TensorCIM: Digital Computing-in-Memory Tensor Processor With Multichip-Module-Based Architecture for Beyond-NN Acceleration

Yiqi Wang, *Graduate Student Member, IEEE*, Zihan Wu, Weiwei Wu, Leibo Liu, *Senior Member, IEEE*, Yang Hu, *Member, IEEE*, Shaojun Wei, *Fellow, IEEE*, Fengbin Tu, *Member, IEEE*, and Shouyi Yin, *Senior Member, IEEE*

***Abstract*— While neural networks (NNs) have achieved great results in various intelligent tasks like image classification and speech recognition, real-world scenarios have more applications beyond just NN processing like graph convolutional network (GCN) and deep-learning recommendation model (DLRM), which typically consist of sparse gathering (SpG) and sparse algebra (SpA). Their large application size leads to substantial data movement. Although the fusion of digital computing-in-memory (CIM) and multichip-module (MCM) can reduce data movement efficiently and scale out CIM's capacity in a high-yield solution, the MCM-CIM system raises new challenges for beyond-NN acceleration: SpG involves repeated off-chip DRAM access, interchiplet access, and redundant reduction operations; SpA suffers from inter-CIM workload imbalance and intra-CIM under-utilization. Thus, we design TensorCIM as the CIM processor chiplet with three corresponding features: 1) the redundancy-eliminated gathering manager (REGM) dynamically maintains frequently accessed features and reduction results in the CIM to eliminate redundant accesses and reductions; 2) the equal operation-based CIM initializer (EOCI) calculates effective multiply-accumulation (MAC) operations and initializes CIM macros with a balanced inter-CIM workload at the subarray level; and 3) the input-lookahead CIM (ILA-CIM) architecture looks ahead at future inputs to fully utilize CIM logic. The fabricated MCM-CIM system consumes only 3.7 nJ/Gather for the GCN model, achieving 8.3-TFLOPS/W algebra efficiency at FP32.**

***Index Terms*— Computing-in-memory (CIM), floating-point, graph neural network (NN), multichip-module (MCM), recommendation system, sparsity.**

Manuscript received 17 January 2024; revised 8 April 2024; accepted 24 May 2024. Date of publication 13 June 2024; date of current version 30 January 2025. This article was approved by Associate Editor Priyanka Raina. This work was supported in part by the National Key Research and Development Program under Grant 2018YFB2202600, in part by the NSFC under Grant U19B2041 and Grant 61774094, in part by the Beijing Science and Technology (S&T) Project under Grant Z191100007519016, and in part by the Beijing Innovation Center for Future Chip. *(Corresponding authors: Fengbin Tu; Shouyi Yin.)*

Yiqi Wang, Zihan Wu, Weiwei Wu, Leibo Liu, Yang Hu, Shaojun Wei, and Shouyi Yin are with the School of Integrated Circuits, Beijing Innovation Center for Future Chip, and Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: yinsy@tsinghua.edu.cn).

Fengbin Tu is with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong, China (e-mail: fengbintu@ust.hk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2024.3406569.

Digital Object Identifier 10.1109/JSSC.2024.3406569

I. Introduction

WHILE neural networks (NNs) have achieved great results in various intelligent tasks [10], real-world scenarios have more applications that have computational and data-movement requirements beyond those seen in typical NN processing, like graph convolutional network (GCN) [8] for graph-based learning and deep-learning recommendation model (DLRM) [14] for recommendation (see Fig. 1). GCN aggregates features on a large graph and then performs an NN-based combination. DLRM gathers and reduces items from an embedding table and generates recommendation results via fully connected layers. However, prior artificial intelligence (AI) chip research focuses on NN-only applications, and we demand a new architecture for beyond-NN acceleration.

To realize efficient beyond-NN acceleration, we first analyze the workload features of beyond-NN applications. Beyond-NN applications typically consist of sparse gathering (SpG) and sparse algebra (SpA). SpG gathers and reduces tensors from sparsely distributed addresses, like GCN's aggregation phase and DLRM's embedding layer. SpA refers to NN-based sparse tensor multiplication for the gathered tensors (GTs) like GCN's combination phase and DLRM's fully-connected (FC) layer. Besides, due to the large application size, data movement is the main bottleneck for beyond-NN acceleration, and FP32 precision is usually required for high accuracy.

Our solution for beyond-NN applications is the fusion of digital computing-in-memory (CIM) and multichip-module (MCM), which reduces data movement while maintaining high precision and accommodates large application sizes, respectively. In the past few years, digital CIM has proven to be an efficient and precise architecture for reducing data movement [4], [5], [13], [16], [19], [20]. The large-scale beyond-NN applications bring huge computational and storage requirements and motivate the demand for scaling out digital CIM processors. MCM provides a high-yield scalable solution for CIM scaling by integrating multiple chiplets in one package to build an MCM-CIM system [15], [28], [29].

Fig. 2 shows a typical MCM-CIM system with four CIM chiplets. In our MCM-CIM system, workloads are partitioned to each chiplet. The CIM chiplet mainly relies on its own DRAM channel to perform SpG and SpA. The MCM architecture enlarges CIM capacity. Intermediate data are

GCN: Graphic Convolutional Network

通过聚合邻居节点的特征，更新每个节点的表示，提取局部信息

flow: ① aggregation 聚合信息 ②. combination 与自身信息结合 + 非线性变换

$H^{(l+1)} = \sigma(\hat{A} \cdot H^{(l)} W^{(l)})$

特点：稀疏，局部性强，图结构(不规则内存访问)，精度要求

DLRM Deep Learning Recommendation Model

用于个性化推荐，融合稀疏特性(用户点击历史，商品 ID)…
稠密特征(年龄、价格)…

flow: ① Embedding: 嵌入层，稀疏特征映射为低维向量

② Interaction: 特征交互，Embedding vector ⊗? 稠密特征 → 组合特征

③ MLP: 全连接网络：输入多层感知机 预测用户行为

特点：Embedding 表巨大，gather + reduce 频繁，稀疏计算 精度

sparsity Gather: GCN 邻居聚合，DLRM: 嵌入向量查找

sparsity Algebra: 稀疏张量的乘加运算

spG: 频繁访问 DRAM 和 chiplet，带宽压力大
相同特征的重复处理

spA: 计算负载不均
CIM 空闲 → 利用率低

Tensor CIM: Redundancy-Eliminated-Gathering Manager

spG: 建立 dynamic cache.
动态维护常访问特征与中间结果于 CIM 内。
避免重复读取/计算

Equal-Operation-CIM Initializer:

spA: 基于 GT 和 WT 的非零项元素计算有效 MAC
并均衡分配到不同的 CIM 核子阵列

CIM 有效 MAC 差距过大

Input-Look-Ahead CIM.

spA 计算时，ILA 预读输入，并行激活子阵列。
↓
每次输入只有部分阵列激活

sparsity. FP32 accuracy. spG spA access-unordered.

data movement

↳ digital CIM + MCM

MAC inside of SRAM + multi chiplet package as system

CIM. accuracy

Connection, independent DRAM channel, storage + expandation.

3 challege and resolution

A: 对于稀疏的数据 需先 spG + spA MAC+ accuracy
gather + reduce.

B: CIM: effective, high accu. expandation.
在 SRAM 单元中加入 2输入 NOR 门 → 按位乘.
阵列周边 加数字加法器. partial sum.

C. MCM: 一个芯片存储能力不够 需要多个组合.

D. spG spA 问题

3条中数据流: spG: CIM gathering
spA init.: CIM Initializer
spA Compute: CIM Computation.

flow: ① spG model.
load tensor from DRAM.
REGM: select frequently-ass. storage into CIM
CIM Core implement feature gathering. → Gathered Tensor.
GT → global buffer.

② spA Init.
EOCI load weight Tensor from DRAM, analysis sparsity arch.
distribute WT to CIM core's SRAM
read GT from Input Buffer.

③ spA Compute:
ILA-CIM 预读 激活子阵列
FP32 MAC.
result store → Global buffer
SIMD: 激活.

datapath复用. spG spA 优化.

Tensor CIM architecture
主要结构.
① ILA-CIM Core (x8).
负责主计算单元
② Redundancy Eliminated Gathering Manager (x8)
③ Equal Operation-CIM Initializer
CIM初始化器
④ Input buffer ⑤ Global buffer
⑥ SIMD Core ⑦ Top Controller.

Redundancy Eliminated Gathering Manager

mechanism

spG: Beyond-NN 第一步. 根据不规则的访问张量 Access Tensor

从 feature tensor 中提取相关特征.

进行归约. 生成 Gathered Tensor.

在 GCN 中: spG: AT: 图的邻接矩阵

DLRM: AT: 用户历史交互

多数情况

访问模式高度集中且重复: 大量节点访问同一节点. /局部性

↳ 缓存能命中. 带宽是瓶颈

优化策略: ① Access Tensor reorder. 让相似访问模式的操作连续处理

② 将 spG 工作分配到 不同的 chiplet. 每个 chiplet 访问局部共享 DRAM

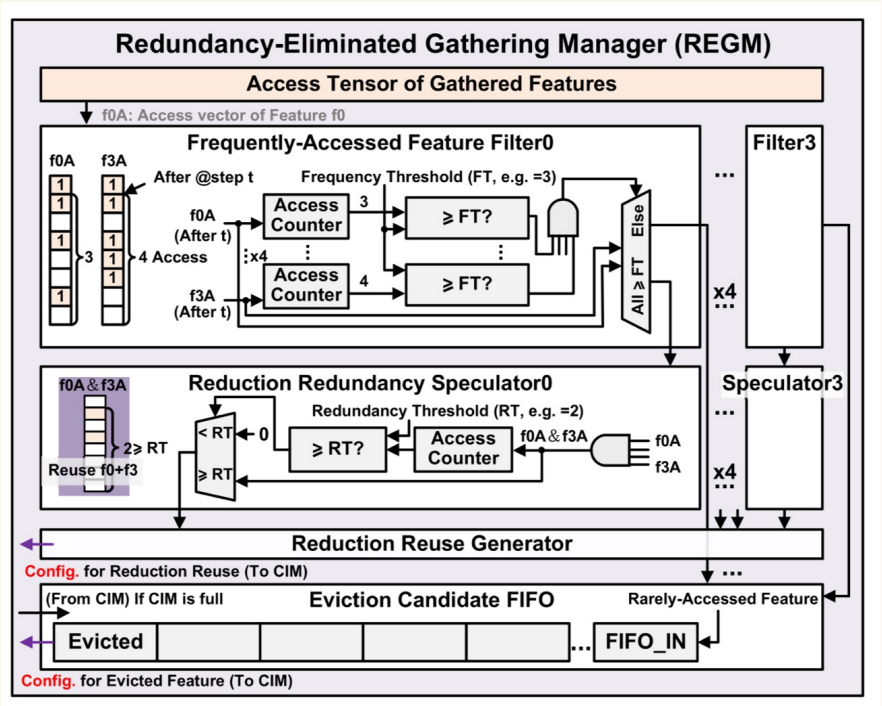
③ REGM 冗余去除:

记录访问频率高的特征及其归约结果

CIM 中缓存 以重用

避免重复访问 重复计算. 减少 DRAM 带宽和 chiplet 间的通信.

architecture:



① ② 获取 AT 信息.

统计每个特征的未来访问次数.

与设定的阈值作比较.

↙ CIM ↘ ECF

③ 对于 f0+f3 使用它存储的是

获取未来可能的访问次数.

↙ CIM ↘ x

生成 configuration 指导 CIM 写入

① Frequently-Accessed Feature Filter (x4)

判断哪些是高频访问 AT. 并存储在 CIM 中.

② Reduction Redundancy Speculator (x4)

判断哪些归约结果值得存入缓存

生成存储与重用控制信号 指导 CIM 存取行为

③ Reduction Reuse Generator ④ Eviction Candidate FIFO

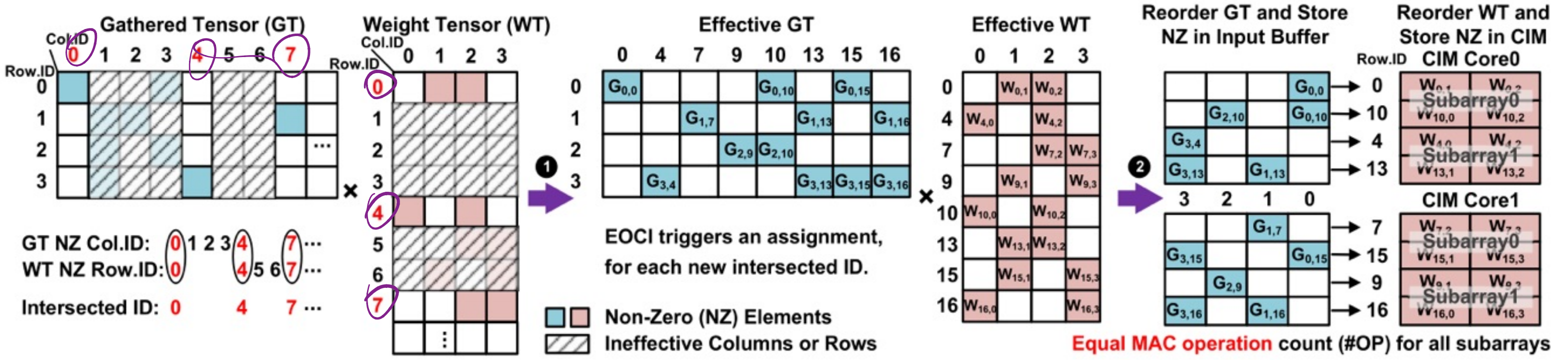


Fig. 12. Equal operation-based CIM initialization example: ❶ effective OP recognition and ❷ balanced WT/GT assignment.

Equal Operation CIM Initializer:

spA中不同的CIM工作负担不同（稀疏性）

↳在CIM初始化，通过智能分配 Weight Tensor 和 Gathered Tensor 实现负载平衡

How: ① 获取非0索引，求交集（有效计算只在两者交集）

② 对于有效的ID，统计对应行列的计算数 ...

③ 选择合适的CIM子阵列映射，平衡工作负载

④ 适配CIM的权重切片 WT Tiling

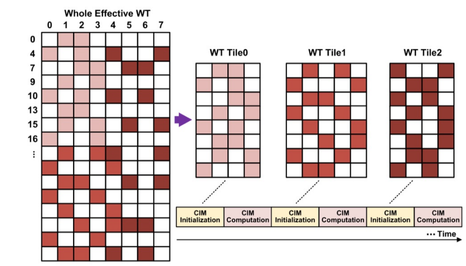


Fig. 13. CIM-adaptive WT mapping. The whole effective WT is divided into multiple small WT tiles and mapped onto CIM during each CIM initialization.

→找出有效计算 → 形成有效计算的WT GT → 均衡分配计算 → 划分数据 → CIM计算

architecture:

Effective Operation Recognizer

检测GT和WT的非零ID交集

On-Demand Weight Fetcher

从DRAM中加载真正被使用的WT

Balanced Inter-CIM Distributor

管理各个CIM子阵列负载/工作分配

How:

从 global buffer 读取GT

↓ E.ID

GT FIFO

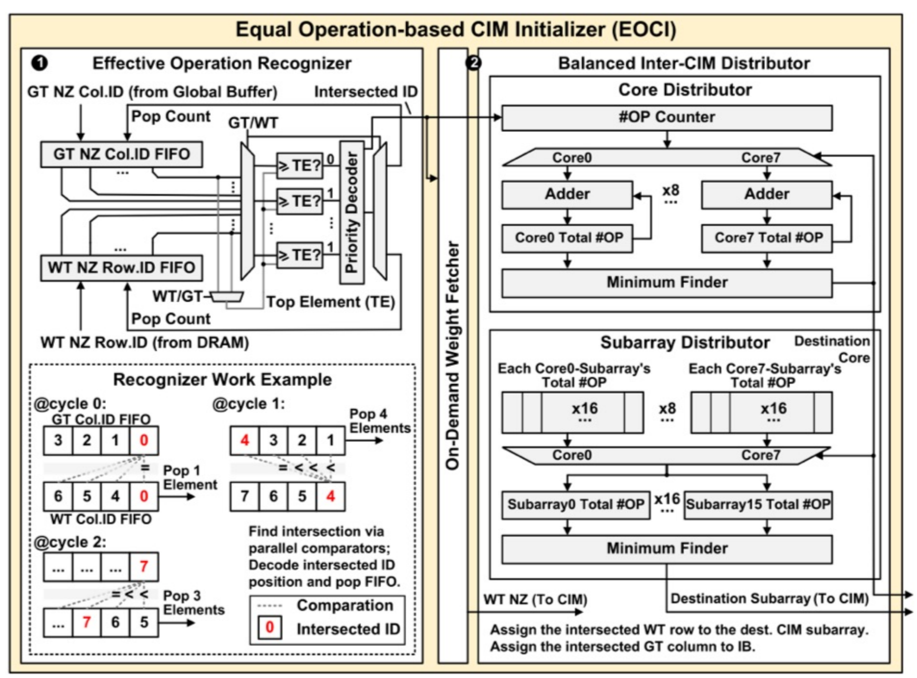


Fig. 14. EOCI architecture: ❶ effective OP recognition and ❷ balanced WT/GT assignment.

从DRAM中读取 WT

↓ E.ID

WT FIFO

↓ ↓ 并行比较 → 交集

WT FIFO – 的有GT

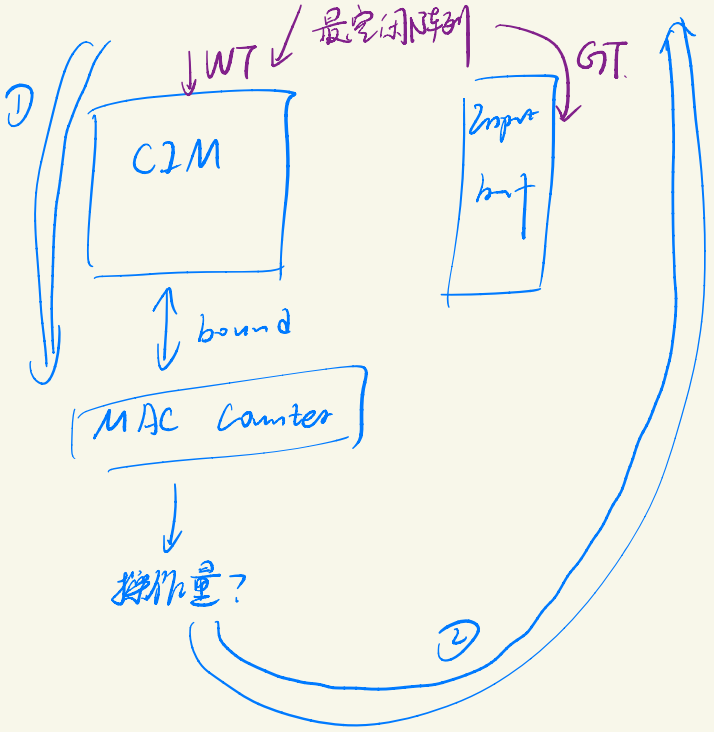
GT FIFO – 的有WT

每次到一个Effective ID.

Input Buffer ← load from Global Buffer — GT (col)

On-Demand-Fetcher ← load from DRAM — WT (row)

Balanced Inter-C2M Distributor.



Input - Look Ahead CIM

负载平衡 EOC1 后，仍然存在 空闲计算单元存在

传统方式 每次只处理一行输入，只能激活部分阵列

→ MAC 闲置 / 延时

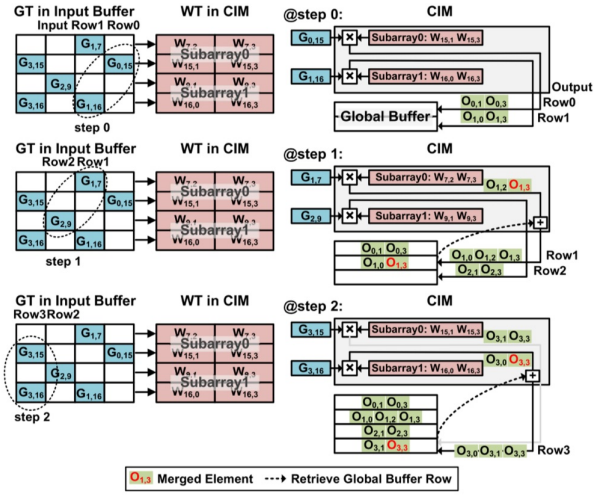


Fig. 16. ILA-CIM computation example.

传统方式 每次只处理一行：如 Input Row0 → 只激活 subarray 0.

此处优化：step0：处理 $G_{0,15}$ 时同时预看 $G_{1,16}$，把 subarray1 也激活，输出 Row0 和 Row1 写入 global buffer.

step1：global buffer 中已有部分和，先取出 $O_{1,3}$ 与结果作移位累加

step3：——

architecture：流水线结构

① Input LookAhead：提前准备下一输入，填满每个阵列 FIFO

② SRAM CIM Macro：{ SRAM Bank / Subarray / Multiplier } 计算 (Booth signals)

③ Output Merger：

合并阵列结果并写回 global buffer

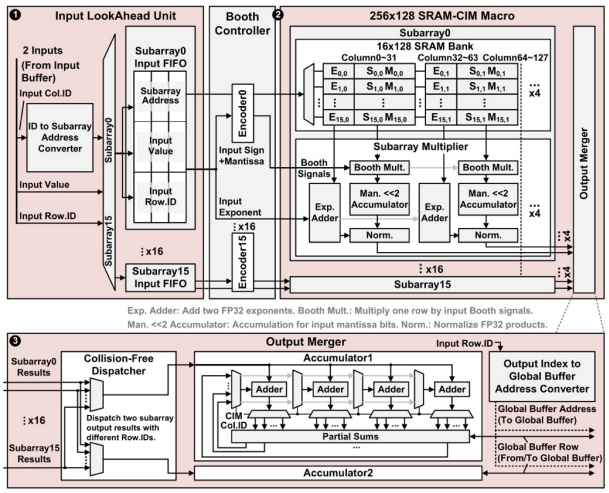


Fig. 18. ILA-CIM architecture: ❶ input lookahead, ❷ CIM multiplication, and ❸ output merging.

How: ①每个 CIM Core有 4个 1LA-CIM macro

每个 1LA-CIM macro有 16个 subarray

内有 1LA-CIM macro关于 Input Look Ahead unit / Booth Controller.

每周期读2个输入 ID + Value.

开始入FIFO.

当准备就绪后 可同时激活16个subarray

② CIM乘法:

每个 subarray有: 16×128个 6T SRAM 存储权重 + 4个 Booth 并行乘法器.

③输出合并:

从有subarray的输出结果 传入 Output Merger.

映射到 global buffer 地址.

把中间结果 读回寄存器.

最终结果写回 global buffer.

1LA-CIM 的datapath: 用于 SpA. 也可用于 SpG Reduction.