图卷积神经网络加速器 GCIM 的复现及改进

钟朝宇

摘要

现阶段，图卷积神经网络（Graph Convolutional Networks，GCNs）已经成为一种强大的处理图数据的深度学习方法。与卷积神经网络（Convolutional Neural Networks，CNNs）等传统神经网络不同，GCNs 可以处理不规则的图数据。然而，GCNs 具有计算受限及访存受限的特性。因此，如何高效利用底层计算资源和内存资源成为现阶段的一个研究热点。新兴的三维堆叠存内计算架构）可以减少计算逻辑单元和内存之间的数据移动，从而为上述问题提供了一个很好的解决方案。一个尚未解决的关键问题在于如何分配 GCNs 以充分利用三维堆叠存内计算架构的快速近内存数据处理的特性。

在这项工作中,我们分析了 GCN 的混合执行模式,并复现了基于三维堆叠存内计算架构的 GCIM[1]，在复现过程，我们发现在原有的工作上，图的分割、分配以及图卷积的策略造成了额外的数据流动开销，于是我们在原有基础上对硬件与图的分配策略进行了改进，与原来工作对比减少了数据流动和能耗。

关键词：近存计算；图卷积神经网络

1 引言

GNNs 是一类处理图域信息的深度学习方法，由于其较好的性能和可解释性，GNNs 成为一种广泛应用的图处理方法。GCNs 是 GNNs 中常用且非常强大的用于处理图数据的神经网络架构之一，在许多如推荐系统和社交网络中有许多应用。

GCN 的执行阶段主要可以分为聚合阶段和组合阶段。在聚合阶段中，图的顶点会访问其所有邻居顶点并进行其特征向量累加或求平均等操作；在组合阶段中，图的处理类似于神经网络，也即矩阵乘法处理，需要较大的计算能力。这两个阶段的混合执行对当下传统计算架构造成巨大压力，原有工作针对图神经网络的执行模式设计了新的计算架构 GCIM，提升了图卷积神经网络的计算速度并减小了能耗。我们在原有工作基础上，对 GCIM 进行了重构，改进了硬件设计以及图的分配策略。相较与 GCIM，改进的工作实现了功耗降低。

2 动机

系统能耗大多由数据移动引起。在 3D-CIM 体系结构的图卷积神经网络加速器中，数据移动问题主要来自两个方面: 未优化的映射和数据流。对于映射，聚合操作被映射到 CIM 单元，可以运用到可观的 DRAM 带宽。在聚合阶段，具有高度出入度的顶点将从不同的库或不同的通道收集它们的所有邻居顶点，这将导致大量的数据移动。对于数据流，采用基于拉取的聚合策略，以利用大容量的 DRAM。由于图拓扑的不确定性，每个顶点的特征向量将被动态访问。跨库特征的地址需要由内容寻址存储器 (CAM) 记录。因此，CAM 有限的容量会导致跨库特性的频繁抓取操作。我们认为混合映射和基于推送的策略可以有效地解决数据移动问题。首先，通过将高出入度的顶点转移到基础芯片 (base die) 上，

可以显著减少跨通道数据移动。其次，采用基于推送的聚合策略，每个顶点的特征向量移动到一个库一次。这需要进一步的体系结构和系统软件支持。这些特点促使我们提出了一种硬件和软件协同设计方法，以开发新兴的 3D CIM 体系结构。

3 相关工作

3.1 用于 GCN 的 ASIC/FPGA 加速器

许多专门的硬件架构被提出来加速 GCNs 的处理。这些硬件架构可以分为两种类型: 混合架构和统一架构。混合架构利用 gcn 的混合模式分别加速内存绑定聚合阶段和计算绑定组合阶段。与混合体系结构不同，统一体系结构将 GCNs 抽象为链稀疏稠密矩阵乘法，以提高硬件利用率。虽然上述工作可以有效地提高 GCNs 的推理性能，但仍然面临着巨大的 DRAM 访问和功耗的问题。

3.2 存内计算架构

作为解决内存墙挑战的有前途的解决方案，CIM 体系结构将计算放置在内存附近，以减少大量数据移动开销。CIM 架构的内存技术主要有 NVM (非易失性存储器) 和 DRAM。尽管这些 CIM 体系结构可以有效地改进特定应用程序的处理，但它们不是为 gcn 设计的。ZHU[2]的研究中提出了基于 3d 堆叠存储器的广义稀疏矩阵-矩阵乘法加速器 (SpGEMM)。与 SpGEMM 不同，GCNs 的聚合操作类似于稀疏矩阵和密集矩阵的乘法。因此，GCNs 既表现出其不规则模性，但也有规则可循。GCIM 利用 3d 堆叠 CIM 体系结构的近内存库带宽来加速混合模式 gcn 的处理并设计了新颖的 CIM 架构，将近库计算单元集成到 3d 堆叠内存中，充分利用了巨大的带宽。这种硬件和软件的协同设计对 GCNs 的处理更有利于访存和减小能耗。

4 本文方法

4.1 本文方法概述

改进版 GCIM 目标是尽量减少数据移动并降低总体能源消耗。在硬件层面，针对聚合阶段访存的随机性提出了一个轻量级处理单元 (BPU)，为每个 CIM 内存库组提供基于图的推送策略的数据流。针对一些特殊的顶点，将放在基础芯片上设计的辅助处理单元 (SPU)。

在软件层面，我们采用混合映射策略来利用潜在的数据局部性，充分利用珍贵的计算资源。将出入度高的顶点映射在 SPU 中，而将剩余顶点映射在 BPU 中。对于这些出入度高的顶点，这样的策略减小了跨通道数据转移的消耗。在初次分配完结点之后，通常 SPU 与 BPU 的计算负载会出现过剩或缺口，于是我们在初次分配的基础上使用的线性规划重新转移某些结点来平衡 SPU 与 BPU 之间的计算负载差距。

4.2 图的推送策略数据流

图 1表示了图的推送策略的数据处理方法，如图所示，顶点将其特征向量广播给它的邻居节点，这也意味着在图的聚合阶段中顶点对应的向量仅需被加载一次。但在累加聚合特征时，用以累加的矩阵会被动态访问。

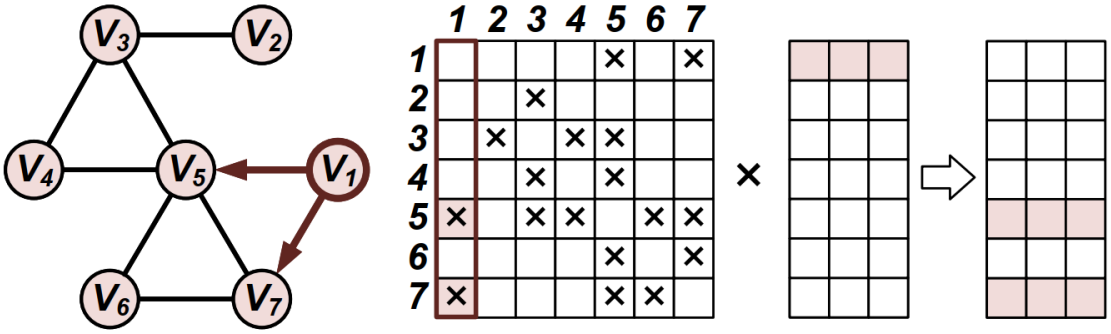

图 1: 图推送策略的数据流

4.3 Bank Processing Unit(BPU) 硬件设计

如图 2所示，BPU 由一个输入向量缓冲区、一个 look-ahead FIFO、一个控制单元和用于矩阵操作的乘法累积 (MAC) 阵列组成。look-ahead FIFO 是一种定制的临时存储器，用于缓冲稀疏矩阵和隐藏延迟。输入向量缓冲区用于缓冲输入向量，包括用于组合阶段 MLP 的权重矩阵的行向量或是与权重矩阵相乘后的特征向量矩阵的行向量。

4.4 Supplementary Processing Unit(SPU) 硬件设计

改进版 GCIM 在基础芯片上引入了 SPU 用以处理出入度高的顶点，度数高的顶点往往比其他顶点的数量少的多，在 GCN 每层的推理过程中这些顶点放入 SPU 中相较于放入 BPU 的顶点能减少其邻居顶点数量次的跨通道访问。如图 3所示，SPU 主要由 MAC 阵列和专用缓冲区组成，基于上述原因，在 SPU 专用缓存区的设计中，少数专用缓冲区被用来临时缓存稀疏矩阵和输入向量而大多数输出缓冲区为输出向量设置。

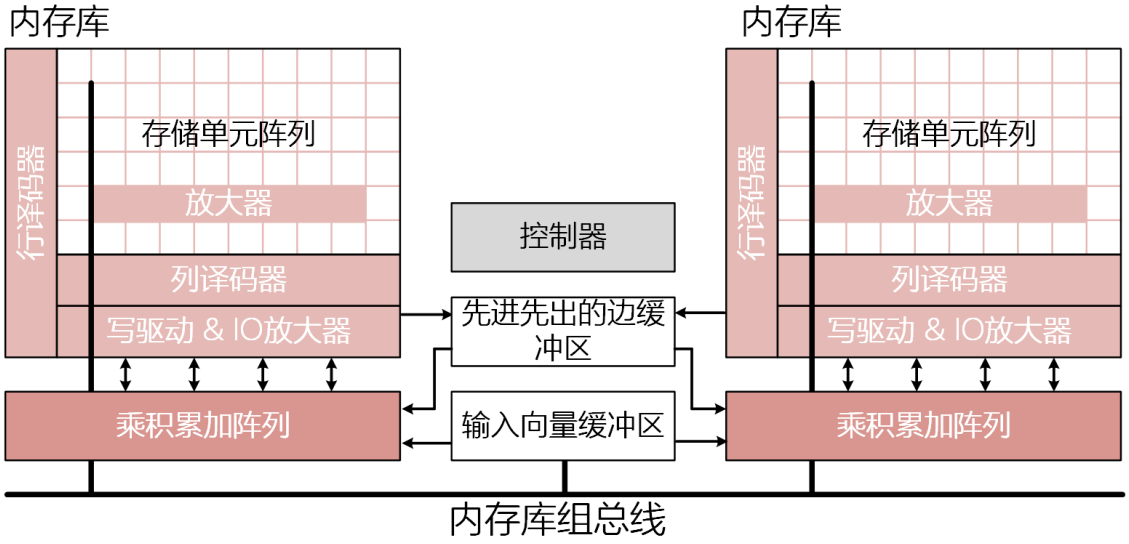

图 2: BPU 硬件设计

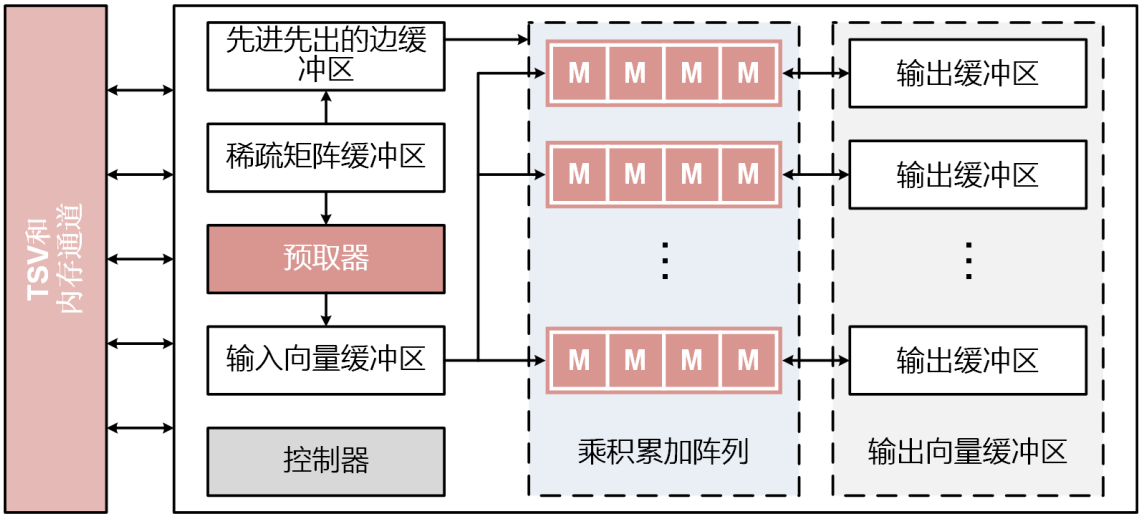

图 3: SPU 硬件设计

5 复现细节

5.1 与已有开源代码对比

在 GCIM 原有代码上进行了改进，包括硬件设计的重构，以及根据图顶点特点设计了映射算法。在聚合阶段的图处理方法上，采用了推送方法代替了原有的拉取方法从而减少了数据流动。此外还分析了图的拓扑特点，将出入高的顶点卸载到基础芯片上进一步减少数据流动。

5.2 复现概述

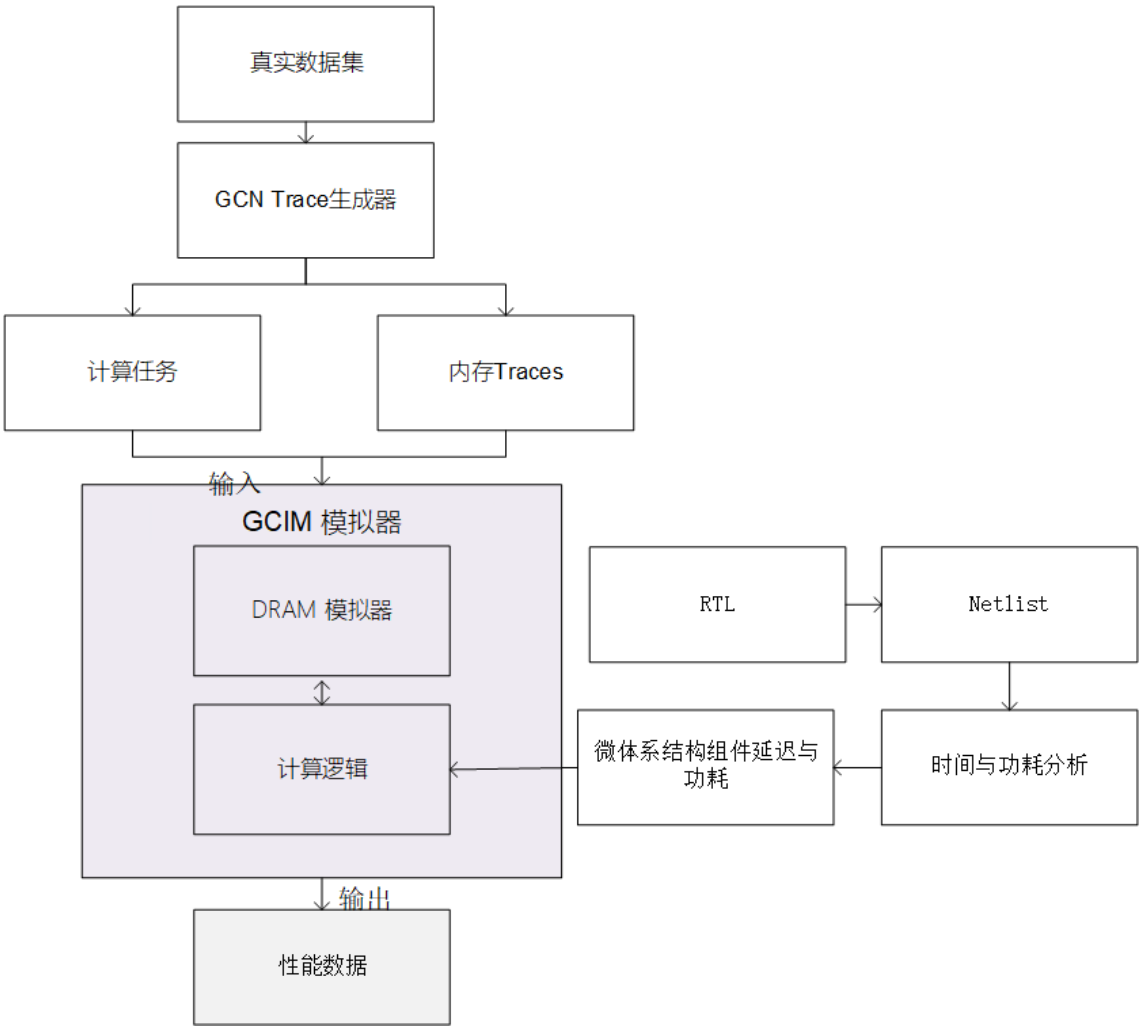

图 4: GCIM 复现示意图

图 4为 GCIM 复现示意图。这里使用了真实数据集作为输入，并通过工作负载生成器生成了对应计算任务和访存请求作为 GCIM 模拟器的输入，来生成最终性能结果输出。

为了模拟内存请求的响应延迟，使用了 DRAMSim3[3]模拟器来对内存访问行为进行建模。对于 GCIM 的各个组件的功耗以及延迟等参数，我们对硬件组件进行了综合和建模。各组件的能耗和延迟等参数也是 GCIM 模拟器的输入，用于得到最终结果。

具体实现：

数据集:fpim//gcn_dataset//

数据结果:fpim//res//

主程序:fpim//fpimtest//test.cpp

5.3 图顶点映射算法

设 BPU 和 SPU 的计算能力分别为 cap_B 和 cap_S; 设输入图的边和顶点的个数，分别为 num_e 和 num_v。然后，得到顶点度阈值 thr_d，其中度大于 thr_d 的顶点的总边数刚好超过:

$$exp_S = \frac{cap_A}{cap_B + cap_A} \cdot num_E \tag{1}$$

度大于 thr_d 的顶点将被映射到 SPU，而度小于或等于 thr_d 的顶点将被映射到 BPU。为了充分利用 CIM bank 组的并行性，改进版 GCIM 定义了分配给每个 BPU 的预期边数:

$$exp_B = \frac{cap_B \cdot num_E}{(cap_B + cap_S) \cdot num_B} \tag{2}$$

其中 num_B 代表 BPU 的数量为了在不破坏数据局域性的情况下映射 BPU 之间的顶点，对输入图应用了有界深度优先搜索 (BDFS)。算法 1 演示了 BPU 的初始映射的生成。改进版 GCIM 将以深度优先的方式遍历每个顶点。度小于或等于 thr_d 的顶点将被映射到当前的 BPU, $BPU[id]$(第 10-11 行)。一旦分配给当前 BPU 的边数超过 exp_B，剩下的顶点将开始分配给下一个 BPU(第 5-6 行)。基于了有界深度优先搜索遍历完成后，所有的顶点将被映射到不同的 BPU 上。映射算法伪代码如下。

Procedure 1 在 BPU 上建立初始顶点映射

Input: 每个 BPU 期望分配的边数 exp_B, BDFS 搜索的深度 $init\_depth$
Output: BPU 的映射结果 $BPU[\,]$

$id \leftarrow 0$
for $u \in V$ **do**
  if $visit[u] \neq true$ **then**
    BDFS$(u, init_depth, id)$ **if** $BPU[id].edge \geqslant exp_B$ **then**
      $id \leftarrow id + 1$
    end
  end
end

function BDFS$(u, depth, id)$
if $depth > 0$ *and* $BPU[id].edge < exp_B$ **then**
  $visit[u] \leftarrow true$ **if** $u.degree \leqslant thr_d$ **then**
    Assign$(BPU[id], u)$
  end
  for $v \in$ *Neighbor*(u) **do**
    if $visit[v] \neq true$ **then**
      BDFS$(v, depth - 1, id)$
    end
  end
end

5.4 基于线性规划的再分配

初始映射只考虑处理单元的计算能力，而不考虑内存访问的延迟。这样的映射将导致不平衡的工作负载，并增加整体处理延迟和能量消耗。因此，改进版 GCIM 应用重新分配来进一步最小化处理延迟。为了计算延迟，我们引入了简化版的性能模型如下：

$$\begin{cases} SPU.t = T_S(SPU.mem, SPU.cmp) \\ BPU[i].t = T_B(BPU[i].mem, BPU[i].cmp) \end{cases} \tag{3}$$

其中 $SPU.mem$ 和 $SPU.cmp$ 分别为 SPU 的内存访问量和计算，函数 $T_A()$ 使用 SPU 的内存访问时延和计算时延之和来估计 SPU 的处理时延 $SPU.t$。同样，$BPU[i].t$ 为第 i 个 BPU 的估计处理时延。为了避免内存访问导致的不平衡，改进版 GCIM 使用哈希函数 (或基于轮循的方法) 在非 cim 的内存库组之间均匀地映射特征矩阵和其他数据。因此，改进版 GCIM 采用平均延迟来估计每次内存访问。

改进版 GCIM 使用线性规划重新分配分配给每个处理单元的边和顶点以最小化延迟。我们用 $x_i (i \leqslant num_B)$ 和 x_S 来决定 BPU 和 SPU 之间需要重新分配多少条边，其中 $\sum_i x_i + x_S = 0$。代价函数 $C_S()$ 和 $C_i()$ 分别用于估计 BPU 和 SPU 每边重新分配导致的平均延迟变化。为了表述这个问题，我们使用延迟 Len 表示所有处理单元之间的最大延迟。那么，这个问题可以通过下面的线性规划有效地解决:

$$\begin{aligned} \text{minimize} \quad & Len \\ \text{subject to} \quad & \textstyle\sum_i x_i + x_S = 0 \\ & SPU.t + C_S(x_S) < Len \\ & \forall i : BPU[i].t + C_i(x_i) < Len \end{aligned} \tag{4}$$

对于 $x_i < 0$ 的第 i 个 BPU，需要卸载几个度最高的顶点到 SPU，直到卸载了 x_i 条边。类似地，x_i 大于 0 的第 i 个 BPU 将从 SPU 中重新分配顶点。

6 实验结果分析

6.1 实验设置

测试数据集实验测试中采用了五个常用的图数据集。使用了一套代表性 GCN 应用，包括 GCN[4], GraphSage[5]和 GINConv(GIN)[6], 每个 GCN 应用有两个图卷积层，隐藏神经元大小为 128。

实验基准我们将改进版 GCIM 与 GCIM 进行比较。改进版 GCIM 和 GCIM 同是 3D CIM 体系结构，GCIM 对 GCN 的图处理采用了基于拉取策略的数据流。改进版 GCIM 在 BPU 配备了 1KB 容量的 Look-ahead FIFO，2KB 容量的专用输入向量缓冲，16 个乘积累加单元。在 SPU 中配备了 8KB 的稀疏矩阵缓冲，16KB 的输入向量缓冲，512KB 的输出向量缓冲以及 256 个乘积累加单元。

6.2 实验结果

加速比图 5显示了 GCIM 和改进版 GCIM 之间的加速比，以 GCIM 为基准的情况下，得到了改进版 GCIM 的加速比。改进版 GCIM 在这些数据集上实现了平均加速比为 4.73 的效果。在 GS 数据集上改进版 GCIM 的加速比表现不如原版 GCIM 这是因为，GS 数据集中没有出入度高的顶点，也就没有充分利用到基础芯片中 SPU 以减小出入度高的顶点的数据流动。

能耗图 6显示了 GCIM 和改进版 GCIM 之间的处理能耗比较，这里使用改进版 GCIM 作为基准，发现改进版 GCIM 在能耗方面平均节约了 6.05 倍，由于改进版 GCIM 采用了推送的图处理策略以及合适的图顶点映射策略，大大减少了数据移动从而减少了能耗。

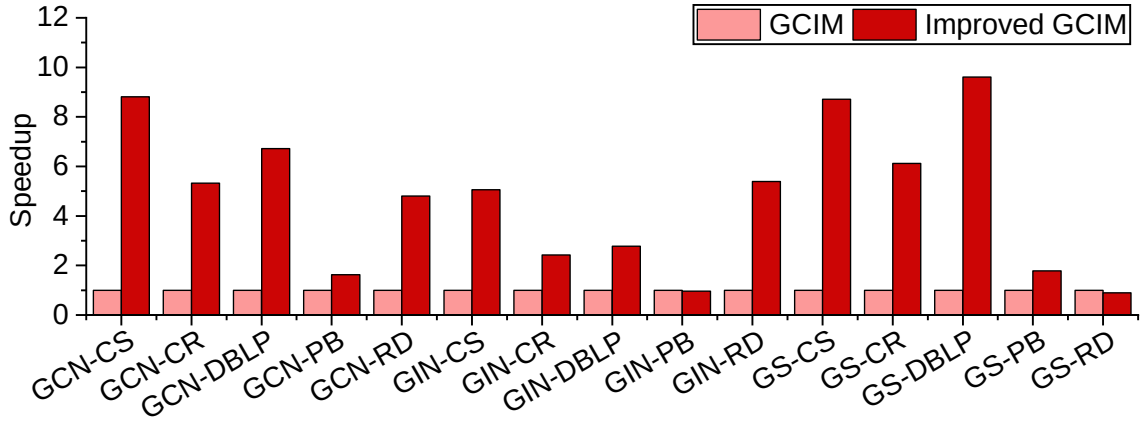

图 5: 加速比

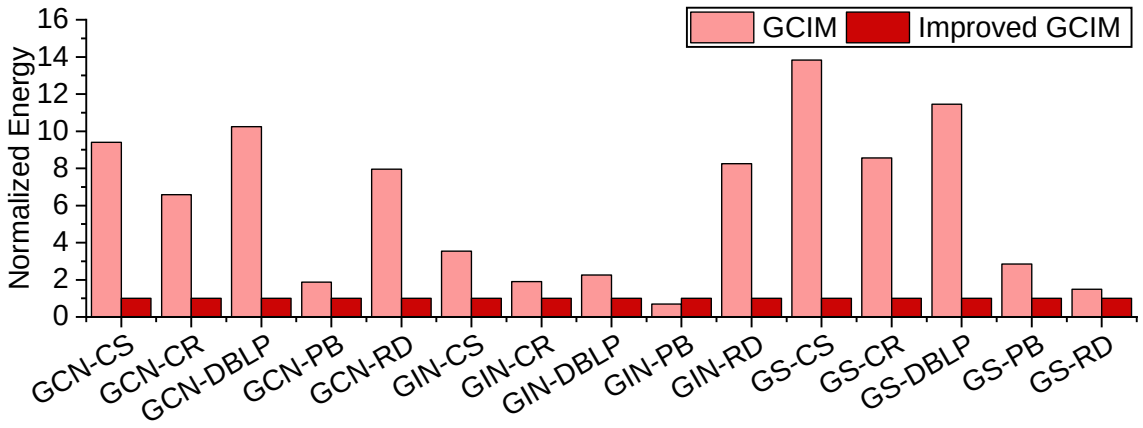

图 6: 能耗

7 总结与展望

本文在复现 GCIM 的基础上对其进行了改进，根据聚合阶段图的不同处理方式以及对卸载图高出入度顶点的方式，对硬件和映射算法进行了更新，达到了普遍优于原版 GCIM 的加速以及能耗表现。未来，我们将评估更多的性能指标，从图的特点探索更优处理方式，达成更好的性能效果。

参考文献

[1] CHEN J, LIN Y, SUN K, et al. GCIM: Toward Efficient Processing of Graph Convolutional Networks in 3D-Stacked Memory[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022: 3579-3590.

[2] ZHU Q, AKIN B, SUMBUL H E, et al. A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing[C]//IEEE International 3D Systems Integration Conference (3DIC). 2013: 1-7.

[3] LI S, YANG Z, REDDY D, et al. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator [J]. IEEE Computer Architecture Letters, 2020: 106-109.

[4] KIPF T N, WELLING M. Semi-Supervised Classification with Graph Convolutional Networks[C]// International Conference on Learning Representations (ICLR). 2017: 1-14.

[5] HAMILTON W, YING Z, LESKOVEC J. Inductive Representation Learning on Large Graphs[C]// Advances in Neural Information Processing Systems (NIPS). 2017: 1-11.

[6] XU K, HU W, LESKOVEC J, et al. How Powerful are Graph Neural Networks?[C]//International Conference on Learning Representations (ICLR). 2019: 1-17.