计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(8): 2005−2023, 2026

基于多智能体强化学习的多 Chiplet 加速器多 DNN 负载的在线调度

王晨赳[1,2]　张志锰[1,2]　郝沁汾[1]

[1]（处理器芯片全国重点实验室（中国科学院计算技术研究所）　北京　100190）

[2]（中国科学院大学　北京　100049）

（wangchenjiu20s@ict.ac.cn）

Multi-Agent Reinforcement Learning Based Online Scheduling for Multi-DNN Workloads on Multi-Chiplet Accelerators

Wang Chenjiu[1,2], Zhang Zhimeng[1,2], and Hao Qinfen[1]

[1]（*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190）

[2]（*University of Chinese Academy of Sciences*, *Beijing* 100049）

Abstract　With the increasing popularity of deep neural network (DNN) applications, Chiplet-based accelerators have become critical infrastructure for supporting large-scale DNN inference services on server platforms. However, the real-time and dynamic characteristics of parallel multi-DNN workloads pose severe challenges to scheduling algorithms, while existing algorithms suffer from limitations in decision overhead, service level agreement (SLA) assurance, and dynamic adaptability. To address these issues, we propose MAOSA, a multi-agent reinforcement learning based online scheduling algorithm for multi-DNN workloads on multi-Chiplet accelerators, aiming to maximize the overall SLA satisfaction rate. The proposed algorithm partitions DNN tasks into DNN blocks and models them as agents. It utilizes the attention mechanism to generate scheduling strategies in parallel, which include execution order and PE-level resource allocation, thereby significantly reducing decision overhead. Furthermore, a cooperative reward function considering time slack is designed to optimize SLA, and a periodic unified preemption mechanism is introduced to adapt to dynamic load changes. Experimental results show that, compared with state-of-the-art algorithms, the proposed algorithm improves the SLA satisfaction rate by up to 18% and system throughput (STP) by 31.9%, reduces tail latency by 28.7%, and decreases scheduling execution time by a factor of approximately 11, effectively enhancing system performance and service quality.

Key words　deep neural network; Chiplet; multi-agent reinforcement learning; online scheduling; service level agreement

摘　要　随着深度神经网络（DNN）应用日益广泛，基于芯粒（Chiplet）架构的加速器成为了服务器平台上支撑大规模 DNN 推理服务的关键设施。然而，并行多 DNN 负载的实时性与动态性特征给调度算法带来了严峻挑战，现有算法在决策开销、服务水平协议（SLA）保障及动态适应性方面存在局限。为此，提出了一种面向多 Chiplet 加速器的多智能体强化学习多 DNN 负载在线调度算法（MAOSA），并以最大化整体 SLA 满足率为目标。该算法将 DNN 任务划分为 DNN 块并建模为智能体，利用注意力机制并行生成包含执行顺序与 PE 级资源分配的调度策略，显著降低了决策开销；设计了考虑时间裕量的协作奖励函数以优化 SLA，并引入周期性统一抢占机制来适应负载动态变化。实验结果表明，与当前最先进算法相比，

收稿日期：2026-03-04；**修回日期**：2026-04-28

基金项目：江苏省创新支撑计划（软科学研究）专项（BE2023006-4）

This work was supported by the Jiangsu Provincial Special Program for Innovation Support (Soft Science Research) (BE2023006-4).

通信作者：郝沁汾（haoqinfen@ict.ac.cn）

所提出算法的SLA满足率最高可提升18%, STP提升31.9%, 尾延迟降低28.7%, 调度决策时间降低至约1/11, 有效提升了系统性能与服务质量。

关键词 深度神经网络; 芯粒; 多智能体强化学习; 在线调度; 服务水平协议

中图法分类号 TP302.7; TP181

DOI: 10.7544/issn1000-1239.202660134 **CSTR:** 32373.14.issn1000-1239.202660134

深度学习技术在解决复杂现实问题中发挥着越来越重要的作用, 以深度神经网络(deep neural network, DNN)为代表的技术方案已在多个领域证明了其有效性, 包括图像识别[1-5]、目标检测[6-7]、语言翻译[8-9]、音频合成[10]和自动驾驶[11]等。随着DNN模型复杂度与精度要求的持续提升, 本地终端设备有限的计算与存储资源已难以满足其执行需求。因此, 当前主流技术趋势是将计算任务卸载至服务器平台(包括边缘服务器[12-14]与云服务器[15-17])执行。

服务器平台通常可以部署大规模的DNN加速器以满足DNN推理的高性能需求。目前已有集成数百亿晶体管的大型单片加速器[18-21], 但单个芯片的计算与存储能力仍难以充分满足前沿大规模DNN模型的需求。受限于单芯片面积扩展瓶颈[22]以及后摩尔时代晶体管微缩放缓[23], 传统单片集成方案面临技术瓶颈。为此, 基于芯粒(Chiplet)的架构[24-29]应运而生。这种架构利用先进封装技术实现多芯片模块化集成, 克服了大尺寸单芯片的物理限制, 能够支持更大规模的晶体管集成, 进而有效扩展系统算力以匹配大规模DNN计算的需求, 有望成为服务器平台提供DNN推理服务的核心硬件基础。

面向DNN推理服务的服务器平台需具备多负载并行处理能力以应对多样化服务需求。例如, 在云端, 基于“推理即服务(inference-as-a-service, INFaaS)”范式[15], 服务商会封装底层算力资源以供多租户共享; 同时, 许多业务需部署异构DNN模型以优化用户体验, 这要求系统支持多模型的协同执行[30]。而在边缘侧, 边缘服务器同样需同时响应来自多用户的推理请求[31-33]。此外, 服务器场景下的DNN推理还呈现出实时性与动态性特征。实时性要求系统保障用户体验, 即严格满足服务水平协议(service level agreement, SLA)中的时间约束[16]。例如, MLPerf测试集[34]为图像分类、目标检测、机器翻译等任务设定了15~250 ms不等的延迟上限。动态性则表现为推理请求随时间持续且随机地到达, 尽管其到达过程可认为服从泊松过程[17,33-35], 但请求到达的随机性仍使得传统的静态调度算法难以适用。综上所述, 本文聚焦于基于多Chiplet加速器构建的服务器平台, 针对多DNN负载并行场景, 结合实时性与动态性特征研究在线调度算法, 核心目标是提高系统的整体SLA满足率。

在多Chiplet加速器上进行多DNN在线调度是一项具有挑战性的任务。多DNN并行执行时会争用加速器的计算、内存等资源, 且不同DNN任务在资源需求与映射方式上存在差异, 调度算法需在有限的系统信息下为各任务分配相对较优的资源。面对持续到达的推理请求, 调度算法还需合理确定任务执行顺序, 尽可能保证请求在截止时间前完成。传统的基于优化的算法[36-38]可以实现近似最优的效果, 但其计算复杂度高, 且面向静态任务集合设计, 难以适配服务器场景的动态特性。启发式算法[30,39-41]常被用于高复杂度问题, 但其往往缺乏全局视野, 通常无法获得最优调度方案。

强化学习(reinforcement learning, RL)具备在与环境交互中自适应学习策略的能力, 适合处理这类复杂调度问题[42-44]。现有研究已探索将深度强化学习算法应用于多Chiplet加速器上的多DNN负载在线调度, 但仍存在若干局限。例如RELMAS[45]采用基于LSTM结构的强化学习模型, 并以DNN层为最小调度单元, 虽然在小规模场景下有效, 但随着并发DNN数量增加, 模型推理开销急剧上升, 难以支撑实时决策。DTRL[46]、THERMOS[47]侧重于优化DNN执行延迟与能耗, 但忽视了对SLA满足率这一关键服务指标的考虑。最后, 现有强化学习算法缺乏灵活调度机制, 均不支持抢占调度。抢占机制允许任务在执行过程中被挂起并释放资源, 对于平衡不同DNN任务的执行时间与资源占用, 应对动态负载变化十分重要。

针对上述问题, 本文提出了MAOSA, 这是一种面向多Chiplet加速器的多智能体强化学习多DNN负载在线调度算法。该算法以提高推理请求的整体SLA满足率为核心目标。算法首先将到达的DNN任务依据其计算图的数据依赖关系划分为DNN块, 并以DNN块为基本调度单元, 在保留任务间并行性

的同时，有效促进了层间数据复用并缩减了决策空间。其次，我们将每个 DNN 块建模为智能体，利用基于注意力机制的强化学习模型并行生成调度策略，显著降低了决策开销。调度策略包含 DNN 块执行顺序与 PE 级的资源分配。同时，本文设计了面向多智能体协作的奖励函数，考虑了 DNN 任务完成时间与截止时间之间的时间裕量，引导任务利用该松弛时间，从而提升整体 SLA 满足率。此外，算法采用周期性统一抢占机制，在增强动态适应性的同时有效控制了抢占开销。最后，与现有启发式及强化学习算法进行了对比实验，实验结果表明本算法能够取得最优的 SLA 满足率与最低的尾延迟，同时可有效降低调度算法自身的执行时间。

本文的主要贡献总结如下：

1）将基本调度单元 DNN 块建模为智能体，提出了一种基于多智能体强化学习的多 DNN 调度算法，结合注意力机制实现并行调度决策以降低调度开销，利用松弛时间优化整体 SLA 满足率。同时引入周期性统一抢占机制，有效应对负载动态变化。

2）将 DNN 块的调度策略分解为执行顺序决策与 PE 级资源分配 2 部分：执行顺序考虑了 DNN 内部的数据依赖关系以及请求的时间约束；PE 级资源分配则可充分利用单 Chiplet 的计算资源并充分挖掘多 Chiplet 系统的并行计算潜力。

3）搭建了面向多 Chiplet 加速器的仿真平台，综合考虑了片上通信开销、多 DNN 并行执行时的资源争用等真实场景约束。基于该平台的 3 种典型多 DNN 负载实验表明，与当前最先进算法相比，本文算法的整体 SLA 满足率最高可提升 18%，系统吞吐量提升 31.9%，尾延迟降低 28.7%，调度算法决策时间降低至约 1/11。

1 背景与相关工作

1.1 DNN 计算

DNN 模型由一系列层组成，包括卷积层、池化层、激活层和全连接层等。激活层和池化层在执行时通常会合并到其他层以减少数据移动。如图 1 所示，卷积层计算可描述为 7 维嵌套循环，维度包括权重张量的高度（fh）和宽度（fw）、输出张量的高度（oh）和宽度（ow）、输入通道数（ic）、输出通道数（oc）和批量大小（N）。矩阵乘法可以使用将 fh、fw、oh、ow 设置为 1 的卷积来描述。矩阵向量乘也可以通过将 fh、fw、oh、ow、N 设置为 1 来描述。

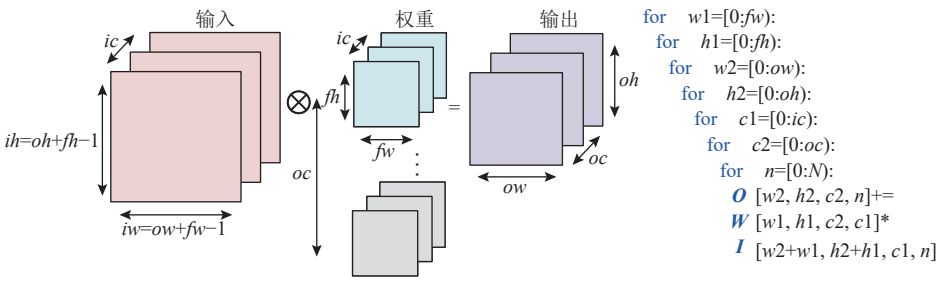


Fig. 1 Convolutional layer computation

图 1 卷积层计算

1.2 强化学习

强化学习问题通常被形式化为马尔可夫决策过程（Markov decision process, MDP），智能体通过与环境交互学习最优策略以最大化累积奖励，其基本框架如图 2 所示。MDP 可以由五元组 $(\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R},\gamma)$ 表示，其中 $\mathcal{S}$ 表示状态空间、$\mathcal{A}$ 表示动作空间、$\mathcal{P}(s'|s,\boldsymbol{a})$ 表示执行动作 $\boldsymbol{a}$ 后状态 s 到 s' 的状态转移概率、$\mathcal{R}$ 与 γ 分别表示奖励向量与折扣因子。智能体依据环境当前状态 s 与当前策略 π 选择动作 $\boldsymbol{a}$；环境基于该动作反馈下一状态 s' 与即时奖励 r。从状态 s 出发遵循策略 π 所获得的期望累积奖励可表示为状态价值函数 $V^{\pi}(s)$。随后，强化学习算法以环境反馈的奖励信号为依据，迭代更新智能体的策略 π 与价值函数 V^{π}，直至智能体到达终止状态或达到最大迭代步数。

当问题扩展至多个智能体时，强化学习通常被形式化为马尔可夫博弈（Markov game）[48-49]，可由六元组 $(N,\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R},\gamma)$ 表示，其中 N 表示智能体数量，$\mathcal{S}$

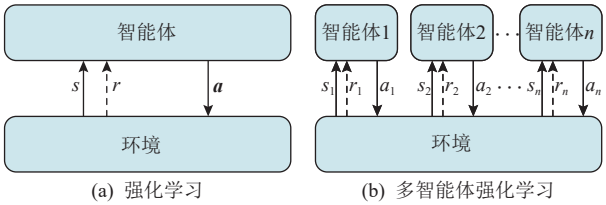


Fig. 2 Basic framework of reinforcement learning

图 2 强化学习基本框架

表示联合状态空间, $\mathcal{A}=\mathcal{A}_1\times\cdots\times\mathcal{A}_n$表示所有智能体的联合动作空间, $\mathcal{P}(s'|s,\boldsymbol{a})$表示执行联合动作 $\boldsymbol{a}=(a_1,\cdots,a_n)$ 后联合状态 s 到 s'的状态转移概率, $\mathcal{R}$表示各智能体的奖励函数集合。在每个决策时刻, 智能体 i 依据当前状态 s 与自身策略 π_i 选择动作 a_i; 环境基于所有智能体构成的联合动作 $\boldsymbol{a}$ 反馈下一状态 s' 以及各智能体的即时奖励 r_i。随后, 多智能体强化学习算法依据环境反馈的奖励信号, 协调更新各智能体的策略 π_i 与价值函数, 以最大化团队整体收益或自身累积奖励。

1.3 相关工作

表 1 展示了本文算法与现有工作的对比情况, 目前对多 DNN 负载调度的研究主要可分为静态与动态 2 类调度算法。

Table 1 Comparison of Multi-DNN Scheduling Algorithms

表 1 多 DNN 调度算法对比

工作	类型	复用情况	调度粒度	资源分配粒度	抢占	优化目标	决策方式
MOHaM	静态		层	Chiplet	不支持	延迟与能耗	元启发式
M2M			块	Chiplet	不支持	延迟	穷举搜索
PREMA	动态	时分	模型	整加速器	支持	SLA	启发式
CD-MSA		空分	模型	Chiplet	支持	SLA	启发式
RELMAS		空分	层	Chiplet	不支持	SLA	强化学习
THERMOS		空分	层	Chiplet	不支持	延迟与能耗	强化学习
MAOSA (本文)		空分	块	PE	支持	SLA	强化学习

1)静态调度算法。静态调度算法的目标是寻找到一组 DNN 任务到一组加速器的最优映射方案, 需要确定 DNN 任务的执行顺序和加速器配置等, 在任务执行开始后配置便不能更改。例如, MAGMA[36] 针对一组固定的 DNN 任务, 采用遗传算法, 将 DNN 任务与加速器配置进行联合编码, 以最大化访存带宽为目标进行多轮迭代搜索, 最终输出固定加速器配置下的任务映射方案。MoHAM[37] 在此基础上扩展为多目标优化框架, 同时考虑能耗与延迟等目标的权衡。虽然 2 种算法都使用了多 Chiplet 加速器, 但并未充分考虑其中的片上通信特点。M2M[38] 则引入 Chiplet 互连拓扑建模, 以延迟为优化目标进行任务映射。

总之, 静态调度算法通常需要遍历庞大的解空间并进行多轮迭代优化, 导致计算开销过高, 难以满足服务器场景下调度实时性的要求。此外, 静态调度算法基于固定任务集合的假设与推理服务的动态特性存在本质冲突, 实际场景中请求是随机到来的, 在各个时刻具有不同的任务集合。

2)动态调度算法。动态调度算法依据运行时状态实时调整任务映射策略, 符合服务器场景下动态负载的特点。目前此类算法的研究主要分为时分复用和空分复用 2 类。

基于时分复用的调度算法允许 DNN 任务在不同时段独占加速器, 从而支持多 DNN 任务并行。PREMA[39]、LazyBatching[40] 提出可抢占式加速器架构以实现灵活的多 DNN 调度, 有效平衡了不同类型任务的执行延迟和整体吞吐量。对抢占的支持可以使调度器不必等待当前任务结束就执行任务切换, 更加契合动态负载的特点。AI-MT[30] 和 LayerWeaver[41] 则采用非抢占式调度, 通过交替执行 DNN 任务的计算、内存密集部分实现资源利用率优化。时分复用算法的缺点是在某个时刻依然只支持单个任务独占加速器, 当加速器资源超过单个 DNN 任务可利用上限时便容易造成资源浪费。此外, 这些算法缺乏对服务质量(quality of service, QoS)的考虑, 难以在复杂场景下实现延迟保障与资源效率的协同优化。

基于空分复用的调度算法通过划分加速器计算资源实现多 DNN 任务并行执行。Planaria[50] 和 DM-NPU[51] 采用脉动阵列动态重构技术, 在单加速器内创建多个独立计算单元。这些算法聚焦于单加速器内部优化, 而且硬件重构会额外引入较多开销, 因此不适用于多 Chiplet 加速器任务调度。CD-MSA[35] 是松弛性感知在线调度算法, 引入了基于截止时间的松弛性分析, 使用周期抢占调度, 可以将多个请求融合后再提交到多 Chiplet 加速器执行。然而其融合任务有数量限制, 映射也只能以 Chiplet 为粒度进行。最后, 上述算法均以完整 DNN 模型为调度粒度, 未能挖掘模型间的并行潜力。

区别于上述基于启发式的空分复用算法, 目前最先进的基于强化学习算法进行多 DNN 调度的研究主要有 RELMAS[45]、DTRL[46]、THERMOS[47]。其中 RELMAS 以提高系统整体 SLA 满足率为目标, 以 DNN 层为调度粒度, 并使用基于 LSTM 结构的强化学习模型, 在每个调度周期按最早截止时间优先(earliest deadline first, EDF)排列的层需要顺序通过 LSTM 模型, 然后依次输出每个层的优先级以及映射的 Chiplet 优先级。这种只能顺序输出的调度方案导致调度决策开销巨大。同时它只能以 Chiplet 为粒度进行 DNN 映射, 这可能导致资源浪费, 并且难以适应负载的动态变化。DTRL、THERMOS 以最小化 DNN 在加速器上的执行延迟以及能耗为目标, 它们以层为调度粒

度，使用先到先服务（first come first serve, FCFS）的执行顺序，为当前 DNN 的每个层选择合适的 Chiplet 执行，同样也只能以 Chiplet 为分配粒度。这 2 种算法都没有考虑 QoS，只使用简单的调度顺序，并且只关注任务执行延迟，无法实现最佳的 SLA 满足率。最后，这些基于强化学习的算法均不支持抢占式调度，难以适应持续到达的动态负载。

2 系统模型和形式化

本节首先对服务器平台的推理请求以及多 Chiplet 加速器进行建模。然后介绍多 Chiplet 加速器上的 DNN 执行模型，包括 DNN 计算图划分、DNN 的并行策略以及加速器上的内存和带宽争用。最后，对多 Chiplet 加速器上多 DNN 调度问题进行了形式化。为便于阅读，本文提及的主要符号在表 2 中列出。

2.1 请求与加速器模型

本文考虑一个持续接收用户推理请求的服务器平台。依据文献 [17, 33−35]，推理请求的到达过程可建模为服从特定到达率的泊松过程。服务器将接收到的请求解析为相应的 DNN 模型，并在由多 Chiplet 构建的加速器平台上执行，如图 3 所示。加速器

Table 2 Summary of Main Notations

表 2 主要符号汇总

符号	解释
$G=(V,E)$	DNN 计算图
$G'=(V',E')$	DNN 划分后的计算图
$L_{\max}$	DNN 划分最大块长
b_{ij}	请求 i 的第 j 个块
ET_{ij}	块 b_{ij} 的执行时间
WT_{ij}	块 b_{ij} 允许执行后在调度队列中的等待时间
FT_i, FT_{ij}	请求 i、块 b_{ij} 的完成时间
DT_i	请求 i 的截止时间
$e^i_{j'j}$	计算图中块 $b_{ij'}$ 指向块 b_{ij} 的边
$\sigma(b_{ij})$	块 b_{ij} 在计算图中的拓扑顺序
$TS(b_{ij})$	块 b_{ij} 提交执行的顺序
S_k	第 k 次调度的开始时刻
B_t	时刻 t 正在执行的 DNN 块集合
C_b	分配给块 b 的 Chiplet 集合
P_b	分配给块 b 的 PE 集合
P^c_b	Chiplet c 上分配给块 b 的 PE 集合
M	加速器 Chiplet 总数
N	每个 Chiplet 上的 PE 数量

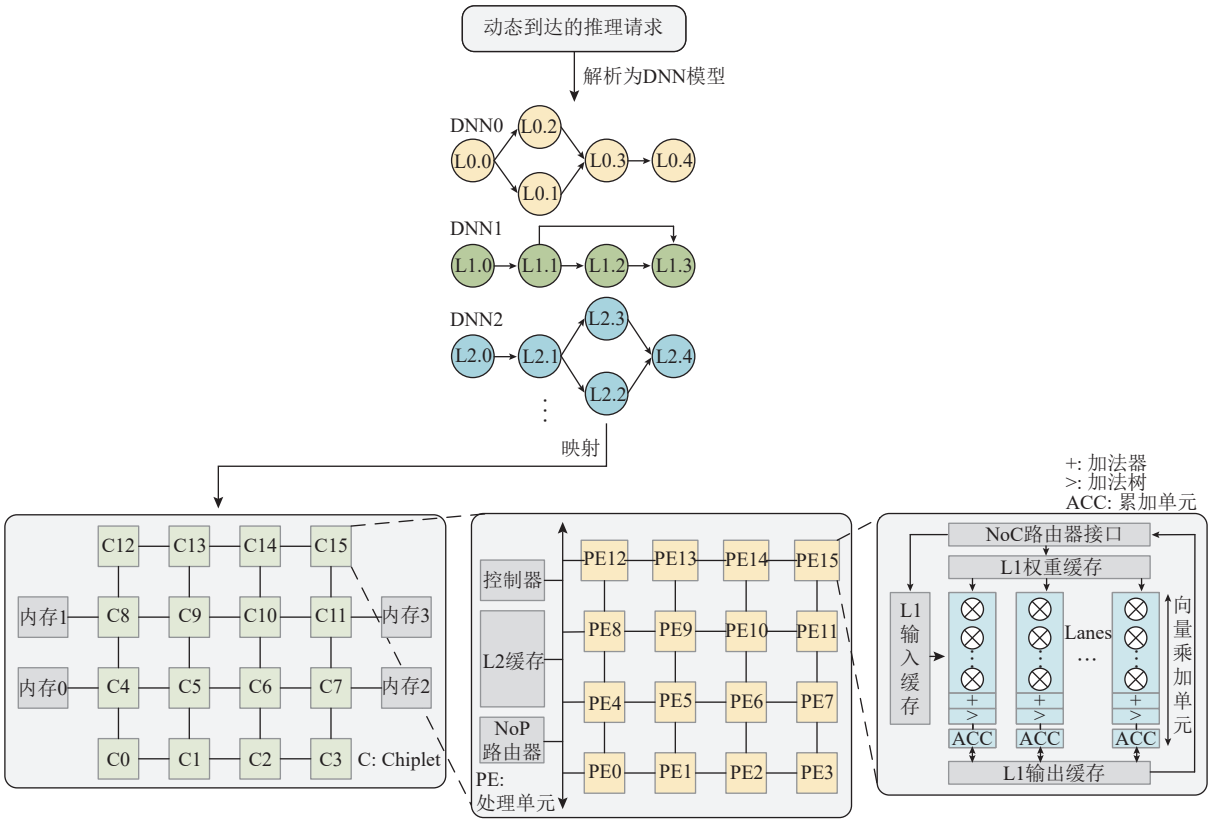


Fig. 3 Inference requests and multi-Chiplet accelerators

图 3 推理请求与多 Chiplet 加速器

由同构 Chiplet 经由 2D-mesh 网络互连构成。每个 Chiplet 内部集成多个功能相同的处理单元(processing element, PE)及由各 PE 共享的全局缓存(L2 缓存)。PE 间同样通过 2D-mesh 网络互连,内部包含执行乘加运算的计算单元与本地存储单元(L1 缓存),可独立执行 DNN 计算。本文以 PE 作为计算资源的基本分配粒度。此外,加速器配置了分布式片外内存,图 3 所示的加速器中,片外内存分别与 C4、C7、C8、C11 相连。采用同构 Chiplet 主要基于以下考量:首先,同构设计具有良好的可扩展性,通过增加 Chiplet 数量即可线性提升系统算力,同时简化了硬件设计、验证与部署流程[52];其次,这种设计支持 PE 级细粒度资源分配与映射,便于充分利用加速器的并行计算能力,提升资源利用率[53]。这种 PE 级的灵活调度更适合实时场景,调度器可将不同 DNN 任务动态分散至各 Chiplet 的 PE 上,从而有效应对请求到达的不确定性,最大化整体 SLA 满足率。相比之下,基于异构 Chiplet 的 DNN 加速器通常针对特定模型或算子定制数据流与存储层次,虽然在特定负载下能获得更高的峰值性能和能效[54],但其任务放置粒度较粗(通常以 Chiplet 为单位),难以适应动态到达的多 DNN 任务调度需求。此外,异构设计会加剧 Chiplet 间互连的不均匀性与路由复杂度,不利于实现可预测的低尾延迟。因此,本文选择同构 Chiplet,旨在通过精细的 PE 级调度,保证可扩展性的同时,提升整体 SLA 满足率。

2.2 DNN 划分

在多 DNN 并行执行场景下,若以整个模型为单位分配资源,由于模型内部各层资源需求差异较大,容易导致资源利用率低下。将模型划分为更小的调度单元独立分配资源,不仅能提升资源利用率,还能增加不同模型间的并行机会,进而改善整体 SLA 满足率。按层划分是一种简单方案,但当网络层数较多时,调度单元的激增将导致搜索空间膨胀,难以满足实时调度需求。考虑到相邻层往往具有相似的计算特征与数据依赖,将其融合为整体调度有利于数据复用。因此,本文采用 DNN 块作为基本调度粒度。DNN 块由若干连续层组成,当块长度为 1 时即退化为单层划分。

算法 1. 基于图遍历与贪心策略的 DNN 划分算法。

输入:DNN 计算图 $G=(V, E)$,最大块长度 $L_{\max}$;

输出:DNN 块集合 B、DNN 块层次列表 L。

① 初始化:将 G 所有输入层加入队列 Q, $B \leftarrow \varnothing$;

② while Q 非空 do

③ $v \leftarrow Q.dequeue()$,初始化当前块 $b \leftarrow []$;

④ while v 未访问 do

⑤ $b.append(v)$,标记 v 为已访问;

⑥ $C \leftarrow \{u \in children(v) | u \notin Visited\}$;

⑦ if $|C| = 0$ or ($|C| = 1$ and $|b| \geqslant L_{\max}$) then break;

⑧ else if $|C| = 1$ then $v \leftarrow C[0]$;

⑨ else

⑩ 贪心选择下游路径最长的子节点 $c^* \in C$,其余子节点 $C \backslash \{c^*\}$ 入队 Q;

⑪ 将 c^* 起始的线性路径合并入 b,确保 $|b| \leqslant L_{\max}$,标记加入块的层已访问;

⑫ break;

⑬ end if

⑭ end while

⑮ $B \leftarrow B \cup \{b\}$;

⑯ end while

⑰ 根据块集合 B,使用 Kahn 算法构建层次列表 L;

⑱ return B, L。

算法 1 展示了 DNN 划分算法。DNN 模型可以由计算图 $G=(V, E)$ 描述,G 是一个有向无环图(directed acyclic graph, DAG),V 表示操作节点,E 表示数据依赖关系。算法 1 在遍历计算图的过程中会动态地将层融合为块,结束当前块划分的条件(行⑦⑨)包括:1)块的层数达到最大长度限制 $L_{\max}$。2)遇到分支点,即当前层具有多个子层,此时需将该层输出写回片外内存供子层共享,故不宜继续融合。3)遇到汇合点,即当前层的唯一子层已经被访问过。针对分支节点,算法采用贪心策略选择下游路径最长的子分支作为当前块的延伸(行⑩~⑪)。划分完成后,根据原依赖关系生成以 DNN 块为节点的新计算图 $G'=(V', E')$,并利用 Kahn 算法进行拓扑排序,形成层次列表 L,同一层次的块间没有依赖关系,可以并行执行(行⑰)。

基于 DNN 块的调度算法在提升并行度与资源利用率的同时,有效控制了调度单元规模,降低了算法复杂度。在执行阶段,DNN 块作为基本单元被提交至加速器,其内部各层按序执行。同一块内相邻层的中间数据可驻留于 Chiplet 的 L2 缓存,实现高效的数据复用并减少片外访存。块执行结束后,输出结果写回至距离其所在 Chiplet 最近的片外内存,供后续块读取。

2.3 DNN 执行模型

2.3.1 并行策略

DNN 块作为基本调度单元提交至加速器执行,

其内部各层按序计算。由于块可能被分配多个 PE，每层可进行数据划分实现并行计算。DNN 层映射涉及庞大的搜索空间（如 7 维循环变换与展开），传统搜索方法（如 Timeloop[55]、GAMMA[56]）虽能求解最优映射，但时间开销过高，不适用于实时调度场景。为此，本文借鉴 AIMT[30] 的启发式策略，以最大化数据复用为目标，将 PE 间的并行方式分为输入并行与权重并行，如图 4 所示。对于卷积层，鉴于卷积核较小且复用率高，采用输入并行策略：按输出张量维度划分输入数据，各 PE 持有完整卷积核；对于全连接层，输入数据复用率通常高于权重，采用权重并行策略：均分权重至各 PE，各 PE 持有完整输入数据。PE 内部映射则采用类 SIMBA[26] 架构的 oc-ic 并行方式，即采用权重静止数据流，在输出通道维度与输入通道维度划分数据，分发至底层乘加单元计算。

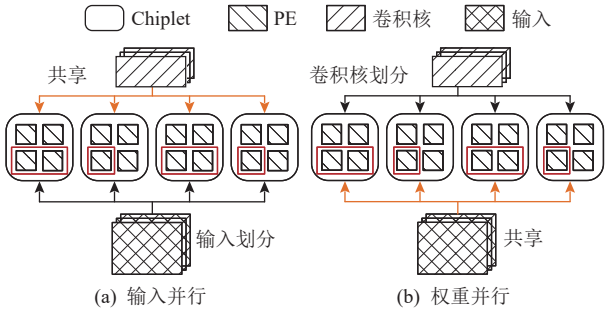


Fig. 4 Parallelization strategy

图 4 并行策略

DNN 块内部层的中间结果暂存于 Chiplet 的 L2 缓存以供复用。由于执行 DNN 计算的 PE 可能分布于不同 Chiplet，层间切换往往会产生跨 Chiplet 通信。该通信开销受并行策略影响，图 5 展示了其中 1 种情况：层 l–1 采用输入并行，输出分布于各 Chiplet；后续层 l 采用权重并行，因此每个 Chiplet 需获取完整输入数据。因此，执行层 l 的各 Chiplet 需从其他 Chiplet 读取中间结果，导致层间切换通信开销。

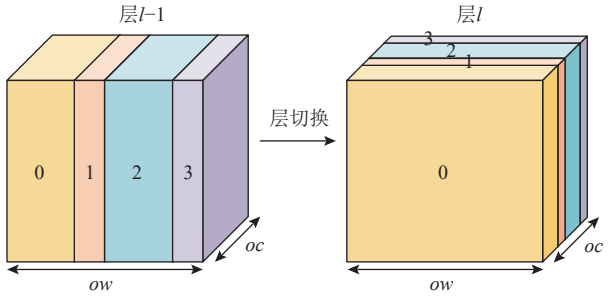


Fig. 5 Example of DNN layer switching for Chiplet 0-3

图 5 Chiplet 0-3 上 DNN 层切换示例

2.3.2 内存和带宽争用

为评估调度算法性能，本文基于 Timeloop[55] 搭建了多 Chiplet 加速器仿真平台。该平台利用 Timeloop 估算 DNN 层在 PE 及 Chiplet 上的执行延迟，以及带宽、内存需求，并扩展了 Timeloop 缺乏的 Chiplet 间通信模拟功能。此外，针对多任务并行导致的资源争用（如链路、L2 缓存及片外内存 DRAM 争用），平台建立了相应的延迟模型。以链路争用为例，当任务带宽需求总和超过链路最大带宽时，相关任务将面临性能降级。通过量化争用开销，该平台能准确反映多 DNN 并行执行的性能。

DNN 执行涉及 DRAM-L2-PE 三级存储层次。利用流水线技术与双缓冲机制，计算时间与通信时间可重叠。因此，DNN 层 l 的执行延迟定义如式（1）所示。

$$Latency = \max(Latency_{\mathrm{PE}}, Latency_{\mathrm{DRAM}}, Latency_{\mathrm{L2}}, Latency_{\mathrm{Network}}), \quad (1)$$

$$Latency_{\mathrm{DRAM}} = Traffic_{\mathrm{DRAM}} / BW_{\mathrm{DRAM}}, \quad (2)$$

$$Latency_{\mathrm{L2}} = Traffic_{\mathrm{L2}} / BW_{\mathrm{L2}}, \quad (3)$$

$$Latency_{\mathrm{Network}} = \max_{\mathrm{link}} (Traffic_{\mathrm{link}} / BW_{\mathrm{link}}), \quad (4)$$

其中 $Latency_{\mathrm{PE}}$ 为 PE 内部计算延迟，由于本文使用 PE 级资源分配，故其不受外部资源争用影响。$Latency_{\mathrm{DRAM}}$ 和 $Latency_{\mathrm{L2}}$ 分别为 DRAM 和 L2 缓存访问延迟，由访问数据量与带宽之比决定，数据量（含输入加载、权重加载及输出回写）可由 Timeloop 分析得出。$Latency_{\mathrm{Network}}$ 是网络通信延迟，包括 Chiplet 之间以及内部通信，取决于路由路径中负载最重的链路延迟。

式（1）中的 $Latency_{\mathrm{DRAM}}$、$Latency_{\mathrm{L2}}$ 以及 $Latency_{\mathrm{Network}}$ 均会受到带宽、内存争用的影响。为统一建模，我们将网络链路与内存访问统称为“链路”，其无争用时的理想延迟记为 $Latency_{\mathrm{link}}$。我们首先考虑带宽争用。由于 DNN 层按照循环嵌套的方式执行，其访存行为具有规则性，因此可假设争用导致的性能下降在执行期间均匀分布。定义带宽争用导致的性能下降因子 $Slowdown$ 如式（5）所示。若总需求带宽超过供给带宽，任务将经受等比例的性能下降，如式（6）所示。

$$Slowdown = \min\left(BW_{\mathrm{sup}} / \sum BW_{\mathrm{req}}, 1\right), \quad (5)$$

$$Latency'_{\mathrm{link}} = Latency_{\mathrm{link}} / Slowdown, \quad (6)$$

其中 BW_{sup} 为链路最大带宽，$\sum BW_{\mathrm{req}}$ 为该链路上所有 DNN 任务的需求带宽之和。

此外，同一 Chiplet 上的多 DNN 并行可能导致 L2 缓存容量争用。当 L2 容量不足以容纳所有任务的

工作集时, 溢出数据将导致额外的 DRAM 访问, 从而增加 DRAM 带宽需求。基于均匀性假设, DRAM 带宽需求的增长因子 *Increase* 可由式(7)得到, 本文假设 DRAM 容量充足, 仅考虑 L2 容量争用。若总需求容量超过 L2 容量, 那么 DRAM 的带宽需求将增加, 如式(8)所示。

$$Increase = \max\left(\sum Mem_{\text{req}}/Mem_{\text{sup}}, 1\right), \tag{7}$$

$$BW'_{\text{req}} = BW_{\text{req}} \times Increase, \tag{8}$$

其中$\sum Mem_{\text{req}}$为 L2 内存总需求, Mem_{sup}为 L2 容量。

综合上述模型, DNN 层的实际执行延迟计算如下: 首先利用式(8)修正因内存争用增加的 DRAM 带宽需求; 接着将修正后的带宽代入式(5)计算 *Slowdown*; 最后通过式(6)求得包含争用开销的实际延迟。由于各 Chiplet 上的计算并行进行, 层 l 的最终延迟由执行时间最长的 Chiplet 决定。DNN 块的总执行延迟则为其内部所有层的延迟之和。

2.3.3 完成时间

基于 2.3.2 节的延迟模型, 设 DNN 块b_{ij}的执行延迟为ET_{ij}。考虑到块间数据依赖关系, 一个 DNN 块仅在其所有前驱块执行完毕后方可开始执行。因此, 块b_{ij}的完成时间FT_{ij}定义为:

$$FT_{ij} = \max_{e^i_{j'j} \in E'_i} FT_{ij'} + WT_{ij} + ET_{ij}, \tag{9}$$

其中, E'_i表示 DNN 模型 i 的有向边集合, $e^i_{j'j}$表示从块$b_{ij'}$指向b_{ij}的边。$\max FT_{ij'}$项表示所有前驱块完成时间的最大值, 即b_{ij}的最早就绪时间(若b_{ij}无前驱, 则$\max FT_{ij'} = 0$)。WT_{ij}表示块b_{ij}就绪后在调度队列中的等待时间, 取决于具体的调度算法与资源可用性。

2.4 问题形式化

设第 k 次调度的开始时刻为 S_k, 此时待调度的 DNN 块集合为 A_k。调度决策包含时间与空间 2 个维度: 前者确定块的提交顺序, 后者确定其计算资源分配。提交顺序必须遵循 DNN 块间的数据依赖关系, 即满足约束:

$$\forall \sigma(b_{ij'}) \leqslant \sigma(b_{ij}), TS(b_{ij'}) \leqslant TS(b_{ij}), \tag{10}$$

其中, $\sigma(b_{ij})$表示块b_{ij}在 G'中的拓扑顺序, $TS(b_{ij})$表示该块的提交顺序。式(10)确保了 DNN 块仅在其所有前驱块提交后方可提交。

对于任意时刻 t $(S_k < t \leqslant S_{k+1})$, 设加速器上正在执行的 DNN 块集合为$B_t$ $(B_t \subseteq A_k)$。定义C_b为分配给块 b 的 Chiplet 集合, P_b为 PE 集合, 它们需要满足如下约束:

$$P_b = \{P_b^c | c \in C_b, b \in B_t\}, \tag{11}$$

$$1 \leqslant |C_b| \leqslant M; \forall c \in C_b, 1 \leqslant \left|P_b^c\right| \leqslant N, \tag{12}$$

其中, P_b^c表示 Chiplet c 上分配给块 b 的 PE 集合, M 为 Chiplet 总数, N 为单个 Chiplet 的 PE 数量。式(12)限定了单个块可使用的资源上限。此外, 多任务并行时的资源分配还需满足系统容量约束:

$$\forall c, \sum_{b \in B_t} \left|P_b^c\right| \leqslant N。 \tag{13}$$

式(13)确保在任意时刻, 每个 Chiplet 上所有活跃任务占用的 PE 总数不超过其物理上限。

基于上述时间、空间调度方案, 块b_{ij}的完成时间FT_{ij}可由式(9)计算得出, 请求 i 的完成时间FT_i即为其最后一个块的完成时间。调度算法的优化目标为最大化整体 SLA 满足率, 定义为:

$$rate = \left(\sum_{i \in RQ} \delta_i\right) / |IR|, \tag{14}$$

$$\delta_i = \begin{cases} 1, & FT_i \leqslant DT_i \\ 0, & FT_i > DT_i \end{cases}, \tag{15}$$

其中IR表示请求集合, DT_i表示请求 i 的截止时间, RQ 表示就绪队列。

3 基于多智能体强化学习的调度算法

3.1 概 述

多 Chiplet 加速器上的多 DNN 任务调度面临资源需求不一、系统约束复杂以及负载动态变化等多重挑战, 且实时性要求进一步增加了决策难度。深度强化学习通过智能体与环境的持续交互学习最优策略, 为解决此类复杂决策问题提供了有效途径。图 6 展示了本文所提出的 MAOSA 算法的调度流程以及多智能体强化学习模型结构。在每个调度时刻, 调度器观测系统状态并做出决策, 以最大化长期累积奖励(即 SLA 满足率)。具体而言, 到达的推理请求经解析与划分后, 生成的 DNN 块被放入就绪队列 RQ。此外, RQ 还包含经上一时刻调度后此时仍尚未完成的块。我们将 RQ 中的每个 DNN 块视为一个智能体, 通过多智能体强化学习算法确定其执行顺序与资源分配。由于优化目标为全局 SLA 满足率, 各智能体间需协作而非竞争, 因此建模为合作式多智能体问题。

为解决 RQ 中智能体数量动态变化及块间潜在的资源竞争关系, 本文采用基于自注意力机制的网络作为策略函数π_θ。在时刻 t, 将此时的就绪队列

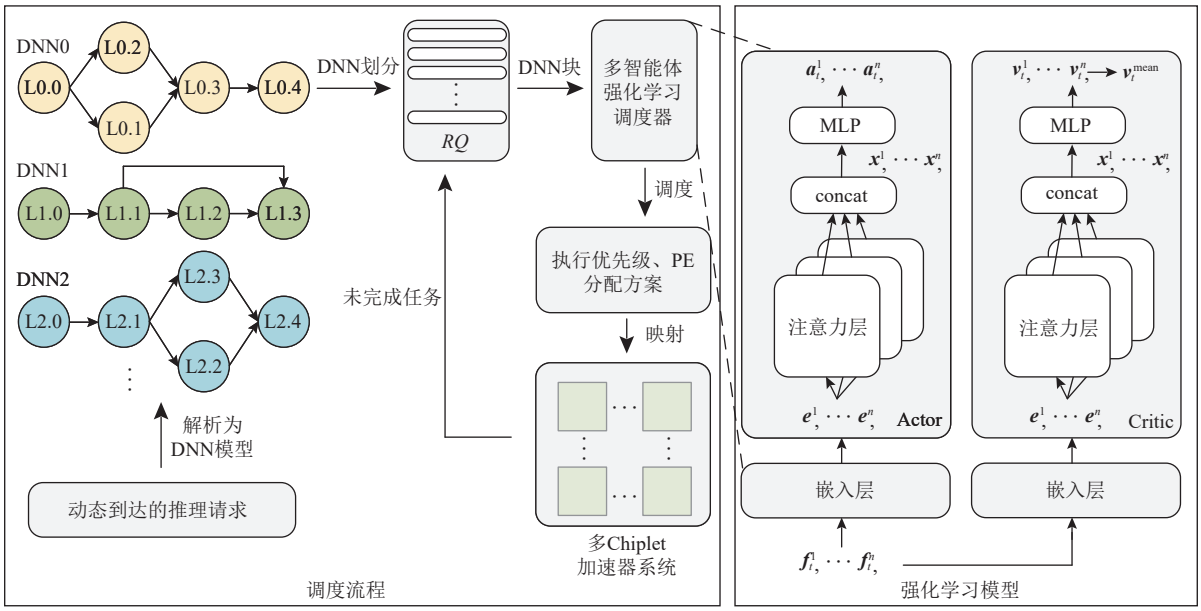


Fig. 6 Framework of MAOSA algorithm

图 6 MAOSA 算法框架

RQ_t视为无序特征集合$\{\boldsymbol{f}_t^1, \boldsymbol{f}_t^2, \cdots, \boldsymbol{f}_t^n\}$，通过注意力层捕捉全局上下文信息，生成增强特征表示，再经多层感知机（MLP）映射到动作空间：

$$\mathcal{A}_t = MLP(Attention(\boldsymbol{F}_t)), \tag{16}$$

其中$\boldsymbol{F}_t$是RQ_t的特征矩阵。该架构支持并行生成动作，避免了 LSTM 等模型的串行推理延迟，满足实时调度需求。此外，鉴于 RQ 中的 DNN 块均属同类计算任务且目标一致，智能体具有同构性，因此本文采用参数共享机制[57]，即所有智能体共享同一策略网络π_θ。结合注意力机制与参数共享，模型能高效处理变长输入，且输出具有排列不变性，确保决策仅依赖于块的内在属性。算法的具体技术细节，包括状态空间、动作空间、奖励函数设计及网络训练过程将在后续章节阐述。

3.2 强化学习构建

1）状态空间设计

在调度时刻 t，系统状态$\mathcal{S}_t$由就绪队列RQ_t中智能体的状态以及加速器状态所决定，定义为：

$$\mathcal{S}_t = \{\boldsymbol{f}_t^b | b \in RQ_t\} \cup MAS_t, \tag{17}$$

其中，$\boldsymbol{f}_t^b$为智能体 b 的特征向量，MAS_t为加速器状态。特征向量涵盖了可能影响调度决策的关键信息，具体定义见表 3。

具体而言，状态特征主要包含 3 类信息。

①资源需求特征，包括预估执行延迟T_{est}、带宽BW_{est}与内存MEM_{est}。T_{est}用于量化计算资源与执行效率的权衡，选取了 5 种配置下的值：单 Chiplet 内的 1、N/2 个 PE 配置，以及跨 Chiplet 的 1、2、4 个 Chiplet 配置。这些数据基于 MoCA[58] 的算法估算，反映了资源扩展对延迟的影响。BW_{est}包括该 DNN 块的 L2 缓存与 DRAM 带宽需求，MEM_{est}则为 L2 预估内存占用，本文基于 NN-Baton[53] 的算法进行估算。

Table 3 Definition of RL States

表 3 强化学习状态定义

状态	解释
T_{est}	DNN 块预估执行延迟
BW_{est}	预估带宽需求
MEM_{est}	预估内存占用
$DRAM_dist$	输入数据在 DRAM 上的分布比例
T_{re}	DNN 块剩余时间
$Request_id$	DNN 块所属请求序号
$Level_id$	DNN 块所属层次序号

② 数据分布特征，即输入数据的 DRAM 分布比例$DRAM_dist$。本文设置 DNN 块的最终计算结果将写回距其所在 Chiplet 最近的 DRAM，因此后续块可能需跨 DRAM 读取数据。该特征描述了输入数据在各 DRAM 上的存储比例（假设初始输入均匀分布），辅助模型优化通信开销。

③ 调度上下文特征，包括 DNN 块的剩余时间T_{re}（距截止时刻的时间）、请求序号$Request_id$和层次序号$Level_id$，用于辅助模型确定调度优先级与依赖约束。

针对加速器状态MAS_t，如果采用异步抢占，即

对每一次任务到达分别响应,那么每一次调度时所面临的加速器运行状态均不相同,需对该状态进行完整的建模,但是显式建模链路带宽、内存占用等细微状态会导致状态空间爆炸,增加训练收敛难度与硬件监控开销。更重要的是,若在异步抢占调度中纳入运行时状态,抢占引发的状态突变将导致决策时的观测值失效,影响调度准确性。例如,假设需对当前到达的任务 A 进行调度,智能体结合此时的加速器状态及其他必要信息做出了任务 A 抢占当前正在加速器上执行的任务 B 的决策。之后任务 B 中断执行,并且释放了其占用的加速器资源。然而,由于这些资源的释放,加速器状态将发生变化,而任务 A 的抢占决策是根据之前的状态做出的,这将导致该决策不准确。为解决上述问题,本文采用周期性统一抢占机制:调度决策将周期性地进行,并对 2 个调度时刻之间到达的任务进行统一调度。在每个调度时刻起始,中断所有正在执行的 DNN 块,将其执行现场保存至 DRAM 并重新加入 RQ 参与调度。此时,加速器处于确定的空闲状态,系统状态具有一致性,无需显式建模 MAS_t,其资源信息已隐含于各块特征的归一化过程中。该机制不仅大幅压缩了状态空间,还赋予了未完成任务重组资源的机会,实现了低开销的周期性抢占。此外,由于调度决策时间(即强化学习模型推理时间)是大致固定的,周期性调度可以分摊此时间,相比之下异步抢占则需要对每一个到达的任务均进行一次调度,增加了调度开销。最后,尽管统一抢占可能需要同时为更多的任务保存现场,但保存现场的时间可与调度决策时间重叠,从而有效隐藏了抢占开销。

2)动作空间设计

智能体的动作 $\boldsymbol{a}_t^b$ 对应于 DNN 块的调度策略,动作集合 $\mathcal{A}_t$ 定义为:

$$\mathcal{A}_t = \{\boldsymbol{a}_t^b | b \in RQ_t\}\text{。} \tag{18}$$

本文为智能体定义了 3 种动作,如表 4 所示。

Task_prio 决定了 DNN 块的执行顺序,属于时间调度维度。空间调度维度则负责资源映射,由于 PE 数量会显著影响 DNN 块执行时间,但过量分配则会导致资源浪费与任务挤占,因此需为其确定合适的 PE 分配数量。为降低动作空间复杂度并兼顾通信局部性,本文将空间调度设计为 *Chiplet_prio_seq* 与 *Num_pes* 的组合。具体映射过程中采用贪心策略:DNN 块根据 *Chiplet_prio_seq* 指定的优先级顺序依次遍历 Chiplet,并在每个 Chiplet 上分配空闲 PE,直至累计分配数量达到 *Num_pes* 或系统无空闲资源为止。为防止资源碎片化,PE 分配粒度设定为 4,即以 4 的倍数进行数量分配。该设计可以利用 Chiplet 内部通信优于片间通信的特性,优先填充同一 Chiplet 内的 PE,有效降低了通信开销。其次,相比于直接映射 PE 位置(动作维度高达 $N\times M$)的方法,该解耦策略将动作空间缩小至 Chiplet 数量级,降低了模型训练难度,使其能适应大规模调度场景。

Table 4 Definition of RL Actions

表 4 强化学习动作定义

动作	解释
Task_prio	DNN 块执行优先级
Chiplet_prio_seq	Chiplet 优先级序列
Num_pes	目标 PE 分配数量

3)奖励函数设计

由于调度目标为最大化系统的整体 SLA 满足率,为此需设计奖励函数引导智能体优化调度决策。奖励函数定义为:

$$\mathcal{R}_t = \frac{1}{|RQ_t|}\sum_{b\in RQ_t} h_b\left(BaseRew_b + \beta\frac{DT_b - FT_b}{QT_b}\right)\text{,} \tag{19}$$

$$h_b = \begin{cases} 1, & \text{块}b\text{完成,} \\ 0.01, & \text{块}b\text{未完成,} \end{cases} \tag{20}$$

$$BaseRew_b = \begin{cases} 0.1, & FT_b \leqslant DT_b\text{,} \\ -0.11, & FT_b > DT_b\text{。} \end{cases} \tag{21}$$

奖励函数主要由 2 部分构成。① 基础奖励:$BaseRew_b$ 用于衡量 SLA 约束满足情况。若块 b 在截止时间 DT_b 前完成,给予正奖励;否则施加负惩罚,引导模型优先满足延迟约束。设置惩罚的绝对值略大于正奖励是为了在不满足 SLA 时给予模型更高的惩罚,避免消极调度。② 时间裕量奖励项:由 $(DT_b - FT_b)/QT_b$ 计算,表示完成时间相对于截止时间的余裕或超时比例。其中 FT_b 表示块 b 的完成时间,QT_b 表示块 b 的 QoS 时间约束。该设计在保证 SLA 基础上,进一步鼓励任务增加完成时间与截止时刻之间的时间裕量,为后续任务留出更多执行时间,有利于整体 SLA 的提升。β 衡量了对该奖励项的重视程度,由于本文最终目标是提高 SLA 满足率,因此 β 的设置需使时间裕量奖励小于基础奖励,否则会导致模型倾向于优先调度执行时间更小的任务,而不是综合考虑所有相关因素以服务于整体 SLA 满足率这一目标。

对于本轮调度周期内未完成的块,其完成时间尚不能确定,通过系数 h_b 削弱其奖励信号,避免不确

定信息干扰训练，因此设置其比完成块奖励小 2 个数量级。最后，考虑到 RQ 长度动态变化，采用平均奖励机制构建团队奖励 $\mathcal{R}_t$。该设计不仅消除了队列长度波动对奖励尺度的影响，还确保所有智能体基于全局反馈协同进化，共同优化整体 SLA 满足率。

3.3 训　练

基于前文定义的状态空间、动作空间与奖励函数，本文采用 MAPPO 算法[59]进行训练。算法基于 Actor-Critic（演员-评论家）架构，所有智能体共享网络参数。Actor 网络 π_θ 根据观测特征 $\boldsymbol{f}_t^b$ 生成动作 $\boldsymbol{a}_t^b$；Critic 网络 V_ϕ 用于估计状态价值。两者采用相似的网络架构（见图 6 强化学习模型），主要区别在于输出层 Actor 输出动作分布，而 Critic 输出标量价值，并通过平均化操作聚合各智能体的价值估计以评估全局状态。针对调度时刻智能体数量动态变化且规模庞大的问题，训练过程设定了最大智能体数量 $N_{\max}$，每轮训练优先选取预估执行时间较短的智能体，数量不超过 $N_{\max}$。

Actor 网络需要被训练为最大化目标函数 $L(\theta)$：

$$L(\theta)=\left[\frac{1}{n}\sum_{b=1}^{n}\min\left(r_{\theta,t}^{b}A_t^b,clip\left(r_{\theta,t}^{b},1-\epsilon,1+\epsilon\right)A_t^b\right)\right]+c_1\frac{1}{n}\sum_{b=1}^{n}H\left[\pi_\theta\left(f_t^b\right)\right],\tag{22}$$

$$r_{\theta,t}^{b}=\frac{\pi_\theta\left(a_t^b|f_t^b\right)}{\pi_{\theta_{\text{old}}}\left(a_t^b|f_t^b\right)},\tag{23}$$

其中 $r_{\theta,t}^{b}$ 为概率比率，用于衡量新旧策略的差异。A_t^b 为采用广义优势估计（GAE）计算的优势函数，$clip(\cdot)$ 为裁剪函数用以控制策略更新的幅度，H 为策略熵，c_1 为策略熵系数，n 为智能体数量。

Critic 网络需要被训练为最小化损失函数，如下：

$$L(\phi)=c_2\max\left[\left(V'_{\phi,t}-\mathcal{R}_t\right)^2,\left(clip(V'_{\phi,t},V'_{\phi_{\text{old}},t}-\epsilon,V'_{\phi_{\text{old}},t}+\epsilon)-\mathcal{R}_t\right)^2\right],\tag{24}$$

$$V'_{\phi,t}=\frac{1}{n}\sum_{b=1}^{n}V_\phi\left(f_t^b\right),\tag{25}$$

其中 $V'_{\phi,t}$ 为全局状态价值估计（各智能体局部价值的均值），$\mathcal{R}_t$ 为团队奖励，c_2 为价值损失系数。最后，训练所用的超参数设置如表 5 所示。

Table 5　Hyper-Parameters of RL

表 5　强化学习超参数

参数	取值
隐藏层维数	64
学习率	3E−4
奖励折扣系数 γ	0.99
余裕项奖励系数 β	0.05
单步最大智能体数 $N_{\max}$	512
GAE λ	0.95
裁剪参数 ϵ	0.2
策略熵系数 c_1	0.01
价值损失系数 c_2	1

3.4 调度算法

基于上述设计，本文提出了 MAOSA 算法。

算法 2. MAOSA。

输入：到达率为 λ 的 DNN 请求流，训练得到的 Actor 网络 π_θ，单步最大智能体数 $N_{\max}$；

输出：DNN 请求执行结果。

① 初始化：就绪队列 $RQ\leftarrow\varnothing$，剩余块集合 $B_{\text{rem}}\leftarrow\varnothing$，时间 $t\leftarrow 0$；

② while 存在未完成请求 or $RQ\neq\varnothing$ or $B_{\text{rem}}\neq\varnothing$ do

③　$B_{\text{new}}\leftarrow$新请求进行 DNN 划分（算法 1）；

④　$RQ\leftarrow RQ\cup B_{\text{new}}\cup B_{\text{rem}}$；

⑤　对 RQ 按预估执行延迟升序排列，选择 $\min(|RQ|,N_{\max})$ 数量的块，得到集合 B；

⑥　$RQ\leftarrow RQ\backslash B$；

⑦　对于 $b\in B$，使用 π_θ 并行生成动作 $a^b=\{Task_prio^b, Chiplet_prio_seq^b, Num_PE^b\}$；

⑧　对 B 按 $Task_prio^b$ 降序排列；

⑨　执行集合 $E\leftarrow\varnothing$；

⑩　while 未到达下一调度时刻 do

⑪　　for $b\in B$ and b 的依赖已满足 do

⑫　　根据 $Chiplet_prio_seq^b$ 与 Num_PE^b 进行贪心分配（3.2 节）；

⑬　　$E\leftarrow E\cup\{b\}, B\leftarrow B\backslash\{b\}$；

⑭　　if 加速器无空闲 PE then break;

⑮　　end if

⑯　end for

⑰　加速器并行执行 E 中的块，直到有块结束执行，将其从 E 中删去；

⑱　end while

⑲　中断 E 中未完成的块，保存执行状态，与 B 中剩余块合并得到 B_{rem}；

⑳　$t\leftarrow t+\Delta t$；

㉑ end while

本文设定 2 种调度触发条件：固定周期触发，或当前调度周期内所有待调度块均执行完毕时触发。在每个调度时刻，调度器首先对新到达请求执行

DNN 划分(算法 1), 在算法 2 中, 生成的 DNN 块与上一时刻未完成的块共同组成就绪队列 RQ(行④)。随后, 依据预估执行延迟对队列排序, 选取不超过 N_{max} 个块输入 Actor 网络, 并行输出包含执行优先级、Chiplet 优先级序列及 PE 数量的动作向量(行⑤~⑦)。执行阶段, 算法按优先级从高到低依次处理各块。为确保执行正确性, 提交前需过滤掉依赖未满足的块。资源映射采用贪心策略, 根据 Chiplet 优先级序列选择目标 Chiplet 并分配空闲 PE(粒度为 4), 直至满足动作中的数量需求或资源耗尽(行⑫~⑭)。接着所有分配了 PE 的块将在加速器上并行执行, 如果有 DNN 块执行完成, 那么调度器将按上述流程再次进行 DNN 块的提交。若周期结束时尚有未完成任务, 系统将中断其执行并保存现场至 DRAM, 待下一时刻重新调度(行⑱)。

算法 2 开销主要包含 3 部分。首先是 DNN 划分, 基于广度优先搜索, 复杂度为 $O(V+E)$, 若 DNN 并非首次出现, 则可直接复用之前的划分结果。其次是队列排序, 复杂度为 $O(n\log n)$, n 为 RQ 长度, 该过程可与硬件执行重叠。最后是策略推理, 得益于并行架构, 推理开销仅为单次网络前向传播时间。相比之下, 采用 LSTM 串行推理的 RELMAS 算法, 其耗时与块数量正相关。本文在 Intel Xeon Gold 6334 CPU @ 3.60 GHz 与 NVIDIA H100 PCIe GPU 平台上进行实验, 算法平均调度耗时为 0.49 ms, 而 RELMAS 为 5.36 ms, 速度提升约 11 倍。划分算法的平均执行时间为 5.4 μs, 并且只在 DNN 模型首次出现时才有此开销, 能够适配实时调度场景。此外, 任务中断的现场保存耗时约 9.8 μs, 可与调度决策过程重叠。鉴于本文将调度周期设定为 5 ms, 上述开销在可接受范围内。

4 实验与结果分析

4.1 实验设置

本文基于 Timeloop 模拟器开发了支持多 DNN 并行执行的仿真平台, 该平台能够模拟多 Chiplet 加速器的互连通信与资源争用。硬件配置参考 SIMBA 架构[26], 具体参数如表 6 所示。如图 3 所示, 加速器包含 16 个 Chiplet, 通过 2D-mesh 网络互连(NoP), 采用 X-Y 路由。每个 Chiplet 内部集成 16 个 PE, 同样采用 2D-mesh 网络互连(NoC)。PE 采用权重静止数据流, 内部包含计算单元及独立的输入、权重与输出缓冲区。此外, 位于 2D-mesh 网络边缘的 4 个 Chiplet(C4、C7、C8、C11)分别连接一个 DRAM 作为片外内存。

Table 6 System Configurations

表 6 系统配置

模块	参数	取值
互连网络	Chiplet 数量	4×4
	NoP 带宽/GBps	100
	NoP 延迟/ns	20
	片外内存带宽/GBps	135
Chiplet	PE 数量	4×4
	时钟频率/GHz	1
	NoC 带宽/GBps	68
	NoC 延迟/ns	10
	L2 缓存大小/KB	256
PE	MAC 数量	64
	权重缓存/KB	32
	输入缓存/KB	8
	输出缓存/KB	3

为模拟真实负载, 本文参照文献 [37] 选取了视觉与语言领域代表性 DNN 模型, 并分别在加速器仿真平台上以不中断的方式独立执行各个模型, 得到每个模型的独立执行延迟, 选择延迟较大的模型形成重型负载, 延迟较小的形成轻型负载, 具体划分如表 7 所示。

Table 7 DNN Workloads

表 7 DNN 负载

负载	领域	模型
轻型	图像分类	MobileNetV3L, GoogleNet
	语言处理	Mobile-BERT
	图像分割	DeepLabV3
重型	目标检测	SSD-ResNet34, SSD-MobileNetV1
	语言处理	BERT-Large
	图像分割	UNet
混合	上述全部	上述全部

为验证本文算法的有效性, 实现并对比了以下基准算法: 1)FCFS-E, 以 DNN 块为粒度, 按请求到达顺序调度, 结合固定 Chiplet 优先级映射与固定 PE 分配数量的启发式策略; 2)EDF-E, 按截止时间顺序调度, 资源分配策略同 FCFS-E; 3)PREMA-E[32], 基于预估执行时间与等待时间的 Token 机制, 按最短作业优先(shortest job first, SJF)排序, 因其原先只面向单芯片加速器, 因此也结合了上述启发式策略进行多 Chiplet 加速器上的资源分配; 4)RELMAS[45]; 5)CD-MSA[35]。实验固定调度周期为 5 ms, 划分算法最大块长度限

制为 10，请求到达率 λ 为每秒到达 100 个请求。本文依据泊松过程为各负载生成推理请求的到达时间序列每次实验生成 1 000 个请求，每个请求对应的 DNN 模型则从负载集中随机选取。所有实验均独立重复运行 10 次，实验结果取平均值。

实验主要评价指标为请求的整体 SLA 满足率，即在截止时间前完成的请求比例。由于同一 DNN 在具有不同性能的加速器上的执行延迟也会不同，因此使用一个绝对的 QoS 约束可能不足以准确反映算法的效果。为公平起见，本文将 DNN 模型在加速器上独立执行延迟作为基准 QoS 时间，实际 QoS 约束定义为基准时间乘以一个 QoS 系数。例如，QoS 系数为 2 表示请求需在 2 倍独立执行延迟内完成，而 QoS 系数小于等于 1 的目标则是无法满足的。算法若能在更小的 QoS 系数下保持高 SLA 满足率，则表明性能更优。此外，本文还统计归一化尾延迟，即请求实际执行延迟与独立执行延迟的比值，用于衡量最差任务的完成效率。系统吞吐量采用指标 *STP*[60] 衡量，定义为：

$$STP=\sum_{i=1}^{n}NP_i,\quad NP_i=\frac{C_i^{\mathrm{SP}}}{C_i^{\mathrm{MP}}},\tag{26}$$

其中 C_i^{SP}、C_i^{MP} 分别表示任务 i 在单程序模式与多程序模式下的执行时间。*STP* 在理想无干扰环境下为任务数量 n，实际值越接近 n 表明并行效率越高。

4.2 PE 数量分配实验

为验证启发式算法在动态环境下的局限性，本节对比了 FCFS-E、EDF-E 及 PREMA-E 在不同 PE 数量分配策略下的性能，结果如图 7 所示。实验采用重型负载，设定 QoS 系数为 5，观测 SLA 满足率与尾延迟随 PE 数量的变化趋势。

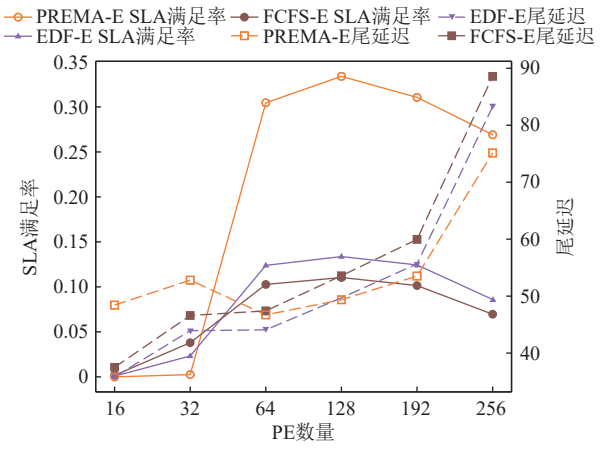


Fig. 7 Comparison of SLA satisfaction rate and tail latency with different number of PEs

图 7 不同 PE 数量下 SLA 满足率与尾延迟对比

实验结果表明，启发式算法的性能对 PE 分配数量高度敏感，存在特定的资源分配区间可使 SLA 满足率达到峰值。如图 7 所示，随着 PE 数量的增加，3 种算法的 SLA 满足率均呈现先升后降的趋势，且均在 PE 分配数量为 128 时取得最高 SLA 满足率。这与 DNN 任务执行的多种制约因素有关。虽然增加 PE 数量可为单个 DNN 任务提供更多计算能力，但 DNN 推理的加速比受限于任务本身的并行度上限，对于计算需求较低的 DNN 任务，过度分配计算资源无法带来等比例的性能提升，反而会造成资源浪费。此外，DNN 任务的执行时间还受到片间通信以及任务间的资源争用影响。将 DNN 计算分布到大量 PE 上会增加片间通信开销，并且在多任务并发场景下，为单个任务分配过多 PE 会减少可同时服务的任务数量，加剧系统资源争用，进而降低整体 SLA 满足率。因此，固定 PE 数量的分配策略存在最优值，但该最优值难以通过静态规则精确预判，且具有波动性。此外，性能指标间存在权衡关系。虽然 PE 数量为 128 时 SLA 满足率最优，但其尾延迟反而高于 PE 数量为 64 时的水平。进一步实验显示，轻型负载与混合负载下的最优 PE 分配数量与重型负载不同。这证实了这些启发式算法无法适应负载的动态变化，且难以在 SLA 满足率与尾延迟之间实现有效平衡。

4.3 QoS 指标

1）SLA 满足率

图 8 展示了在不同负载类型下，各算法随 QoS 系数变化的 SLA 满足率。为保证对比公平性，启发式算法均选取了其性能最优的 PE 分配方案（基于 4.2 节实验结果）。在轻型负载下（图 8(a)），所有算法均表现出较高的 SLA 满足率，表明低负载场景下资源竞争较弱，调度策略的差异对性能影响有限。在重型负载下（图 8(b)），MAOSA 展现出显著优势，在 QoS 系数为 10 时，较次优算法 PREMA-E 以及同样使用强化学习的 RELMAS 算法分别提升了 18%、25.5% 的 SLA 满足率。这说明在高资源竞争压力下，对调度顺序与资源分配的精度要求更高，传统启发式算法难以适应，而 MAOSA 通过强化学习自主学习到了更优策略。对比基于强化学习的 RELMAS 算法，MAOSA 优势主要包括支持抢占式调度，能更灵活地适应动态负载，避免了长任务阻塞；采用更细粒度（PE 级）的资源分配策略，解决了 RELMAS 因 Chiplet 级分配导致的资源利用率低下问题。在混合负载下（图 8(c)），MAOSA 与改进的 PREMA-E 算法性能相近，均优于其他基线算法。RELMAS 与 CD-MSA 算

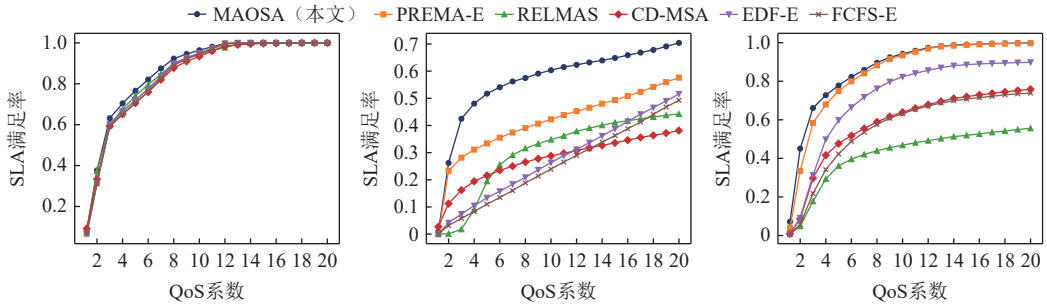


Fig. 8 Comparison of SLA satisfaction rates across algorithms with different workloads

图 8 不同负载下算法的 SLA 满足率对比

法表现较差, 原因在于混合负载增加了任务异构性, 而 RELMAS 缺乏抢占机制导致响应迟缓, CD-MSA 则受限于模型级调度粒度, 难以利用跨模型的块级并行机会, 在负载剧烈波动时容易产生性能瓶颈。

2)尾延迟

图 9 展示了各算法在不同负载下的 95% 尾延迟。MAOSA 在所有场景下均取得了最低的尾延迟, 尤其在重型负载下, 较次优算法 PREMA-E 以及 RELMAS 算法分别降低了 28.7%、60.6%。与 SLA 满足率结果一致, 轻型负载下各算法差距较小。而在重型与混合负载下, RELMAS 因不支持抢占, 长任务可能阻塞后续任务, 导致尾延迟显著增加; CD-MSA 则因模型级调度粒度过粗, 错失了块级并行机会, 同样导致部分任务完成时间延长。

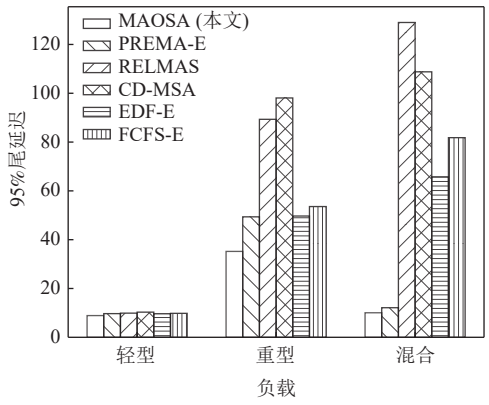


Fig. 9 Comparison of 95% tail latency across algorithms with different workloads

图 9 不同负载下算法的 95% 尾延迟对比

图 10 展示了各基准算法相对于 MAOSA 的归一化 STP。结果表明, MAOSA 在 3 种负载下均实现了最高的 *STP*, 较次优算法 PREMA-E 以及 RELMAS 算法最多可提升 31.9%、211.9%。*STP* 指标反映了多任务并行时的系统执行效率, 该结果证明 MAOSA 通过合理的调度与资源分配, 有效减少了资源争用与闲置, 最大程度地发挥了多 Chiplet 加速器的并行计算能力。

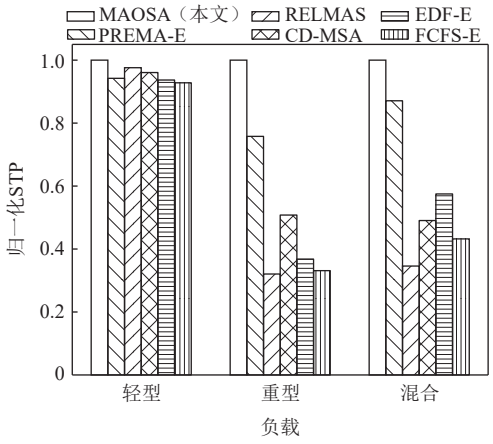


Fig. 10 Comparison of *STP* across algorithms with different workloads

图 10 不同负载下算法的 *STP* 对比

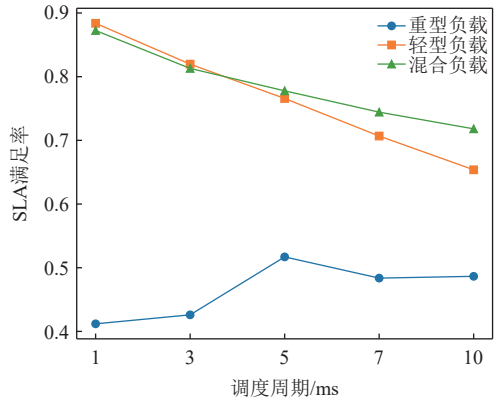
上述指标均为独立重复 10 次实验的平均值, 各组实验数据的相对标准差均在 4.85% 以内, 证明了本文实验的可靠性。综上所述, 本文提出的多智能体强化学习调度算法, 结合块级并行与周期性抢占机制, 在 3 种负载场景下均表现优异。相较于次优算法, 本文算法在重型负载下最高可提升 18% 的 SLA 满足率, 降低 28.7% 的尾延迟, 最高可使 *STP* 提升 31.9%, 验证了其在复杂动态环境下的有效性。

4.4 参数敏感性

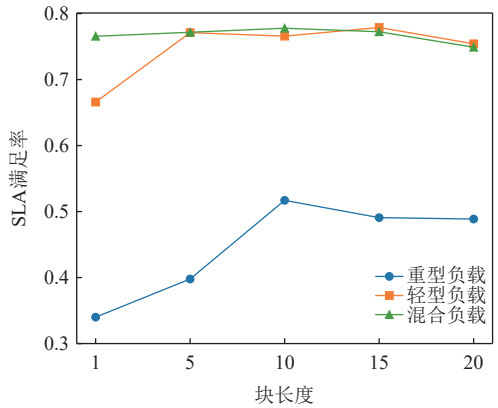
调度周期与 DNN 划分算法的块长度限制是影响系统性能的关键参数。为探究其影响, 本节在 3 种负载场景下(QoS 系数为 5 时), 对这 2 个参数进行了敏感性分析。

图 11(a)展示了固定最大块长度为 10 时, SLA 满足率随调度周期的变化趋势。在重型负载下, SLA 满足率随调度周期的增加呈先升后降趋势, 并在调度周期为 5 ms 时达到峰值。这是因为较短的调

度周期虽能提高响应频率，但会引入频繁的抢占中断与上下文切换开销，挤占了有效计算时间。较长的调度周期则导致调度决策滞后，无法及时捕捉系统负载的动态变化，致使部分紧迫任务无法被及时调度执行。而在轻型负载与混合负载下，SLA 满足率随调度周期的增加逐渐降低，在 1 ms 和 3 ms 的表现优于 5 ms 的。这是因为在负载压力较低的情况下，任务相对可以更快地完成执行，在中断时需保存现场的任务数量更少，并且单个任务中断保存的开销也低于重型负载。此时，短调度周期带来的动态响应收益超过了其引入的抢占开销。然而，调度周期的选取还需要考虑调度决策时间，越小的调度周期，决策开销占比越大，若采用 1 ms 的调度周期，调度决策时间占比将达到 49%，该开销已不可接受。因此，综合考虑 3 种负载在不同调度周期下的表现，本文最终选择 5 ms 作为调度周期。



(a) SLA满足率随不同调度周期的变化



(b) SLA满足率随不同块长度的变化

Fig. 11 Sensitivity experiments on scheduling period and block length

图 11 调度周期与块长度敏感性实验

图 11(b)展示了固定调度周期为 5 ms 时，SLA 满足率随块长度的变化趋势。在重型负载下，性能曲线呈现先升后降趋势，峰值位于块长度为 10 处。当块长度过小(如长度为 1，退化为层级调度)时，细粒度的任务划分导致层间数据交互频繁，中间结果需反复写回片外内存，降低了 L2 缓存的数据复用收益，从而增加了执行开销。反之，当块长度过大时，任务粒度变粗，限制了不同模型间的并行性，降低了系统资源利用率。轻型负载与混合负载的表现与重型负载类似，区别在于它们在最大块长度为 5、10、15 时的表现较为接近。最后，综合考虑 3 种负载的情况，本文将最大块长度设置为 10。

综合考虑数据局部性、并行潜力与调度开销，本文最终选取最大块长度为 10、调度周期为 5 ms 作为实验配置。

最后，本文进一步验证了强化学习超参数对系统性能的影响。在使用重型负载，QoS 系数为 5 的情况下，以表 5 中的超参数为基准，学习率分别为 1E−4、7E−4 时，SLA 满足率的变化均小于 2.5%；奖励折扣系数分别为 0.9、0.95 时，SLA 满足率的变化均小于 1.8%。上述结果表明本文算法在一个较宽的超参数范围内稳定有效。

4.5 消融实验

为了评估 MAOSA 各关键组件对性能提升的贡献，本文设计了消融实验，图 12 展示了 QoS 系数为 5 时，3 种负载场景下的 SLA 满足率。其中 w/o attention 代表移除注意力模块，改为对各 DNN 块的特征向量进行平均聚合，并将其与自身特征拼接后再输入策略网络；w/o slack 代表不使用奖励函数中的时间裕量奖励项；w/o preemption 代表周期性调度时不进行抢占，而是等待加速器上的任务执行结束后再提交任务执行；w/o PE_grad 代表不使用 PE 级资源分配粒度，改之以 Chiplet 为粒度。

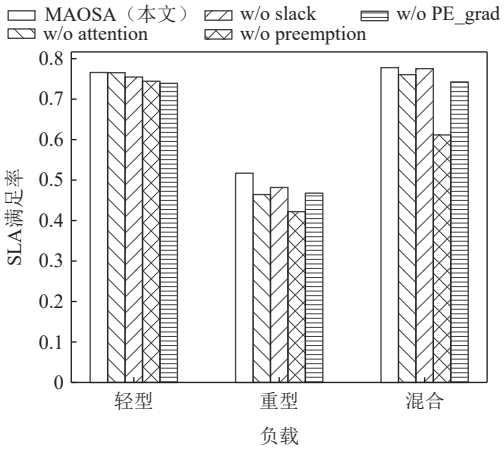


Fig. 12 Impact of removing key components on SLA satisfaction rate with different workloads

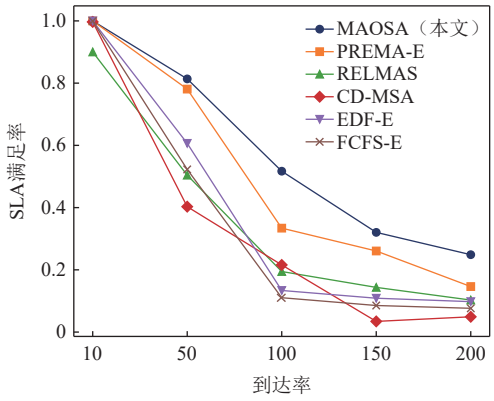
图 12 不同负载下移除关键组件对 SLA 满足率的影响

实验结果表明, 在 3 种负载下, 移除了关键组件后本文算法的性能均有不同程度的下降, 这证明了各组件设计的有效性。以重型负载为例, 上述 4 种变体的 SLA 满足率相较于完整算法分别下降了 5.3%、3.5%、9.5% 和 4.9%。轻型负载下算法的性能下降不如重型与混合负载, 这说明在负载压力较小的情况, 算法对复杂调度策略的依赖有所降低。重型负载下 w/o preemption 的影响小于混合负载, 这说明对于更加异构的混合任务, 抢占机制可发挥更重要的作用。相反, w/o attention、w/o slack 对重型负载的影响大于混合负载, 说明了在资源竞争更为激烈的场景下, 准确识别任务特征并充分利用时间裕量的重要性。最后, w/o PE_grad 在 3 种负载下均表现出可观的性能下降, 说明了 PE 级分配粒度较 Chiplet 粒度的优势。

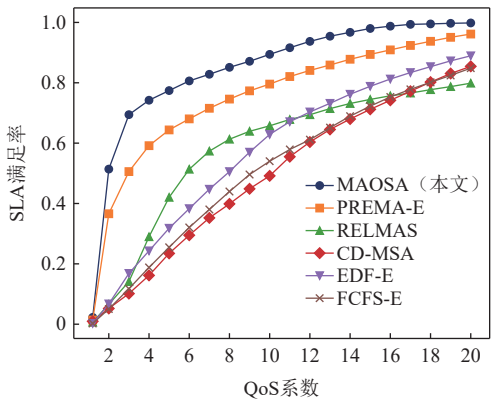
4.6 扩展性

为进一步验证 MAOSA 的可扩展性, 本节从请求并发度与系统规模 2 个维度进行测试, 实验均采用重型负载进行。

图 13(a)展示了 QoS 系数为 5 时, 各算法 SLA 满足率随请求到达率变化的情况。随着到达率从 10 增加至 200, 系统并发压力增加, 资源竞争加剧, 导致所有算法的 SLA 满足率均呈下降趋势。然而, MAOSA 始终保持着最佳性能, 且下降幅度明显小于基线算法。这说明在高并发场景下, MAOSA 能更有效地应对激烈的资源竞争, 展现出良好的负载适应能力。



(a) SLA满足率随到达率的变化



(b) 36个Chiplet加速器上各算法SLA满足率对比

Fig. 13 Scalability experiments

图 13 扩展性实验

图 13(b)展示了在扩展至具有 36 个 Chiplet(6×6 阵列)的加速器上, 请求到达率为 100 时的性能表现。得益于硬件资源的增加, 所有算法的性能均较 16 个 Chiplet 配置有所提升。其中, MAOSA 依然保持着显著的性能优势。这表明在面对更大规模的系统架构时, MAOSA 能够有效协调复杂的片间通信与计算资源, 具备良好的系统可扩展性。

5 结 论

本文面向服务器平台多 DNN 负载并行的应用需求, 结合推理服务的实时性与动态性特征, 提出了一种面向多 Chiplet 加速器的多智能体强化学习多 DNN 负载在线调度算法 MAOSA。算法以由 DNN 任务划分得到的 DNN 块为基本调度单元, 并将 DNN 块建模为智能体, 采用基于注意力机制的强化学习模型并行地为每个智能体生成调度策略, 并使用具有松弛性感知的奖励函数进一步提升 SLA 满足率。调度策略解耦为执行顺序与 PE 级资源分配 2 部分, 既考虑了请求的 QoS 时间约束也可充分发挥多 Chiplet 的并行潜力。最后, 采用周期性统一抢占机制平衡了调度灵活性与系统开销。仿真实验验证了本文算法在动态环境中的自适应学习能力, 不仅提升了系统整体 SLA 满足率, 还同步降低了尾延迟与调度算法执行开销, 且展现出良好的可扩展性, 为多 Chiplet 加速器场景下的多 DNN 在线调度提供了高效可行的解决方案。

本文目前主要关注整体 SLA 满足率的优化, 未来可进一步拓展至多目标优化场景, 例如增加功耗等考虑。此外, 当前调度决策主要关注对 Chiplet 内部计算资源(即 PE)的分配, 未来可延申至涵盖内存、带宽等资源的联合调度。最后, 可将算法从同构 Chiplet 扩展至异构 Chiplet 场景, 以适配更复杂的实际应用需求。

作者贡献声明: 王晨赳提出论文方案, 并负责实施实验和论文撰写; 张志锰参与仿真平台搭建与论文的修改; 郝沁汾对整体研究方向进行指导。

参 考 文 献

[1] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep residual learning for image recognition[C]//Proc of the 2016 IEEE Conf on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ: IEEE, 2016: 770−778

[2] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84−90

[3] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint, arXiv: 1409.1556, 2014

[4] Szegedy C, Liu Wei, Jia Yangqing, et al. Going deeper with convolutions[C]//Proc of the 2015 IEEE Conf on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ: IEEE, 2015: 1−9

[5] Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision[C]//Proc of the 2016 IEEE Conf on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ: IEEE, 2016: 2818−2826

[6] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//Proc of the 2014 IEEE Conf on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ: IEEE, 2014: 580−587

[7] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[J]. arXiv preprint, arXiv: 1506.02640, 2016

[8] Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks[J]. arXiv preprint, arXiv: 1409.3215, 2014

[9] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. arXiv preprint, arXiv: 1706.03762, 2017

[10] Oord A V D, Dieleman S, Zen H, et al. WaveNet: A generative model for raw audio[J]. arXiv preprint, arXiv: 1609.03499, 2016

[11] Bojarski M, Yeres P, Choromanska A, et al. Explaining how a deep neural network trained with end-to-end learning steers a car[J]. arXiv preprint, arXiv: 1704.07911, 2017

[12] Chen Xu, Jiao Lei, Li Wenzhong, et al. Efficient multi-user computation offloading for mobile-edge cloud computing[J]. IEEE/ACM Transactions on Networking, 2016, 24(5): 2795−2808

[13] Zhou Bowen, Dastjerdi A V, Calheiros R N, et al. An online algorithm for task offloading in heterogeneous mobile clouds[J]. ACM Transactions on Internet Technology, 2018, 18(2): 1−25

[14] Lin Li, Liao Xiaofei, Jin Hai, et al. Computation offloading toward edge computing[J]. Proceedings of the IEEE, 2019, 107(8): 1584−1607

[15] Romero F, Li Qian, Yadwadkar N J, et al. INFaaS: Automated model-less inference serving[C]//Proc of the 2021 USENIX Annual Technical Conf (USENIX ATC 21). Berkeley, CA: USENIX Association, 2021: 397–411

[16] Fowers J, Ovtcharov K, Papamichael M, et al. A configurable cloud-scale DNN processor for real-time AI[C]//Proc of the 2018 ACM/IEEE 45th Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2018: 1−14

[17] Gupta U, Hsia S, Saraph V, et al. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference[C]//Proc of the 2020 ACM/IEEE 47th Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2020: 982−995

[18] Jiao Yang, Han Liang, Jin Rong, et al. 7.2 A 12nm programmable convolution-efficient neural-processing-unit chip achieving 825TOPS [C]//Proc of the 2020 IEEE Int Solid-State Circuits Conference (ISSCC). Piscataway, NJ: IEEE, 2020: 136−140

[19] Jouppi N P, Yoon D H, Ashcraft M, et al. Ten lessons from three generations shaped Google's tpuv4i: Industrial product[C]//Proc of the 48th ACM/IEEE Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2021: 1−14

[20] Knowles S. Graphcore[C]//Proc of the 2021 IEEE Hot Chips 33 Symposium (HCS). Piscataway, NJ: IEEE, 2021: 1–25

[21] Vasiljevic J, Bajic L, Capalija D, et al. Compute substrate for software 2.0[J]. IEEE Micro, 2021, 41(2): 50−55

[22] Lauterbach G. The path to successful wafer-scale integration: The cerebras story[J]. IEEE Micro, 2021, 41(6): 52−57

[23] Moore G E. Cramming more components onto integrated circuits[J]. Proceedings of the IEEE, 1998, 86(1): 82−85

[24] Ascia G, Catania V, Mineo A, et al. Improving inference latency and energy of DNNs through wireless enabled multi-chip-module-based architectures and model parameters compression[C]//Proc of the 2020 14th IEEE/ACM Int Symp on Networks-on-Chip (NOCS). Piscataway, NJ: IEEE, 2020: 1–6

[25] Hwang R, Kim T, Kwon Y, et al. Centaur: Achiplet-based, hybrid sparse-dense accelerator for personalized recommendations[C]//Proc of the 2020 ACM/IEEE 47th Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2020: 968–981

[26] Shao Y S, Clemons J, Venkatesan R, et al. SIMBA: Scaling deep-learning inference with multi-chip-module based architecture[C]// Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2019: 14–27

[27] Lie S. Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning[J]. IEEE Micro, 2023, 43(3): 18−30

[28] Talpes E, Sarma D D, Williams D, et al. The microarchitecture of DOJO, tesla's exa-scale computer[J]. IEEE Micro, 2023, 43(3): 31−39

[29] Dong Wenkuo, Yin Chunsuo, Zhang Zhimeng, et al. Yingtian-Lake: A wafer-scale general-purpose heterogeneous multi-chiplet petascale computer[J]. Journal of Computer Research and Development, 2025, 62(6): 1492−1512 (in Chinese)
(董文阔, 殷春锁, 张志锰, 等. 映天湖: 晶圆级通用异构多芯粒千万亿次计算机[J]. 计算机研究与发展, 2025, 62(6): 1492−1512)

[30] Baek E, Kwon D, Kim J. A multi-neural network acceleration architecture[C]//Proc of the 2020 ACM/IEEE 47th Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2020: 940−953

[31] Heidari S, Ghasemi M, Vrudhula S. Uncertainty-aware RL-based scheduling of multi-DNN workloads on edge mpsocs[C]//Proc of the 10th ACM/IEEE Symp on Edge Computing. New York: ACM, 2025: 1–16

[32] Li Yihong, Zhang Xiaoxi, Zeng Tianyu, et al. Task placement and resource allocation for edge machine learning: A GNN-based multi-agent reinforcement learning paradigm[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(12): 3073−3089

[33] Feng Yiming, Hu Shihong, Chen Lingqiang, et al. An intelligent scheduling framework for DNN task acceleration in heterogeneous edge networks[J]. Computer Communications, 2023, 201: 91−101

[34] Reddi V J, Cheng C, Kanter D, et al. MLPerf inference benchmark[C]//Proc of the 2020 ACM/IEEE 47th Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2020: 446−459

[35] Wang Chunyang, Bai Yuebin, Sun Desen. CD-MSA: Cooperative and deadline-aware scheduling for efficient multi-tenancy on DNN accelerators[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(7): 2091−2106

[36] Kao S C, Krishna T. MAGMA: An optimization framework for mapping multiple dnns on multiple accelerator cores[C]//Proc of the 2022 IEEE Int Symp on High-Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2022: 814−830

[37] Das A, Russo E, Palesi M. Multi-objective hardware-mapping co-optimisation for multi-DNN workloads on chiplet-based accelerators[J]. IEEE Transactions on Computers, 2024, 73(8): 1883−1898

[38] Zhang Jinming, Wang Xuyan, Ye Yaoyao, et al. M2M: A fine-grained mapping framework to accelerate multiple DNNs on a multi-chiplet architecture[J]. IEEE Transactions on Very Large Scale Integration Systems, 2024, 32(10): 1864−1877

[39] Choi Y, Rhu M. PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units[C]//Proc of the 2020 IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2020: 220−233

[40] Choi Y, Kim Y, Rhu M. LazyBatching: An sla-aware batching system for cloud machine learning inference[C]//Proc of the 2021 IEEE Int Symp on High-Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2021: 493−506

[41] Oh Y H, Kim S, Jin Y, et al. LayerWeaver: Maximizing resource utilization of neural processing units via layer-wise scheduling[C]// Proc of the 2021 IEEE Int Symp on High-Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2021: 584−597

[42] Wang Fuyu, Shen Minghua, Lu Yutong, et al. Ceiba: An efficient and scalable DNN scheduler for spatial accelerators[J]. ACM Transactions on Architecture and Code Optimization, 2025, 22(2): 1−23

[43] Li Jialun, Xiao Danyang, Yang Diying, et al. Integrated and fungible scheduling of deep learning workloads using multi-agent reinforcement learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2025, 36(3): 391−406

[44] Sun Jinghan, Reidys B, Li Daixuan, et al. FleetIO: Managing multi-tenant cloud storage with multi-agent reinforcement learning[C]//Proc of the 30th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS). New York: ACM, 2025: 478–492

[45] Blanco F G, Russo E, Palesi M, et al. A deep reinforcement learning based online scheduling policy for deep neural network multi-tenant multi-accelerator systems[C]//Proc of the 61st ACM/IEEE Design Automation Conf (DAC). New York: ACM, 2024: 1–6

[46] Basaklar T, Goksoy A A, Krishnakumar A, et al. DTRL: Decision tree-based multi-objective reinforcement learning for runtime task scheduling in domain-specific system-on-chips[J]. ACM Transactions on Embedded Computing Systems, 2023, 22(5): 1−22

[47] Kanani A, Pfromm L, Sharma H, et al. THERMOS: Thermally-aware multi-objective scheduling of ai workloads on heterogeneous multi-chiplet pim architectures[J]. ACM Transactions on Embedded Computing Systems, 2025, 24(5): 1−26

[48] Littman M L. Markov games as a framework for multiagent reinforcement learning[C]//Proc of the Machine Learning Proceedings 1994. Amsterdam: Elsevier, 1994: 157–163

[49] Ding Shifei, Du Wei, Zhang Jian, et al. Research progress of multi-agent deep reinforcement learning[J]. Chinese Journal of Computers, 2024, 47(7): 1547−1567 (in Chinese)
(丁世飞, 杜威, 张健, 等. 多智能体深度强化学习研究进展[J]. 计算机学报, 2024, 47(7): 1547−1567)

[50] Ghodrati S, Ahn B H, Kim J K, et al. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks[C]//Proc of the 2020 53rd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2020: 681−697

[51] Choi J, Ha Y, Lee J, et al. Enabling fine-grained spatial multitasking on systolic-array NPUS using dataflow mirroring[J]. IEEE Transactions on Computers, 2023, 72(12): 3383−3398

[52] Liu Yafei, Li Xiangyu, Yin Shouyi. Review of chiplet-based design: System architecture and interconnection[J]. Science China Information Sciences, 2024, 67(10): 200401

[53] Tan Zhanhong, Cai Hongyu, Dong Runpei, et al. NN-Baton: DNN workload orchestration and chiplet granularity exploration for multichip accelerators[C]//Proc of the 48th 2021 ACM/IEEE Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2021: 1013−1026

[54] Jang H, Ha D, Lee H, et al. DeSpa: Heterogeneous multi-core accelerators for energy-efficient dense and sparse computation at the tile level in deep neural networks[J]. Journal of Systems Architecture, 2026, 172: 103650

[55] Parashar A, Raina P, Shao Y S, et al. Timeloop: A systematic approach to DNN accelerator evaluation[C]//Proc of the 2019 IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2019: 304–315

[56] Kao S C, Krishna T. GAMMA: Automating the HW mapping of dnn models on accelerators via genetic algorithm[C]//Proc of the 2020 IEEE/ACM Int Conf On Computer Aided Design (ICCAD). Piscataway, NJ: IEEE, 2020: 1–9

[57] Chu Xiangxiang, Ye Hangjun. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning[J]. arXiv preprint, arXiv: 1710.00336, 2017

[58] Kim S, Genc H, Nikiforov V V, et al. Moca: Memory-centric, adaptive

execution for multi-tenant deep neural networks[C]//Proc of the 2023 IEEE Int Symp on High-Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2023: 828−841

[59] Chao Yu, Velu A, Vinitsky E, et al. The surprising effectiveness of PPO in cooperative, multi-agent games[J]. arXiv preprint, arXiv: 2103.01955, 2022

[60] Eyerman S, Eeckhout L. System-level performance metrics for multiprogram workloads[J]. IEEE Micro, 2008, 28(3): 42−53



Wang Chenjiu, born in 1998. PhD candidate. Student member of CCF. His main research interests include deep learning accelerators and wafer-scale systems.

王晨赳, 1998 年生。博士研究生。CCF 学生会员。主要研究方向为深度学习加速器、晶圆级系统。



Zhang Zhimeng, born in 1999. PhD candidate. His main research interests include neural networks and wafer-scale systems.

张志锰, 1999 年生。博士研究生。主要研究方向为神经网络、晶圆级系统。



Hao Qinfen, born in 1969. PhD, professor. Distinguished member of CCF, senior member of IEEE. His main research interests include computer architecture, Chiplet-based processor design, and wafer-scale computing systems.

郝沁汾, 1969 年生。博士，教授。CCF 杰出会员，IEEE 高级会员。主要研究方向为计算机体系结构、基于 Chiplet 的处理器设计、晶圆级计算系统。