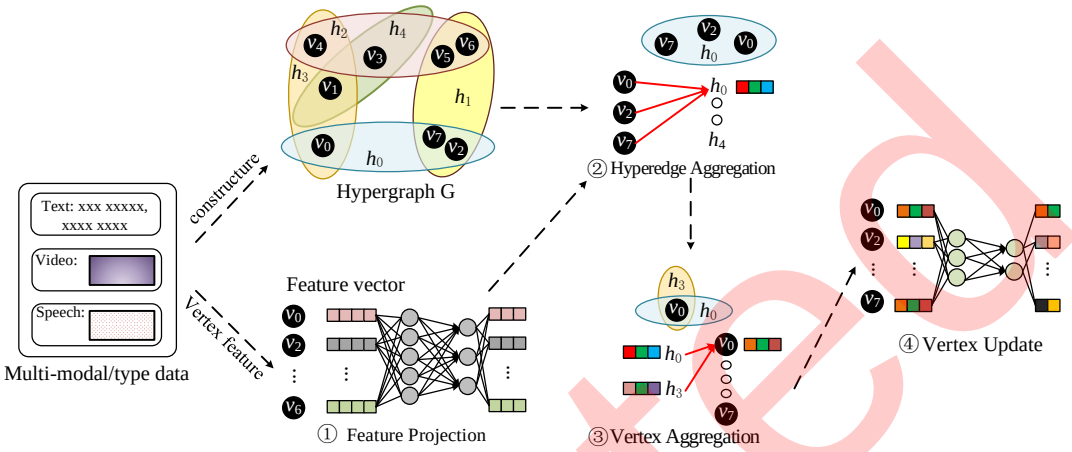

图 1　超图神经网络模型示例

Figure 1　An illustrative example for hyperGNN model

2　背景和动机

2.1　超图神经网络基础概念

超图 [11, 14, 22]. 超图通常可以定义为 $G = (V, H)$, 其中 V 代表为超图的顶点集合, H 表示为超边集合. 与普通图不同的是, 超图的超边是一条包含任意数量顶点的边. $|V|$ 和 $|H|$ 分别表示为超图中顶点的数量和超边数量. 对于给定的超边 h 和顶点 v, 超边度 $deg(h)$ 表示超边 h 所包含的顶点数量, 顶点度 $deg(v)$ 表示与顶点 v 相连的超边数量. $N(h)$ 表示由超边 h 连接的顶点集合, $N(v)$ 表示与顶点 v 相连的超边集合. 超图中的顶点和超边都具有特征, 分别用特征向量 $X(v)$ 和 $X(e)$ 表示. 当两条超边 h_i 和 h_j 至少有一个共同的顶点时 (即 $N(h_i) \cap N(h_j) \neq \emptyset$), 这些共享的顶点被称为公共顶点.

超图神经网络 [12, 13, 23]. 超图神经网络（Hypergraph Neural Networks, HyperGNN）是一种专门用于处理超图结构数据的深度学习模型. 与传统 GNN 不同, HyperGNN 能够直接建模超图中的高阶关系, 捕捉更为复杂的结构信息, 从而更准确地表征顶点和超边之间的多重关联. 如图 1 所示, 超图神经网络的推理过程通常分为四个关键步骤. 首先是特征映射阶段（Feature Projection, FP）, 在该阶段, 顶点的原始特征（即特征向量）通过神经网络映射到一个更高维度的特征空间中, 为后续的特征聚合提供更丰富的表达. 接下来是超边聚合阶段（Hyperedge Aggregation, HA）, 在此过程中, 超边内的所有顶点特征通过聚合操作生成超边的特征表示, 以捕捉多个顶点之间的高阶关联. 随后是顶点聚合阶段（Vertex Aggregation, VA）, 该阶段将超边的特征信息传递回与其相连的顶点, 通过聚合与每个顶点关联的超边特征, 进一步丰富顶点的特征表达. 最后是特征更新阶段（Vertex Update, VU）, 通过额外的神经网络层对顶点的特征向量进行更新, 将聚合后的超边和顶点信息综合处理, 生成最终的顶点特征表示.

超图神经网络遵循消息传递的机制, 其中信息在超图结构中传播和聚合. 在这个过程中, 每个超

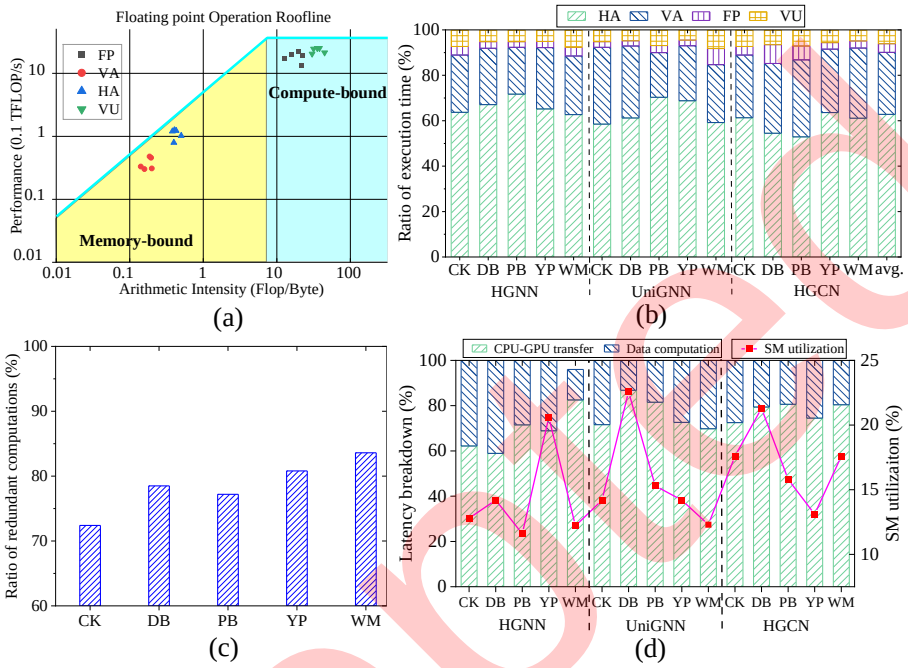

图 2 HyperGef 在 HyperGNN 模型训练中的性能分析: (a) 在 NVIDIA Tesla V100 上训练 HGNN 模型的 Roofline 分析; (b) HyperGef 在不同数据集和模型上的训练时间分解; (c) HyperGef 训练 HGNN 模型时冗余数据计算占所有数据计算的比例; (d) HyperGef 在 NVIDIA Tesla A100 上训练不同模型时的时延分解和 SM 利用率

Figure 2 Studies of the performance of HyperGNN model: (a) Roofline analysis of the HyperGNN model on NVIDIA Tesla V100; (b) breakdown of training time for HyperGef across different datasets and models; (c) ratio of redundant computations to total vertex computations in HGNN training on HyperGef; (d) latency breakdown and SM utilization of HyperGef while training different models on NVIDIA Tesla A100

边和顶点的特征被更新并传递. 其基本消息传递可以通过以下公式表示:

$$\begin{aligned} X(h_e)^l &= F_1\left(U_1\left(X(x_j)^{(l-1)} | v_j \in e\right)\right) \\ X(X_i)^l &= \sigma\left(U_2\left(F_2\left(X(h_e)^l | e \in N(v_i)\right)\right)\right) \end{aligned} \tag{1}$$

其中, $X(x_i)$ 和 $X(h_e)$ 分别表示顶点 x_i 和 h_e 的特征. F_1, F_2, U_1, U_1 和σ 分别表示消息传递中的超边聚合函数、顶点聚合函数、超边更新函数、顶点更新函数和激活函数.

2.2 超图神经网络模型特性

超图数据存储. 类似于普通图, 超图也可以采用压缩稀疏行 (Compressed Sparse Row, CSR) 格式进行存储 [3,5,8,9], 以节省存储空间. 超图需要分别对顶点和超边进行存储, 在这里我们只考虑基于 CSR 的顶点存储, 因为超边的 CSR 格式与顶点的 CSR 格式类似. 顶点的 CSR 格式包含三个数组: 顶点偏移 (Vertex Offset)、超边索引 (Hyperedge Index) 和顶点特征 (Vertex Features). 超边索引数组存储与每个顶点相关的超边信息. 顶点偏移数组指示某个顶点的超边在超边索引数组中的起始和结束位置. 每个顶点的特征存储在顶点特征数组中.

超图神经网络 vs 图神经网络. (1) 边聚合与超边聚合. 传统 GNN 中, 顶点的特征通过与其相连的边进行聚合, 这些边连接两个顶点. 而在 HyperGNN 中, 引入了超边聚合, 它将超边内任意

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

数量顶点的特征进行结合. (2) 单次聚合与多级聚合. GNN 通常在一次聚合操作中完成邻域信息的聚合. 相比之下, HyperGNN 采用多级聚合策略, 首先进行从顶点到超边的信息聚合, 然后再进行从超边到顶点的后续聚合. 通过这种方式, HyperGNN 能够准确建模并捕捉高阶关系的细微差异.

2.3 现有方案存在的问题

现有系统在支持 HyperGNN 模型训练时, 通常面临着大量的冗余计算和高额的数据移动开销. 具体来说, 首先是它们会重复多次获取同一顶点的特征向量, 这是因为超边之间存在着拓扑重叠的特性. 其次是它们会重复获取公共子集的部分聚合值, 这是因为多条超边共享的顶点集合往往会导致重复的聚合计算. 最后是它们会冗余获取同一超边的特征向量. 特别地, 当相同的超边特征在不同计算阶段或多个顶点上被重复访问时, 会进一步增加冗余的计算负担和数据传输开销.

为了证明上述问题, 我们对现有主流的 HyperGNN 系统, 即 HyperGef [12], 在不同的数据集 (Cooking200 (CK), CoauthorshipDBLP (DB), CocitationPubmed (PB), YelpRestaurant (YP) 和 WalmartTrips (WM)) 上支持不同的 HyperGNN 模型 (HGNN, HGCN 和 UniGNN) 训练进行了测试与分析, 测试的平台、模型和数据集的详细信息请查看第 5.1 章节. 图 2 (a) 描述了 HyperGef 在 DB 数据集上训练 HGNN 模型的 Roofline 分析, 可以看出, HA 和 VA 阶段表现为内存受限, 而 FP 和 VU 则是计算密集型. 总而言之, 内存访问显然是导致 HyperGNN 模型性能瓶颈的主要原因, 尤其是在 HA 和 VA 阶段, 内存带宽的不足进一步限制了系统的训练效率. 图 2 (b) 展示了 HyperGef 在不同数据集上训练不同 HyperGNN 模型的执行时间分解, 可以看出超边聚合和顶点聚合的执行时间占总执行时间的 67.5%-83.8%, 成为了 HyperGNN 模型训练的主要瓶颈, 而且还限制了 GPU 计算资源的充分利用. 这一现象主要由于以下两个原因导致的:

冗余数据计算开销. 我们用图 1 的例子来解释 HyperGNN 模型训练过程中涉及的冗余计算. 在执行 h_0 超边聚合时会获取顶点 v_0, v_2 和 v_7 的特征向量, 然后使用归约函数将它们聚合成超边 h_0 的特征向量. 同样地, 在执行 h_1 超边聚合时会获取顶点 v_2, v_7, v_5 和 v_6 的特征向量, 从而聚合成超边 h_1 的特征向量. 这里面就涉及到对顶点 v_2 和 v_7 特征向量的重复获取. 在执行 v_0 的顶点聚合时会获取超边 h_0 和 $h3$ 的特征向量, 从而聚合成顶点 v_0 的特征向量. 相似地, 在执行 v_2 的顶点聚合时会重复获取超边 h_0 特征向量. 如图 2 (c) 所示, 超过 82.3% 的顶点和超边计算是冗余的.

频繁数据移动开销. 在现有 HyperGNN 训练过程中, 由于大多数顶点特征向量和反向传播传播中的梯度参数的大小远超 GPU 内存容量, 必须通过频繁的 CPU-GPU 通信来访问这些数据, 以完成超边和顶点的特征更新. 这种频繁的数据移动不仅增加了系统的通信开销, 还导致了大量的内存带宽浪费. 此外, 由于多条超边共享存在大量的公共顶点, 这种特征向量的重复访问引发了大量冗余计算, 使得同一顶点的特征向量在不同超边计算中被多次读取. 这种冗余的数据访问模式进一步削弱了内存系统的效率, 造成了严重的内存带宽不足和 GPU 内存利用率低下问题. 如图 2 (d) 所示, CPU-GPU 通信的大部分数据 (超过 76.2%) 在 HyperGef 中是冗余的, 因为它们在不同超边/顶点计算中是相同的, 导致内存带宽和 GPU 利用率不足.

2.4 创新与动机

根据图 3 展示的 HyperGNN 模型训练基本特性, 我们得出以下两个观察.

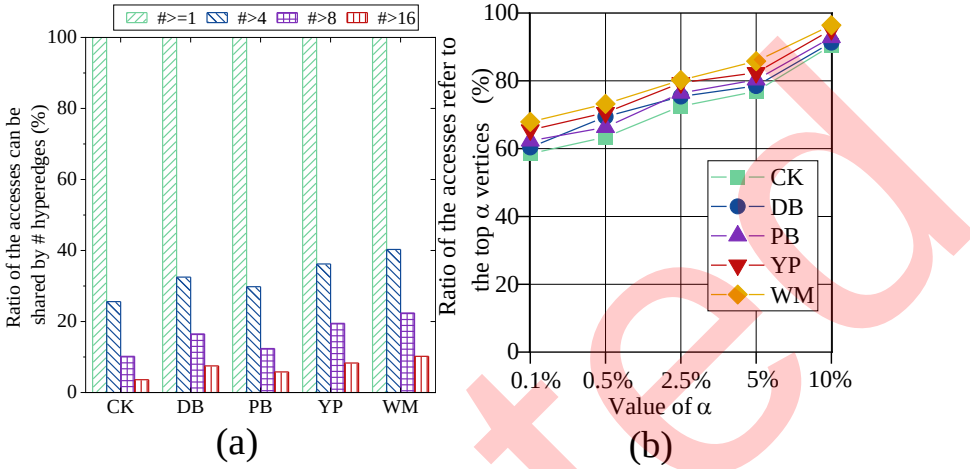

图 3 HyperGNN 的数据访问研究：(a) 被不同数量超边共享的顶点数量占总顶点数量的比例；(b) 最高优先级的前 α 个顶点的输入特征访问次数与所有顶点的输入特征访问次数的比例

Figure 3 Data access studies of the HyperGNN model: (a) the ratio of the accesses of the vertices data that can be shared by different number of hyperedges; (b) the ratio of the accesses refer to the input features of the top α vertices to those of all vertices

观察 1. *HyperGNN* 模型训练具有显著的**时间相似性**，即当多条超边在同时执行超边聚合计算时，它们往往会在短时间内频繁访问相同顶点的特征向量. 这种时间相似性源于超图中普遍存在的超边重叠现象，即不同超边之间通常共享大量相同的顶点. 图 3 (a) 显示了不同数据集中平均被 4 条超边共享的顶点数量占总顶点数量的百分之 32.9%. 更有甚者，平均被 8 条超边共享的顶点数量占总顶点数量的百分之 16.2%. 当多条超边之间共享的顶点数量增加时，共享顶点的特征向量会出现更多的冗余. 这是因为在 HyperGNN 模型的训练过程中，既需要对每条超边执行聚合操作，又需要对每个顶点进行超边聚合操作. 具体而言，在执行超边聚合时，如果多条超边共享同一顶点，这些共享顶点的特征向量将在每次聚合过程中被反复访问和计算. 由于同一顶点的特征向量会被多个超边重复引用和处理，这种重复操作不可避免地导致了大量冗余的数据访问和计算.

观察 2. *HyperGNN* 模型训练还表现出明显的**空间相似性**，即不同超边聚合和顶点聚合计算中通常会访问一小部分顶点和超边的特征向量. 为了评估这一特性，我们统计了度数排名前 α 的顶点特征向量的访问次数占总访问次数的比例，其中顶点按照其度数降序排列. 图 3 (b) 显示，超过 73.4% 的访问次数集中在度数排名前 0.5% 的顶点特征向量上. 这表明，少数的高度顶点在 HyperGNN 模型训练过程中扮演着至关重要的角色，其特征向量会被频繁地访问和参与计算. 因此，我们可以将这些高度顶点的特征向量也缓存在片上存储器中，进一步减少数据传输开销.

3 核心方法

现有 HyperGNN 模型训练方法 [12, 16, 24] 采用以超边或顶点的顺序索引方式来构建超边和顶点邻接矩阵，并据此执行相关计算和梯度更新. 然而，这种传统的训练模式忽略了超边之间的拓扑重叠特性，导致大量冗余计算和频繁的数据移动，降低了训练效率. 为了有效解决这两个关键挑战，本文提出了一种创新的以超边为中心的冗余消除执行方法，并设计了超度感知的层次缓存策略，以期显著提升 HyperGNN 在 CPU-GPU 异构环境下的训练性能.

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

Algorithm 1 基于动态阈值的超边块划分策略

Input: Hyperedge set H, base threshold θ_0, similarity contribution factor γ, maximum block size $S_{\max}$
Output: Hyperedge block partition result P

$P \leftarrow \emptyset$; /* Initialize partition result*/
Compute the topological similarity matrix $TS(h_i, h_j)$ for all hyperedges;
$visited \leftarrow \emptyset$; /* Initialize hyperedge visit frontier*/
for each hyperedge $h_i \in H$ **do**
  if $h_i \notin visited$ **then**
    $visited \leftarrow visited \cup \{h_i\}$;
    $B \leftarrow \{h_i\}$; /* Initialize a new partition block*/
    for each hyperedge $h_j \in H \setminus visited$ **do**
      $\theta(h_i, h_j) = \theta_0 + \gamma \cdot \frac{1}{\sum_{v \in N(h_i) \cap N(h_j)} \deg(v)}$; /* Compute dynamic threshold for each pari hyperedge */
      if $TS(h_i, h_j) \geqslant \theta(h_i, h_j)$ **and** $|B| < S_{\max}$ **then**
        $B \leftarrow B \cup \{h_j\}$;
        $visited \leftarrow visited \cup \{h_j\}$;
      end if
    end for
    $P \leftarrow P \cup \{B\}$;
  end if
end for
return P

3.1 以超边为中心的冗余消除执行方法

在 CPU-GPU 异构环境下, 频繁的数据移动往往成为 HyperGNN 模型训练的性能瓶颈. 为了有效减少数据传输开销, 我们提出了以超边为中心的冗余消除执行方法. 该方法的核心思想是充分挖掘和利用超边之间的拓扑重叠特性, 将具有高度拓扑相似性的超边划分到同一计算块中, 从而以最小化顶点特征向量传输实现最大化多条超边计算. 首先, 我们提出了一种新颖的超边拓扑相似度指标, 用于精准刻画超边之间的相似程度. 具体来说, 给定两个超边 h_i 和 h_j, 它们的**拓扑相似度** $TS(h_i, h_j)$ 定义为:

$$TS\left(h_i, h_j\right) = \frac{\left|N\left(h_i\right) \cap N\left(h_j\right)\right|}{\left|N\left(h_i\right) \cup N\left(h_j\right)\right|}$$

其中, $|N(h_i) \cup N(h_j)|$ 表示超边 h_i 和 h_j 的顶点并集的基数, $|N(h_i) \cap N(h_j)|$ 为超边 h_i 和 h_j 的共享顶点数量. $TS(h_i, h_j) \in [0, 1]$, $TS(h_i, h_j)$ 越接近 0 代表它们相似度越低, 越接近 1 代表它们相似度越高. 超边之间的拓扑相似度不仅考虑了超边之间共享顶点的数量, 我们还进一步通过对共享顶点度数 $\deg(v)$ 的加权求和 $\sum_{v \in N(h_i) \cap N(h_j)} \deg\left(v\right)$, 来定义每个超边块的优先级. 直观地说, 度数越大的共享顶点对相似度的贡献越大, 就意味着这些顶点共享了更多的超边, 通过优先传输和聚合优先级高的超边块, 不仅可以实现传输尽可能少的顶点特征实现最多的超边计算, 而且还可以最大限度减少跨块的数据传输与计算冗余, 从而大幅度减少 CPU-GPU 的数据通信量和冗余计算.

基于拓扑相似度 $TS(h_i, h_j)$, 我们设计了一种高效的基于动态阈值的超边块划分策略. 首先, 我们计算所有超边之间的拓扑相似度, 生成一个完备的拓扑相似度矩阵. 然后, 我们引入了一种动态阈值机制, 旨在自适应地对超边进行聚类. 具体而言, 对于每个超边对 h_i, h_j, 我们定义其动态阈值

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

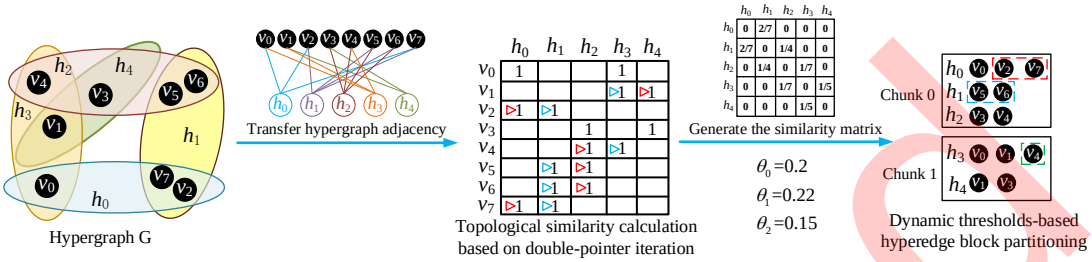

图 4　以超边为中心的冗余消除方法示意图

Figure 4　A simplified example for illustrating our approach

$\theta(h_i, h_j)$ 为:

$$\theta(h_i, h_j) = \theta_0 + \gamma \cdot \frac{1}{\sum\limits_{v \in N(h_i) \cap N(h_j)} \deg(v)} \tag{2}$$

其中, θ_0 为基础阈值, 用于控制划分的整体粒度; $\sum\limits_{v \in N(h_i) \cap N(h_j)} \deg(v)$ 为两条超边的共享顶点的度求和, 用来识别超边块内具有高度拓扑相似性的超边;γ 为相似度贡献系数, 控制了相似度对阈值的影响, 优先聚合高相似度的超边. 该动态阈值综合考虑了超边的拓扑特性和相似性, 能够自适应地调整划分粒度, 在减少冗余计算和保持块均衡之间取得平衡. 在实验中, 我们对数据集 WM 的 θ_0 和 γ 进行了敏感性分析 (4.3 节), 并分别采用了 0.45 和 0.3 作为默认值.

算法 1给出了基于动态阈值的超边块划分过程, 它通过遍历每条超边, 计算其与其他超边的相似度, 并将其相似度与动态阈值 $\theta(h_i, h_j)$ 进行比较. 如果 $TS(h_i, h_j) \geqslant \theta(h_i, h_j)$, 则将超边 h_i 和 h_j 归入同一个超边计算块. 重复这一过程, 直到所有超边都被划分为止. 该算法的时间复杂度为 $O(|H|^2 * V)$, 空间复杂度为 $O(|H|)$, 其中 $|H|$ 为超边数量, V 为顶点数量. 在实际应用中, 由于大多数超边的相似度远低于阈值而无需比较, 因此算法的实际运行时间远低于理论复杂度.

我们以图 4为例来阐述我们的方法, 首先对于超图我们将其表示为二分图, 然后基于二分图生成相似完备矩阵, 可以看出超边 $TS(h_0, h_1)$, $TS(h_1, h_2)$ 和 $TS(h_2, h_3)$ 和 $TS(h_3, h_4)$ 分别为 2/7, 1/4, 1/7 和 1/5, 并且这些边的动态阈值 θ_0, θ_1 和 θ_2 分别为 0.2, 0.22 和 0.15. 因此我们的方法将超边块划分为超边块 0 和超边 1, 其中超边块 0 包含 h_0, h_1 和 h_2, 超边块 1 包含 h_3 和 h_4. 这是因为 $0.15 < TS(h_2, h_3) < TS(h_3, h_4) < 0.22$, 从而将超边 h_3 和 h_4 划分到超边块 1 中. 通过在划分块内消除冗余计算, RHGNN 能够最小化 CPU 与 GPU 之间的数据传输, 并充分利用 GPU 的并行计算能力, 从而显著提升整个训练流程的效率. 此外, 超边拓扑相似度和自适应划分策略的引入, 使得 RHGNN 能够更精准地挖掘和利用超边之间的相似性, 在减少冗余的同时, 也降低了划分粒度带来的负面影响.

3.2　超度感知数据缓存策略

为了减少跨块访问时的通信开销, RHGNN 在 GPU 中实现了基于相邻超边块相似顶点特征的一级缓存机制. 在 CPU-GPU 异构环境中, 跨块数据传输会带来较大的通信开销, 因此通过分析相邻超边块之间的顶点数据相似性, 将超边块内的高相似度顶点特征优先存储在 GPU 的共享内存中. 具体来说, 首先在 CPU 上计算所有超边块之间的拓扑相似度矩阵, 识别出单个超边块内的所有共享顶点. 然后基于超边块内包含的全部顶点和共享顶点, 挖掘出共享顶点之外的高度顶点, 这些高度顶点

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

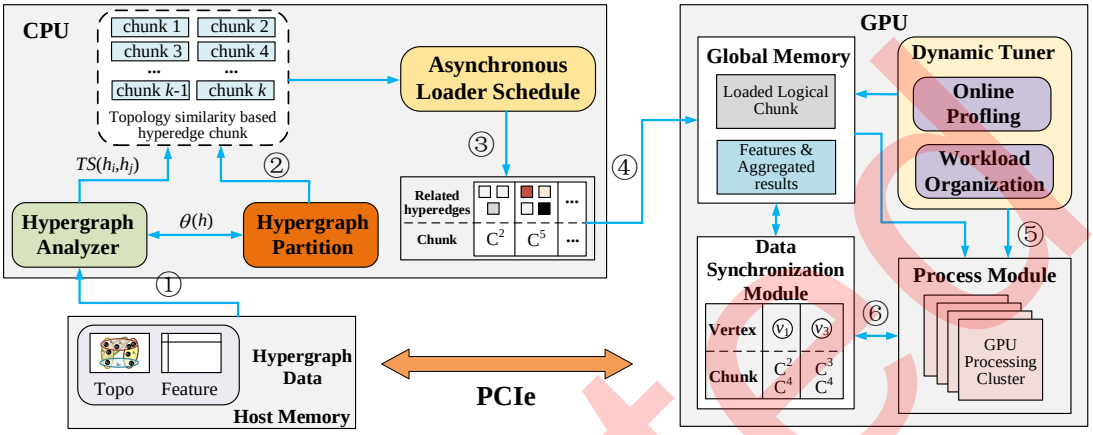

图 5 RHGNN 系统架构

Figure 5 RHGNN arcitecture

的特征会被标记为高优先级数据, 并传输到 GPU 的共享内存中, 供多个超边块复用. 这是因为高相似度顶点在多个超边块之间被频繁访问的概率最大, 并且由于共享内存的访问速度远高于全局内存, 多个 GPU 线程可以高效地共享这些顶点特征, 从而避免频繁的跨块数据传输和冗余计算.

为了优化对高超度顶点的访问, RHGNN 在 GPU 全局内存中实现了二级缓存机制, 将那些连接多个超边块、访问频率较高的顶点特征直接存储在 GPU 的全局内存中, 减少频繁的数据传输. 具体来说, 系统首先在 CPU 上统计所有顶点的超度, 即每个顶点所连接的超边块数量. 对于超度高的顶点, 其特征向量被直接存储在 GPU 全局内存中, 从而在 GPU 上执行的多个超边块可以快速访问这些顶点的特征, 而无需重复从 CPU 内存中加载. 此外, 系统动态监控这些高超度顶点的访问频率, 并根据计算负载的变化调整缓存策略, 确保 GPU 内存中保留的是访问频率最高的顶点特征. 这样做的效果是最大化高超度顶点的访问效率, 避免了 CPU-GPU 之间频繁的数据传输, 同时利用全局内存的较大容量存储更多的高频顶点特征.

4 RGHNN 系统设计与实现

为了在 CPU-GPU 异构架构上高效支持基于超边相似度的块划分和超度感知数据缓存策略, 本文设计了一种高效的系统架构 RHGNN, 如图 5 所示. 该架构主要包含超图分析器、超边划分模块、数据加载模块、运行时分析模块和数据同步模块, 各模块分别负责超图的预处理、任务划分、调度、运行时缓存管理以及数据同步.

4.1 系统架构

超图分析器. 如图 5 所示, CPU 端首先包含一个超图分析器模块, 该模块负责对超图结构进行分析. 具体而言, RHGNN 从 CPU 主存中加载超图拓扑结构和顶点特征向量数据 ①. 接下来, 超图分析器模块通过分析顶点和超边之间的关系, 计算每对超边的拓扑相似度 $TS(h_i, h_j)$, 生成相似度矩阵作为后续超边块划分的基础. 该模块能够有效捕捉超边之间的重叠与关联, 为块划分提供必要的相似性信息, 从而为后续计算做好准备.

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

超边划分模块. 在获得超边相似度矩阵后, 超边划分模块根据超边之间的相似度进行合理划分. 该模块基于动态阈值机制, 将相似度高于阈值的超边聚合到同一个计算块中 ②. 划分过程中, 系统会根据超边块的动态阈值自适应调整块的大小和结构, 确保冗余计算最小化. 此外, 系统在划分时还会综合考虑超边的复杂性与相似性, 进一步提升整体计算效率.

异步数据调度模块. 在超边划分完成后, 异步数据调度模块根据每个超边计算块的优先级进行调度. 调度策略依据超边块的计算量和相似度来决定优先级, 优先加载相似度较高且计算任务较重的超边块 ③, 从而确保这些块能够优先调度到 GPU 进行并行计算. 通过这一策略, 系统能够有效利用 GPU 资源, 最大化并行度, 减少计算等待时间, 从而提升整体性能.

运行时分析模块. GPU 端的运行时分析模块负责系统的动态优化. 该模块根据超边块的具体计算需求, 动态更新缓存, 并执行特征预加载操作 ④. 借助超度感知数据缓存策略, 运行时分析模块能够优先将高超度顶点的特征以及共享顶点的聚合值缓存在 GPU 内存中, 以此减少 CPU 和 GPU 之间频繁的数据通信开销. 接下来, 系统将这些超边块及其相关的特征数据送入 GPU 处理模块, 以完成并行超边计算 ⑤. 此外, 运行时分析模块还会根据任务执行过程中的计算核利用情况, 动态调整各计算核的工作负载, 确保资源得到充分利用, 从而提高整体计算效率.

数据同步模块. 数据同步模块负责在超边聚合计算完成后, 进行顶点聚合过程中的数据同步操作 ⑥. 由于每个顶点可能参与多个超边块的计算, 为确保在顶点聚合阶段能够快速访问所有相关的超边计算结果, 系统会在超边计算结束后, 记录每个顶点所关联的超边块 ID, 并将这些信息同步到 GPU 端. 在超边块计算结束时, 数据同步模块会整合顶点参与的超边块的计算结果, 并将这些数据同步到 GPU 内存中, 确保顶点聚合过程中能够完整、高效地获取所有相关特征. 通过对跨多个超边块的顶点进行关联处理, 数据同步模块有效避免了重复计算和数据延迟, 确保顶点特征能够快速更新, 提升整体系统的响应效率.

4.2 数据驱动的拓扑相似度计算策略

为了加快以超边为中心的冗余消除方法的执行效率, RGHNN 首先考虑采用双指针优化策略加速超边对的拓扑相似度计算. 具体来说, 我们注意到, 在计算两个超边 h_i 和 h_j 的相似度时, 最耗时的部分在于求解 $|h_i \cap h_j|$, 即两个超边所包含顶点集合的交集大小. 原始的计算方法是, 先遍历 h_i 中的每个顶点 v, 再在 h_j 中进行顺序查找, 检查 v 是否也属于 h_j. 很明显, 这种计算方式的时间复杂度为 $O(|h_i||h_j|)$, 导致整体训练效率低下.

为此, RGHNN 提出了一种基于双指针迭代的拓扑相似度计算优化策略. 首先, 它在预处理阶段, 对超图中每条超边所包含的顶点集合进行升序排序, 并将排序后的顶点序列存储为一个整型数组. 接下来, 我们使用两个指针 p_i 和 p_j, 分别指向 h_i 和 h_j 的顶点序列起始位置. 在每一轮迭代中, 我们比较 p_i 和 p_j 当前指向的顶点 v_i 和 v_j 的大小关系: (1) 如果 $v_i = v_j$, 则说明找到了一个公共顶点, 我们将交集计数器 cnt 加 1, 并同时将 p_i 和 p_j 向后移动一个位置; (2) 如果 $v_i < v_j$, 则说明 v_i 不可能在 h_j 中出现, 我们将 p_i 向后移动一个位置; (3) 如果 $v_i > v_j$, 则说明 v_j 不可能在 h_i 中出现, 我们将 p_j 向后移动一个位置. 以上过程一直重复, 直到 p_i 或 p_j 达到其对应顶点序列的末尾. 此时, 交集计数器 cnt 的值即为 $|h_i \cap h_j|$. 可以看出, 这种双指针迭代的时间复杂度为 $O(|h_i| + |h_j|)$, 远优于朴素的嵌套遍历. 需要特别指出的是, 超边内顶点序列的预排序虽然引入了额外的预处理开销, 但

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

这一开销可以在后续相似度计算中被充分地摊销. 此外, 由于不同超边所包含的顶点数量差异巨大, 为了进一步优化排序性能, 我们采用了内省式的排序算法选择策略: 对于顶点数较少 (小于 16) 的超边, 我们直接使用插入排序; 对于顶点数适中 (小于 128) 的超边, 我们使用希尔排序; 而对于顶点数很大的超边, 我们则使用快速排序. 通过这种分段式的算法选择, 我们在排序的平均时间复杂度与实际执行效率之间取得了有效平衡.

在双指针优化的基础之上, 我们进一步考虑对拓扑相似度计算过程进行细粒度的并行化. 传统的并行化实现通常采用边级并行的方式, 即每个线程负责计算一对超边之间的相似度. 然而, 我们发现这种粗粒度的并行方式存在两个主要问题: 一是并行粒度不够细腻, 难以充分利用多核 CPU 的计算资源; 二是不同线程之间竞争排序后的顶点序列数组, 导致频繁的缓存失效. 为了克服这些问题, RGHNN 提出了一种细粒度的指针级并行方法. 具体地, RGHNN 将每对超边的双指针迭代过程进一步划分为多个子任务, 每个子任务负责比较指针 p_i 和 p_j 在一个固定长度区间 (如 64) 内的顶点子序列. 在实际实现中, 它预先计算每个超边的顶点序列长度, 并据此得到需要划分的子任务总数 N, 然后使用 OpenMP 对这 N 个子任务进行并行化处理并保证了不同线程变量的原子性更新. 由于各个子任务所处理的顶点子序列在空间上是连续且不重叠的, 我们无需再担心缓存失效问题, 而细粒度的并行方式也充分发挥了工作量较大的超边对的并行优势.

在完成超边块的划分后, RHGNN 系统会根据划分结果, 生成一个超边块的优先级队列. 具有更多的共享顶点数量的超边块将被赋予更高的优先级, 从而尽早地被调度到 GPU 加速器上执行后续的特征聚合与更新计算. 在实际的调度过程中, 我们设计了一个异步流水线的执行模型: CPU 端的划分模块在持续地将超边块提交到优先级队列的同时, GPU 端的运行时分析模块也在不断地从队列中取出优先级最高的超边块, 并对其进行工作量估计与任务分解. 这种并行的流水线执行方式, 充分掩盖了 CPU 到 GPU 之间的通信延迟, 并实现了划分结果的实时流式传输, 从而避免了不必要的同步开销.

4.3 局部性感知的数据通信优化机制

当一个超边块被 GPU 端的运行时分析模块成功接收后, 该模块将立即对块内的所有顶点进行统计分析, 识别出其中的高相似性顶点 (即超边块内除了非共享顶点之外的高度顶点), 从而进行有效的数据缓存来减少 CPU-GPU 通信数据量. 为了快速识别出块内的高相似性顶点, 运行时分析模块采用了一种基于 GPU 的 Top-K 度数选择算法. 首先, 我们在 GPU 端为每个非共享顶点分配一个对应的线程, 并行计算它们的度数. 由于所有的共享顶点都已经在 CPU 端被识别出来, 因此 GPU 端的计算只需要关注非共享顶点的度数统计. 我们使用 CUDA 的原子操作 atomicAdd 来实现度数的并行累加. 具体地, 对于超边块内的每一条超边, 我们遍历其包含的所有顶点, 并将这些顶点的度数分别加 1. 在遍历过程中, 我们充分利用 CUDA 的 warp 机制, 确保同一线程束内的 32 个线程在访问同一条超边时可以实现协同操作, 避免了不必要的分支发散. 在完成度数计算后, 我们得到了一个存储了所有非共享顶点度数的数组.

接下来, 我们需要从中选出度数最大的 K 个顶点, 作为高相似性顶点集合. 这里的 K 值可以根据实际需求进行调节, 通常取顶点总数的 10%-20%. 具体来说, 我们首先在 GPU 端分配一个与度数数组大小相同的辅助数组, 用于存储每个顶点的初始索引. 然后, 我们对这两个数组同时进行

部分排序, 即每次只将当前未排序部分的前 K 个元素移动到数组的前端. 在排序过程中, 我们采用了 CUDA 内置的基于 Bitonic 排序网络的并行排序算法 *thrust* :: *sort_by_key*. 该算法可以在 $O(\log^2 n)$ 的时间内完成对 n 个元素的排序, 其中每一轮排序都可以通过并行的比较-交换操作高效实现. 通过适当调整排序网络的生成参数, 如比较器的布局和排序方向等, 我们可以进一步优化排序性能, 尽可能减少全局内存访问与线程束分化. 在完成初始的部分排序后, 我们获得了度数最大的 K 个顶点, 但它们在数组中的相对顺序可能还不满足要求. 因此, 我们需要对这 K 个顶点单独再进行一次排序. 这里我们直接采用 CUDA 的高性能并行排序原语 *thrust::sort*, 在几个微秒内即可完成对少量元素的排序. 至此, 我们就得到了超边块内度数最大的 K 个非共享顶点, 将它们的特征向量保存在 GPU 共享内存中以减少超边块间数据通信.

而对于高度顶点的识别, 需要从全局的角度进行考虑. 首先, RHGNN 对所有顶点的超度值进行一次全局排序, 并计算出前 0.5% 顶点的超度阈值. 接下来, 在每个线程块中, 我们利用 CUDA 的 curand 库生成一系列高质量的伪随机数, 并以此为索引, 从当前超边块的顶点集合中抽取固定数量的样本. 对于每个被采样到的顶点, 我们比较其超度值与预先计算出的全局阈值, 如果大于等于该阈值, 则将其标记为高度顶点, 并在共享内存中为其分配特征向量的缓存空间. 通过这种随机化的采样策略, 我们在保证较高精度的同时, 也显著降低了计算复杂度, 使得高度顶点识别的时间开销可以控制在毫秒量级. 在识别出高度顶点之后, 运行时分析模块将它们的特征向量拷贝到全局内存中, 并在必要时重排这些向量的布局, 以适应后续计算任务对访问模式的要求. 例如, 我们发现特征聚合的计算通常表现出规则的 SIMD 访问特征, 因此我们在缓存时, 会将连续多个顶点的同一维度特征存储到连续的全局内存地址, 从而实现向量化加载与存储. 相反地, 特征更新计算中则可能频繁地出现不连续的 Gather/Scatter 访问, 这就需要我们在缓存时保持特征向量原有的行优先布局, 以避免不必要的维度转置开销. 得益于 CUDA 精细的共享内存管理与同步机制, 这些数据布局优化可以显著提升缓存数据的重用效率, 并充分利用 GPU 的高带宽优势.

最后, 对于实际的特征聚合与更新计算, 我们采用了 CUDA 动态并行 (CDP) 技术, 以实现更高的设备利用率与执行效率. 传统的超边并行方法通常以" 一个线程块处理一条超边" 的方式来组织计算, 但这种粗粒度的并行粒度往往难以充分利用 GPU 设备, 尤其是当不同超边所包含的顶点数差异较大时, 会出现显著的负载不均衡现象. 而 CDP 则允许每个线程块在执行时, 进一步动态产生新的 CUDA 核函数, 用于处理当前超边中的一个子区间. 这实现了特征聚合与更新计算的" 两级并行", 有效平衡了负载, 提升了 GPU 的利用率. 在我们的实现中, 每个线程块首先以预定义的调度粒度 (如 128) 划分当前超边的顶点索引区间, 然后每个线程都独立地递归产生新的子核函数, 以处理对应的顶点子区间. 我们引入了一个基于 CUDA 原子操作的全局计数器, 用于协调线程块之间的区间划分行为, 确保不会出现重复或遗漏的计算.

需要特别指出的是, 为了在运行时适应不同超边块的调度粒度需求, 我们在分析模块中设计了一个自适应的负载均衡机制. 它基于当前超边块的顶点数量、特征维度、历史执行时间等统计信息, 动态评估特征聚合与更新计算的计算强度, 预测使用不同调度粒度时的预期加速比. 当计算强度较高且加速比超过特定阈值时, 并倾向于选择更小的调度粒度, 以实现细粒度的任务分解和动态负载均衡; 反之, 如果计算强度较低且加速比不明显, 则适当提高调度粒度, 减小 CDP 的递归深度和函数调用开销. 通过这种自适应的调节策略, RHGNN 可以根据实际的工作量特点, 在减小调度开销和提高设备利用率之间取得最佳平衡.

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

表 1　真实世界超图数据集

Table 1　The real-world hypergraph datasets

Datasets	Vertices	Edges	Features	label	Max hyperedge degree
Cooking200 (CK)	7, 403	2, 755	7, 403	20	25
CoauthorshipDBLP (DB)	41, 302	22, 363	1, 425	6	2, 838
CocitationPubmed (PB)	19, 717	7, 963	500	3	2, 241
YelpRestaurant (YP)	50, 758	679, 302	1, 862	11	173
WalmartTrips (WM)	88, 860	69, 906	88, 860	12	1, 808

4.4　价值感知数据同步机制

数据同步模块在超边聚合计算完成后, 负责进行顶点聚合过程中的数据同步操作. 由于每个顶点可能参与多个超边块的计算, 为确保在顶点聚合阶段能够快速访问所有相关的超边计算结果, 系统会在超边计算结束后, 记录每个顶点所关联的超边块 ID, 并将这些信息同步到 GPU 端. 在超边块计算结束时, 数据同步模块会整合顶点参与的超边块的计算结果, 并将这些数据同步到 GPU 内存中, 确保顶点聚合过程中能够完整、高效地获取所有相关特征. 通过对跨多个超边块的顶点进行关联处理, 数据同步模块有效避免了重复计算和数据延迟, 确保顶点特征能够快速更新, 提升整体系统的响应效率.

具体实现时, 当一个超边块完成其部分的顶点特征聚合计算后, RHGNN 立即启动一个异步的 CUDA 内核函数, 以记录该超边块内每个顶点的中间计算结果. 该内核函数将遍历超边块内的所有顶点, 并为每个顶点在全局内存中申请一个对应的索引项. 考虑到多个超边块可能同时完成计算, 因此这里需要使用 CUDA 的原子操作来保证对全局索引表的写入是原子性的, 避免出现竞争条件. 与此同时, 该内核函数还会记录当前超边块的 ID 到相应顶点的关联超边块列表中. 同样, 我们使用原子操作来更新这些列表, 保证多个线程可以安全地并发访问. 值得注意的是, 由于一个顶点最多只会参与到图中所有超边块的计算中, 因此关联超边块列表的最大长度是固定且可预知的. RHGNN 可以利用这一特性, 提前以颗粒度较大的方式为每个列表分配一段连续的全局内存, 避免了内存碎片化问题, 并实现了更高效的列表插入与查找性能. 当所有超边块都完成计算后, 每个顶点的索引项和关联超边块列表也都已准备就绪.

此时, 数据同步模块将启动另一个 CUDA 内核函数, 以遍历所有顶点并执行跨超边块的特征聚合. 对于每个目标顶点, 该函数首先读取其关联超边块的列表, 然后依次将这些超边块的中间计算结果从全局内存加载到当前内核线程的寄存器或共享内存中. 接下来, 内核函数对这些中间结果执行规约操作, 即逐元素地执行求和、取最大值等聚合操作, 最终得到该顶点的完整特征表示. 考虑到不同顶点的跨超边块聚合操作是相互独立的, 我们可以为每个顶点分配一个独立的 CUDA 线程块, 从而实现更高的并行度. 同时, 我们还可以利用 CUDA 的协作组 (Cooperative Groups) 机制, 将线程块内的线程划分为若干个更小的线程组, 每个线程组负责聚合一个超边块的中间结果. 这种分层的并行方式可以减少线程块内部的同步开销, 提升整体的计算效率. 最后, 当所有顶点的跨超边块聚合都完成后, 数据同步模块将把更新后的顶点特征从内核线程的局部内存写回到全局显存中, 以备后续下一层的超边聚合计算使用. 同时, 我们还会清空全局的顶点索引表和关联超边块列表, 为下一轮迭代做准备.

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

5 实验与结果

我们在 PyTorch [25] 框架的基础上使用 C++、CUDA C 和 Python 实现了 RHGNN. 而且, 我们的方法具有通用性, 并且独立于具体的深度学习或 GNN 框架.

5.1 实验配置

硬件设备. 实验使用的硬件平台为一个 65 核的 Intel Xeon 6151 CPU 处理器, 其主频为 3.0 GHz, 配备 696 GB DRAM 和一个 NVIDIA Tesla A100 GPU, 其具有 6, 912 个核心和 80 GB 高带宽. 我们将超边块范围最大设置为 128, GPU 启用了 CUDA 11.4 运行时和 418.67 驱动程序, 主机端运行的是 Ubuntu 18.04 操作系统, Linux 内核版本为 4.13.0. 所有源代码均使用 O3 优化进行了编译.

比较对象. RHGNN 的性能比较对象为 3 个主流 HyperGNN 系统, 即 PyG [24]、DGL [16] 和 HyperGef [12]. PyG 和 DGL 是目前最主流的 GNN 框架, 它们的版本分别为 2.5.3 [24] 和 2.3.0 [16]. HyperGef 是首个支持 HyperGNN 模型训练的 GPU 存内计算框架. 需要注意的是, PyG 和 DGL 是支持 CPU-GPU 异构模型训练的, 而 HyperGef 仅仅支持超图拓扑和特征向量数据都存储在 GPU 内才能训练模型. 因此, 为了与 HyperGef 进行公平的性能比较测试, 我们在 CPU 端采用常用的超图划分策略 [26~28] 对超图拓扑数据进行划分, 并分块传输到 GPU 内部进行训练. 对于每个 HyperGNN 模型, 我们训练了 200 个 epoch 并测试测量端到端的训练时间, 包括超图划分、CPU-GPU 数据通信、GPU 计算以及硬件利用率.

HyperGNN 模型和真实世界的超图数据集. 表 1 展示了 5 个真实世界的超图数据集, 即 (Cooking200 (CK) [3], CoauthorshipDBLP (DB) [4], CocitationPubmed (PB) [5], YelpRestaurant (YP) [6] 和 WalmartTrips (WM) [7]) , 这些数据集被广泛使用于 HyperGNN 模型应用研究中 [12, 22, 29]. 我们还选择了三个典型的 HyperGNN 模型, 即 Hypergraph Neural Network (HGNN) [14], UniGNN [30] 和 Hypergraph Convolution Network (HGCN) [22], 每一个模型的配置都为 3 层和隐藏层维度为 128.

5.2 整体性能分析

图 6 比较了 RHGNN 相较于其他系统在 5 个超图数据集下进行不同 HyperGNN 模型训练的标准化执行时间. 由图可以看出, RHGNN 相较于 PyG,DGL 和 HyperGef 在不同数据集和模型下分别实现了 2.5-3.4 倍, 2.1-3.1 倍和 1.4-2.3 倍的性能提升. 这性能提升的主要原因有三个方面: 首先, 基于超边相似度的动态划分策略使得 RHGNN 能够通过更合理的块划分减少冗余计算. 与传统的逐点或逐边索引不同, RHGNN 利用超边之间的相似度来有效地将高度相似的超边聚合在一起, 从而显

3) https://deephypergraph.readthedocs.io/en/latest/__modules/dhg/data/cooking_200.html

4) https://deephypergraph.readthedocs.io/en/latest/generated/dhg.data.CoauthorshipDBLP.html#dhg.data.CoauthorshipDBLP

5) https://deephypergraph.readthedocs.io/en/latest/generated/dhg.data.CocitationPubmed.html#dhg.data.CocitationPubmed

6) https://deephypergraph.readthedocs.io/en/latest/generated/dhg.data.CocitationPubmed.html#dhg.data.YelpRestaurant

7) https://deephypergraph.readthedocs.io/en/latest/generated/dhg.data.WalmartTrips.html#dhg.data.WalmartTrips

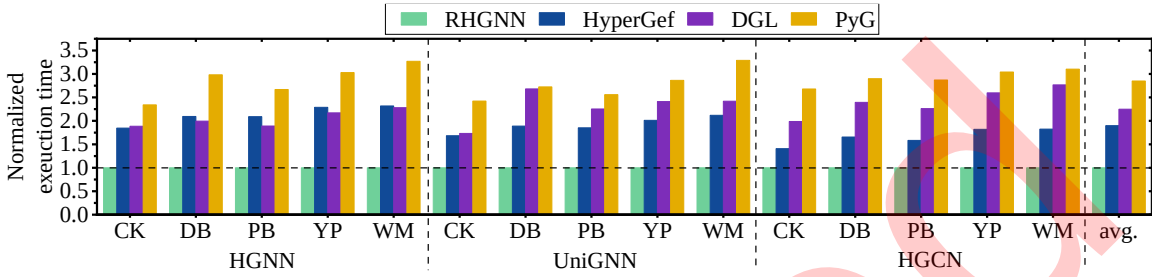

图 6 不同方案的标准化执行时间

Figure 6 The performance of different schemes normalized to that of RHGNN

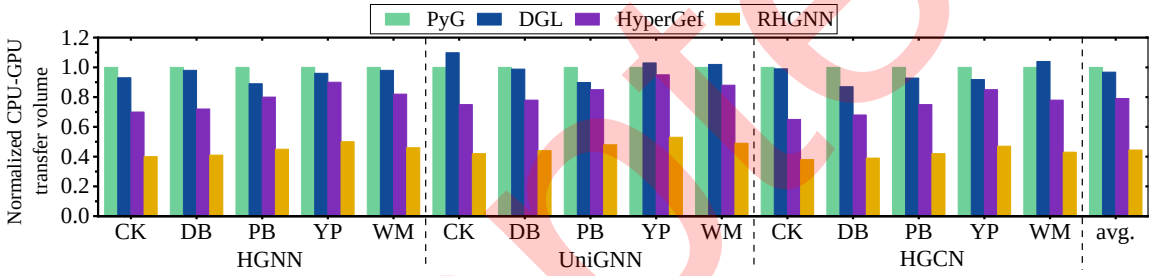

图 7 不同方案的标准化 CPU-GPU 通信量

Figure 7 The number of CPU-GPU transfer volumes of RHGNN against PyG, DGL, and HyperGef

著减少 CPU-GPU 之间的通信开销, 并通过分块的方式提升了 GPU 并行计算的效率. 其次, 超度感知的层次缓存机制极大地优化了 GPU 计算中的数据传输效率. 通过优先缓存高度顶点及高相似顶点的特征, RHGNN 减少了频繁的数据加载和特征传输. 这一机制保证了在多个超边块同时执行计算时, 系统能够快速访问关键顶点的特征数据, 避免了在多个计算任务之间重复加载相同的数据. 这种缓存机制有效降低了内存访问的延迟, 使得计算任务能够更加连贯地执行, 从而显著提升了整体计算效率.

5.3 数据通信量分析

图 7 比较了 RHGNN 相较于其他系统在 5 个超图数据集下进行不同 HyperGNN 模型训练的 CPU-GPU 数据通信量. 由图可以看出, RHGNN 相较于 PyG,DGL 和 HyperGef 在不同数据集和模型下减少了 47.2%-62.3% (平均 55.4%)、44.8%-67.8% (平均 48.3%) 和 27.1%-42.2% (平均 35.6%) 的 CPU-GPU 数据通信量. 这主要是因为 RHGNN 引入了超度感知的层次缓存机制, 通过优先缓存高频访问的顶点特征, 特别是高度顶点和高相似顶点的特征向量, 有效减少了 CPU 和 GPU 之间频繁的数据传输. 此外, RHGNN 的动态阈值超边划分策略通过将具有高相似度的超边块分配在一起, 减少了需要跨设备传输的顶点特征数据, 从而进一步降低了通信开销, 这些优化措施使得 RHGNN 能够显著减少 CPU-GPU 数据传输, 提升系统的整体效率.

为了进一步分析系统性能, 我们度量了不同系统在 5 种数据集上进行不同 HyperGNN 模型训练的 CPU 和 GPU 利用率. 在实验中, 我们每 100 毫秒记录一次 CPU 和 GPU 利用率, 并在十个周期内取平均值. RHGNN 通过在 CPU 和 GPU 之间智能调度训练任务, 并采用以超边为中心的冗余消除执行方法, 进一步减少了 CPU-GPU 的通信时间, 从而展现了更高的 CPU 和 GPU 资源利用

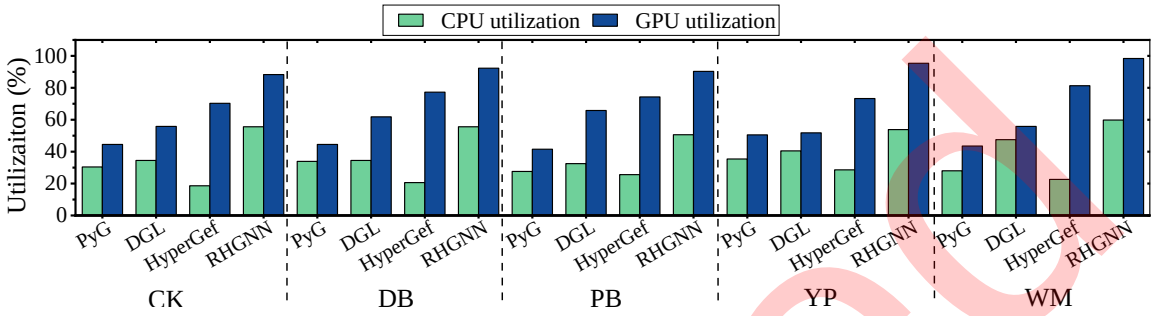

图 8　不同方案在不同数据集上的 CPU 和 GPU 利用率

Figure 8　The CPU and GPU utilization of different solutions over different datasets

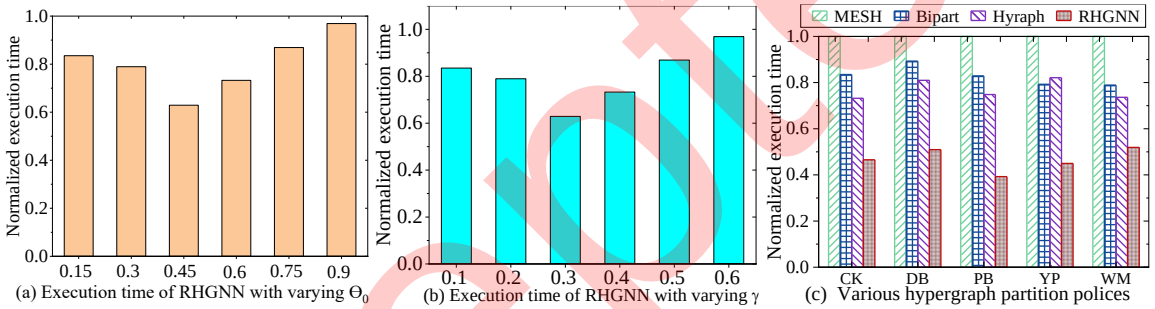

图 9　RHGNN 的敏感性分析: (a) RHGNN 在 WM 数据集上随着参数 θ_0 变化的执行时间; (b)RHGNN 在 WM 数据集上随着参数 γ 变化的执行时间; (c) RHGNN 在训练 HGNN 模型时使用不同超图分区策略的执行时间

Figure 9　Sensitivity studies of RHGNN: (a) sensitivity to θ_0 value; (b) sensitivity to γ value; (c) sensitivity to different hypergraph partition policies

率. 具体来说, RHGNN 的超边为中心的冗余消除方法有效减少了特征的重复计算和跨设备的数据移动, 使得计算能够更高效地在 GPU 上完成, 而无需频繁等待数据传输. 相比之下, DGL 和 PyG 的 CPU 利用率较高, 但 GPU 利用率较差. 这是因为它们在 CPU 上执行完整的超边特征收集步骤可以提高 CPU 的工作负载, 然而, 由于大量的特征数据需要从 CPU 传输到 GPU, 导致了较高的通信开销, GPU 核常常处于等待状态, 未能充分利用其计算能力. 另一方面, HyperGef 的 GPU 利用率相对较高, 因为它能够有效地在 GPU 上执行计算, 但 CPU 利用率较差. 这表明, 尽管 HyperGef 在拥有充足的 GPU 资源时表现出色, 但它在使用 CPU 进行预处理和数据传输方面的效率不高, 导致了系统性能的瓶颈. 相比之下, RHGNN 在 CPU 和 GPU 资源的利用率上达到了更好的平衡, 分别平均为 55.4% 和 93.1%. 这意味着 RHGNN 不仅能够充分利用 GPU 的强大计算能力, 还能够通过合理分配 CPU 的工作负载, 确保系统在不同硬件资源之间的高效协同, 显著提高了整体系统性能.

5.4　可扩展性分析

为了评估 RHGNN 的可扩展性, 我们对 RHGNN 的动态阈值中的 θ_0 和 γ 值以及 RHGNN 采用不同的超图划分策略进行了详细的分析.

图 9 (a) 描述了 RHGNN 在 WM 数据集上训练 HGNN 模型时, 随着 θ_0 变化的执行时间 ($\theta_0 \in [0.15, 0.9]$). 当 θ_0 越大时, 初始阈值过高会导致更多不相关的超边被划分到同一个计算块中, 块内计算负载增大, 进而增加计算时间. 此外, 由于超边间的相似度被过度聚合, 这样的划分方式会

减少块间并行度, 增加计算瓶颈. 因此, θ_0 过大会对性能产生负面影响. 相反, 当 θ_0 过小时, 划分粒度过细, 导致块间通信频繁, CPU-GPU 通信量显著增加. 我们的实验结果显示, $\theta_0 = 0.45$ 是一个能获得良好性能的最佳点, 在这一点上, 计算负载和通信开销实现了良好的平衡.

同样地, 为了能够让 RHGNN 自适应不同负载情况, 我们还对动态阈值的 γ 值进行了详细分析. 图 9 (b) 描述了 RHGNN 在 WM 数据集上训练 HGNN 模型时, 随着 γ 变化的执行时间 ($\gamma \in [0, 1]$). 当 γ 越大时, 动态调整阈值的幅度增大, 超边块划分的灵活性提高. 然而, 过大的 γ 值可能导致过于频繁的块划分调整, 增加计算开销和数据同步负担, 特别是在负载波动较大的超图中, 过大的 γ 值可能会引发不必要的重新划分. 因此, 合理选择 γ 值非常重要. 我们的实验结果显示,$\gamma = 0.3$ 是在 WM 数据集上实现最佳性能的点, 在此值下, RHGNN 能够平衡负载变化, 并减少不必要的超边块调整.

为了进一步展示 RHGNN 基于动态阈值的超边块划分策略的优越特性, 我们将 RHGNN 与 RHGNN 采用 MESH [26]、Bipart [31] 和 Hyraph [27] 的划分策略进行了比较. 从图 9 (c) 可以看出, RHGNN 的划分策略在执行时间上明显优于其他划分策略. 这是因为 RHGNN 的动态阈值超边块划分策略能够自适应地调整超边块的大小和相似度阈值, 确保相关性强的超边被聚合到同一块中, 减少了块内冗余计算和 CPU-GPU 间的数据传输量. 此外, 动态阈值策略还能灵活应对负载波动, 持较高的块内并行度, 最终大幅提升了系统的整体性能.

6 相关工作

基于 GPU 的图处理系统. 许多图迭代处理系统 [32∼37] 已被提出以加速传统的图算法在 GPU 上的运行. 其中一些研究工作还尝试探索不同的并行化策略, 包括顶点并行性、边并行性等. 此外, 也有一些努力通过领域特定语言 (DSL) [38∼41] 来探索动态并行化策略. 然而, 这些 GPU 图计算框架仅只能支持在普通图拓扑结构上进行迭代计算, 无法捕捉超图顶点之间的复杂关联关系. 为了在超图上支持超图迭代计算, 一些超图计算系统也相继被提出. HyperX [28] 和 MESH [26] 是分布式环境中第一代通用超图系统. HyperX 提供了 “像顶点或超边思考” 的模型, 用以统一表达各种超图算法. MESH 提出了从现有成熟的普通图系统快速构建超图系统的方案. 一个对称感知的超图划分机制被开发出来, 以改善分布式超图处理中的负载均衡 [42]. CHGL [43] 提供了高级操作符, 以提高编程效率. Hygra [27] 是第一个内存中的超图系统, 充分利用超图处理的运行时信息来显著提高性能. 然而, HyperGNN 与传统超图算法在图操作特性和特征嵌入维度方面有所不同, 传统的超图迭代计算系统高效支持 HyperGNN 模型训练.

GNN 优化技术. DGL [16] 和 PyG [24] 是两个目前最主流的 GNN 框架, 它们都使用基于 DNN 框架的消息传递编程接口来实现图相关操作. 为了使用 GPU 高并行能力, GNNAdvisor [7] 提出了基于 GPU 的运行时系统来有效支持 GNN 训练. 为了进一步支持大规模图神经网络模型训练, NeuGraph [18] 将数据流和顶点编程模式相结合来表示图神经网络的计算, 并在数据流中引入了以顶点为中心的消息传递模型. ROC [19] 使用在线线性回归模型来对图分区进行建模, 并进一步构建一个成本模型来预测计算图分区, 从而实现高效大规模 GNN 训练. 为了解决这个问题, HyperGef [12] 被开发出来以支持 HyperGNN 模型, 并利用高效的统一操作符融合. 然而, 这些软件框架仍然面临着冗余内存访问和计算的问题以及仅能支持小规模 HyperGNN 训练. Graph-CIM [44] 提出了一种新颖的基于 3D 堆叠存内计算的异构架构上的通用 GCN 任务分配策略, 相比现有方法能够充分减

少 GCN 的处理延迟和最小化数据移动开销. 为了进一步高效支持 GCN 推理, GCIM [45] 首次提出了一种新颖的基于 3D 堆叠的高效的软硬协同机制. Lift [46] 后续进一步提出了一种新颖的自适应混合 3D 堆叠存内计算架构和一种缓存友好的混合映射策略, 能够充分利用数据局部性和混合计算资源, 从而最大化 GCN 推理性能.

关于冗余消除的技术. HAG [47] 通过识别出公共邻居, 并创建聚合顶点来表示中间聚合结果, 以管理和重用中间聚合结果. I-GCN [17] 提出了一种在线图重构算法, 而 ReGNN [48] 提出了一个动态冗余消除算法, 以减少 GNN 推理中的冗余计算. 尽管这些设计理念表面上看似与 RHGNN 相似, 但它们无法减少 HyperGNN 模型训练中的冗余计算和昂贵的数据移动开销. 这是因为它们在使用现有的 GNN 方法支持超图时, 对超图顶点和普通顶点执行相同的 GNN 聚合语义, 以实现顶点聚合和超边聚合, 而未考虑超图的特性.

7 结论

本论文提出了 RHGNN, 一个面向 CPU-GPU 异构环境的高性能超图神经网络系统, 旨在解决大规模超图神经网络训练中的冗余计算和频繁数据通信问题. 通过利用超图中超边的拓扑重叠特性, RHGNN 实现了一种新颖的块中心训练方法, 优化了顶点特征向量的加载和通信, 使得一次加载可以服务于多个超边的计算. 此外, 系统还引入了基于超度感知的层次缓存机制, 优先将频繁访问的顶点特征向量和中间结果存储在 GPU 端, 从而进一步减少了通信开销. 我们的实验评估表明, RHGNN 相较于现有的最先进 HyperGNN 系统实现了 1.4-3.4 倍的性能提升.

An efficient high throughput system of hypergraph neural network under CPU-GPU heterogeneous environments

Hui Yu[1, 2, 3, 4], Yu Zhang[1, 2, 3, 4*], Xintao Li[1, 2, 3, 4], Zikang Chen[1, 2, 3, 4], Yingqi Zhao[1, 2, 3, 4], Jin Zhao[1, 2, 3, 4], Hao Qi[1, 2, 3, 4], Xiaofei Liao[1, 2, 3, 4] & Hai Jin[1, 2, 3, 4]

1. *National Engineering Research Center for Big Data Technology and System, Huazhong University of Science and Technology, Wuhan* 430074, *China*;
2. *Service Computing Technology and System Lab, Huazhong University of Science and Technology, Wuhan* 430074, *China*;
3. *Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan* 430074, *China*;
4. *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074, *China*

* Corresponding author. E-mail: zhyu@hust.edu.cn

https://www.sciengine.com/doi/10.1360/SSI-2024-0282

Abstract In recent years, *Graph Neural Networks* (GNNs) have gained significant attention for their exceptional ability to learn and reason with non-Euclidean data. However, real-world graph structures often involve complex higher-order relationships between vertices, typically represented as hypergraphs. To effectively capture these intricate features and underlying semantic information in hypergraphs, numerous *Hypergraph Neural Network* (HGNN) models have been proposed. Despite the development of various software systems that leverage the high parallel computing power of GPUs to accelerate HGNN training, these systems still face challenges such as *excessive redundant computation* and *frequent data communication*, leading to suboptimal GPU utilization. Moreover, these systems are limited to training HGNN models on relatively small-scale graphs. To address these issues, we observed that multiple hyperedge computations often repeatedly access and compute the identical vertex feature vectors due to significant topological overlap between hyperedges in hypergraphs when handling the HyperGNN models. Based on this observation, we propose *RHGNN*, a redundancy-elimination HyperGNN training system tailored for CPU-GPU heterogeneous environments. RHGNN exploits the overlapping characteristics of hyperedges to optimize the computation and updating of hyperedge and vertex features, significantly reducing CPU-GPU data communication overhead. Specifically, RHGNN presents a novel hyperedge-centric redundancy elimination approach, enabling the system to load the feature vectors of vertices to optimize communication, allowing a single load operation to serve multiple hyperedge and vertex feature computations. In addition, RHGNN implements an efficient hyperdegree-aware hierarchy caching mechanism that prioritizes frequently accessed vertex feature vectors and intermediate results on the GPU, further reducing communication costs. To validate the effectiveness of RHGNN, we compared its performance against the state-of-the-art HGNN software systems DGL, PyG, and HyperGef. Experimental results demonstrate that RHGNN achieves a performance improvement of 2.5-3.4x over PyG, 2.1-3.1x over DGL, and 1.4-2.3x over HyperGef in HGNN model training, respectively.

Keywords hypergraph graph neural network, training optimization, CPU-GPU heterogeneous environment, redundancy elimination, topology similarity

参考文献

1 Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, pages 1–14, 2017.

2 Hulin Dai, Xuan Peng, Xuanhua Shi, Ligang He, Qian Xiong, and Hai Jin. Reveal training performance mystery between tensorflow and pytorch in the single GPU environment. *Science China Information Sciences*, 65(1):1–17, 2022.

3 Hui Yu, Yu Zhang, Ligang He, Donghao He, Qikun Li, Jin Zhao, Xiaofei Liao, Hai Jin, Lin Gu, and Haikun Liu. Cda-gnn: A chain-driven accelerator for efficient asynchronous graph neural network. In *Proceedings of the 61th ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

4 Adam Auten, Matthew Tomei, and Rakesh Kumar. Hardware acceleration of graph neural networks. In *Proceedings of the 57th ACM/IEEE Design Automation Conference*, pages 1–6, 2020.

5 Hui Yu, Yu Zhang, Andong Tan, Chenze Lu, Jin Zhao, Xiaofei Liao, Hai Jin, and Haikun Liu. Rtga: A redundancy-free accelerator for high-performance temporal graph neural network infere. In *Proceedings of the 61th ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

6 Dang Yang, Haikun Liu, Hai Jin, and Yu Zhang. Hmvisor: dynamic hybrid memory management for virtual machines. *Science China Information Sciences*, 64(9):1–16, 2021.

7 Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on gpus. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation*, pages 515–531, 2021.

8 Hui Yu andYu Zhang, Ligang He, Yingqi Zhao, Xintao Li, Ruida Xin, Xiaofei Liao, Hai Jin, Bingsheng He, and Haikun Liu. Rahp: A redundancy-aware accelerator for high-performance hypergraph neural network. In *Proceedings of the 57th ACM/IEEE international symposium on microarchitecture*, pages 1–14, 2024.

9 Hui Yu, Yu Zhang, Jin Zhao, Yujian Liao, Zhiying Huang, Donghao He, Lin Gu, Hai Jin, Xiaofei Liao, Haikun Liu, Bingsheng He, and Jianhui Yue. RACE: an efficient redundancy-aware accelerator for dynamic graph neural network. *ACM Transactions on Architecture and Code Optimization*, 20(4):53:1–53:26, 2023.

10 Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. HyGCN: A GCN accelerator with hybrid architecture. In *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture*, pages 15–29, 2020.

11 Yue Gao, Zizhao Zhang, Haojie Lin, Xibin Zhao, Shaoyi Du, and Changqing Zou. Hypergraph learning: Methods and practices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(5):2548–2566, 2022.

12 Zhongming Yu, Guohao Dai, Shang Yang, Genghan Zhang, Hengrui Zhang, Feiwen Zhu, June Yang, Jishen Zhao, and Yu Wang. HyperGef: A framework enabling efficient fusion for hypergraph neural network on gpus. In *Proceedings of the 6th Conference on Machine Learning and Systems*, pages 1–13, 2023.

13 Devanshu Arya, Deepak K. Gupta, Stevan Rudinac, and Marcel Worring. HyperSAGE: Generalizing inductive representation learning on hypergraphs. *CoRR*, abs/2010.04558:1–14, 2020.

14 Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the 2019 AAAI Conference on Artificial Intelligence*, pages 3558–3565, 2019.

15 Haoyu Chen, Deqing Zou, Hai Jin, Shouhuai Xu, and Bin Yuan. SAND: semi-automated adaptive network defense via programmable rule generation and deployment. *Science China Information Sciences*, 65(7):1–18, 2022.

16 Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv, abs/1909.01315*, pages 1–18, 2019.

17 Tong Geng, Chunshu Wu, Yongan Zhang, Cheng Tan, Chenhao Xie, Haoran You, Martin C. Herbordt, Yingyan Lin, and Ang Li. I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1051–1063, 2021.

18 Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. Neugraph: Parallel deep neural network computation on large graphs. In *Proceedings of the 2019 USENIX Annual Technical Conference*, pages 443–458, 2019.

19 Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. In *Proceedings of the Third Conference on Machine Learning and Systems*, 2020.

20 Zhiping Wang, Haofei Yin, and Xin Jiang. Exploring the dynamic growth mechanism of social networks using evolutionary hypergraph. *Physica A: Statistical Mechanics and its Applications*, 544:1–15, 2020.

21 Chen Avin, Zvi Lotker, Yinon Nahum, and David Peleg. Random preferential attachment hypergraph. In *Proceedings of the 2019 International Conference on Advances in Social Networks Analysis and Mining*, pages 398–405, 2019.

22 Yangding Li, Yingying Wan, and Xingyi Liu. Semi-supervised learning with graph convolutional networks based on hypergraph. *Neural Processing Letters*, 54(4):2629–2644, 2022.

23 Yue Gao, Yifan Feng, Shuyi Ji, and Rongrong Ji. HGNN+: General hypergraph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3181–3199, 2022.

24 Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019.

25 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

26 Benjamin Heintz, Rankyung Hong, Shivangi Singh, Gaurav Khandelwal, Corey Tesdahl, and Abhishek Chandra. MESH: A flexible distributed hypergraph processing system. In *Proceedings of the 2019 IEEE International Conference on Cloud Engineering*, pages 12–22, 2019.

27 Julian Shun. Practical parallel hypergraph algorithms. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 232–249, 2020.

28 Wenkai Jiang, Jianzhong Qi, Jeffrey Xu Yu, Jin Huang, and Rui Zhang. Hyperx: A scalable hypergraph framework. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):909–922, 2019.

29 Cheng Wang, Nan Ma, Zhixuan Wu, Jin Zhang, and Yongqiang Yao. Survey of hypergraph neural networks and its application to action recognition. In *Proceedings of the 2nd Artificial Intelligence International Conference*, pages 387–398, 2022.

30 Jing Huang and Jie Yang. UniGNN: A unified framework for graph and hypergraph neural networks. In *Proceedings of the 13rd International Joint Conference on Artificial Intelligence*, pages 2563–2569, 2021.

31 Sepideh Maleki, Udit Agarwal, Martin Burtscher, and Keshav Pingali. Bipart: a parallel and deterministic hypergraph partitioner. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*,

pages 161–174, 2021.

32 Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. PowerGraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, pages 17–30, 2012.

33 Zuhair Khayyat, Karim Awara, Amani AlOnazi, Hani Jamjoom, Dan Williams, and Panos Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proceedings of the Eighth European Conference on Computer Systems*, pages 169–182, 2013.

34 Farzad Khorasani, Keval Vora, Rajiv Gupta, and Laxmi N. Bhuyan. Cusha: vertex-centric graph processing on gpus. In *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, pages 239–252, 2014.

35 Jin Zhao, Yu Zhang, Xiaofei Liao, Ligang He, Bingsheng He, Hai Jin, and Haikun Liu. LCCG: A locality-centric hardware accelerator for high throughput of concurrent graph processing. In *Proceedings of the 2021 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 45:1–45:14, 2021.

36 Yu Zhang, Xiaofei Liao, Hai Jin, Ligang He, Bingsheng He, Haikun Liu, and Lin Gu. DepGraph: A dependency-driven accelerator for efficient iterative graph processing. In *Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture*, pages 371–384, 2021.

37 Hui Yu, Xinyu Jiang, Jin Zhao, Hao Qi, Yu Zhang, Xiaofei Liao, Haikun Liu, Fubing Mao, and Hai Jin. Toward high-performance delta-based iterative processing with a group-based approach. *Journal of Computer Science and Technology*, 37(4):797–813, 2022.

38 Ajay Brahmakshatriya, Yunming Zhang, Changwan Hong, Shoaib Kamil, Julian Shun, and Saman P. Amarasinghe. Compiling graph applications for GPU s with graphit. In *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization*, pages 248–261, 2021.

39 Mario Luca Bernardi, Marta Cimitile, and Giuseppe A. Di Lucca. Design pattern detection using a dsl-driven graph matching approach. *Journal of Software: Evolution and Process*, 26(12):1233–1266, 2014.

40 Sungpack Hong, Hassan Chafi, Eric Sedlar, and Kunle Olukotun. Green-marl: a DSL for easy and efficient graph analysis. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 349–362, 2012.

41 Yunming Zhang, Ajay Brahmakshatriya, Xinyi Chen, Laxman Dhulipala, Shoaib Kamil, Saman P. Amarasinghe, and Julian Shun. Optimizing ordered graph algorithms with graphit. In *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, pages 158–170, 2020.

42 Yu Gu, Kaiqiang Yu, Zhen Song, Jianzhong Qi, Zhigang Wang, Ge Yu, and Rui Zhang. Distributed hypergraph processing using intersection graphs. *IEEE Transactions on Knowledge and Data Engineering*, 34(7):3182–3195, 2022.

43 Louis Jenkins, Tanveer Hossain Bhuiyan, Sarah Harun, Christopher Lightsey, David Mentgen, Sinan G. Aksoy, Timothy Stavcnger, Marcin Zalewski, Hugh R. Medal, and Cliff A. Joslyn. Chapel hypergraph library (CHGL). In *Proceedings of the 2018 IEEE High Performance Extreme Computing Conference*, pages 1–6, 2018.

44 Jiaxian Chen, Guanquan Lin, Jiexin Chen, and Yi Wang. Towards efficient allocation of graph convolutional networks on hybrid computation-in-memory architecture. *Science China Information Sciences*, 64(6):1–14, 2021.

45 Jiaxian Chen, Yiquan Lin, Kaoyi Sun, Jiexin Chen, Chenlin Ma, Rui Mao, and Yi Wang. GCIM: toward efficient processing of graph convolutional networks in 3d-stacked memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):3579–3590, 2022.

46 Jiaxian Chen, Zhaoyu Zhong, Kaoyi Sun, Chenlin Ma, Rui Mao, and Yi Wang. Lift: Exploiting hybrid stacked memory for energy-efficient processing of graph convolutional networks. In *Proceedings of the 60th ACM/IEEE Design Automation Conference*, pages 1–6, 2023.

47 Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. Redundancy-free computation for graph neural networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 997–1005, 2020.

48 Cen Chen, Kenli Li, Yangfan Li, and Xiaofeng Zou. ReGNN: A redundancy-eliminated graph neural networks accelerator. In *Proceedings of the 28th IEEE International Symposium on High-Performance Computer Architecture*, pages 1–14, 2022.