可重构 CNN 处理器的高效能自适应映射策略

常立博[1,2)], 武丹妮[1)], 杜慧敏[1)*], 张盛兵[2)], 郝鹏[2)], 蔡秀霞[1)]

[1)] (西安邮电大学电子工程学院 西安 710121)
[2)] (西北工业大学计算机学院 西安 710072)
(fv@xupt.edu.cn)

摘 要: 针对因 CNN 模型巨大的参数规模和数据访问量, 及不同 CNN 模型或同一模型中不同层的计算模式存在多样性, 导致其仅可支持单一或固定映射方式及片上数据缓存的计算系统整体效能较低的问题, 提出软硬件协同设计策略. 将基于深度强化学习的自适应访存优化机制与片上弹性缓存动态划分方法结合, 根据可重构 CNN 处理器中存储结构相关参数, 针对不同 CNN 运算层自动搜索最优的循环调度策略; 并通过设计可重构片上互联结构、地址映射逻辑以及动态存储调度方法, 使片上弹性缓存可根据不同调度策略动态划分地址映射空间. 在基于 Eyeriss 和 TPU 的 CNN 处理器构架上, 与对比的调度策略相比, 采用所提方法可分别将 2 种可重构 CNN 处理器的效能提升约 3 倍和 4 倍; 并且, 采用相同的调度策略, 与固定容量双缓存结构相比, 文中的弹性存储划分方法可分别将功耗减少 30.28% 和 18.43%. 与基于 FPGA 平台的相关研究相比, 文中可重构 CNN 处理器将计算效率和计算效能分别提高了约 10 倍和 2 倍.

关键词: 卷积神经网络模型映射器; 卷积神经网络模型数据流; 深度强化学习; 片上弹性缓存
中图法分类号: TP391.41 **DOI:** 10.3724/SP.J.1089.2023-00433

Adaptive Mapping for Data Orchestration in Reconfigurable CNN Processors via Deep Reinforcement Learning

Chang Libo[1,2)], Wu Danni[1)], Du Huimin[1)*], Zhang Shengbing[2)], Hao Peng[2)], and Cai Xiuxia[1)]

[1)] (*School of Electronic Engineering, Xi'an University of Posts and Telecommunication, Xi'an* 710121)
[2)] (*School of Computer Science, Northwestern Polytechnical University, Xi'an* 710072)

Abstract: The huge parameter scale and data access of existing CNN models, as well as the diversity of computing modes of different CNN models or different layers in the same model, lead to the low overall computing efficiency of computing systems that can only support a single or fixed mapping mode and on-chip data cache. Using the strategy of hardware and software coordination design, we combine the adaptive memory access optimization mechanism based on deep reinforcement learning with the dynamic partition method of on-chip elastic cache, and automatically searches the optimal cyclic scheduling strategy for different CNN operation layers according to the relevant parameters of the storage structure in the reconfigurable CNN processor. By designing the on-chip interconnection structure, address mapping logic and dynamic partition method, elastic storage can dynamically partition the address mapping space according to different scheduling strategies during

收稿日期: 2023-07-14; 修回日期: 2024-04-01. **基金项目**: 国家重点研发计划(2019YFB1803600); 陕西省自然科学基础研究计划(2025JC-YBMS-736); 陕西省教育厅重点科学研究计划(22JY059); 中国民航适航中心开放基金(SH2021111903). **常立博**(1985—), 男, 博士, 讲师, CCF 会员, 主要研究方向为先进微处理器体系结构设计、深度神经网络加速器设计; **武丹妮**(2001—), 女, 硕士研究生, 主要研究方向为先进微处理器体系结构设计; **杜慧敏**(1966—), 女, 博士, 教授, 硕士生导师, CCF 会员, 论文通信作者, 主要研究方向为先进微处理器体系结构设计; **张盛兵**(1967—), 男, 博士, 教授, 博士生导师, CCF 会员, 主要研究方向为先进微处理器体系结构设计; **郝鹏**(1984—), 男, 博士研究生, CCF 会员, 主要研究方向为先进微处理器体系结构设计、5G 通信芯片设计; **蔡秀霞**(1987—), 女, 博士, CCF 会员, 主要研究方向为基于深度神经网络的智能算法研究.

accelerator operation. In the CNN processor architecture based on Eyeriss and TPU architectures, compared with the optimal scheduling strategy, the proposed scheduling strategy and on-chip cache partitioning method can improve the energy efficiency of the two reconfigurable CNN processors by almost 3 times and 4 times respectively. Moreover, compared with the fixed capacity dual cache structure, the elastic storage partitioning method proposed in this paper can reduce the power consumption by 30.28% and 18.43%, respectively, using the same scheduling strategy. Compared with the latest research based on FPGA platform, the reconfigurable CNN processor in this paper can improve the computational efficiency and computational energy efficiency by almost 10 times and 2 times respectively.

Key words: mapping of CNN; dataflow of CNN; deep reinforcement learning; elastic on-chip buffer

随着半导体工艺逼近物理极限, 集成电路的计算和存储能力提升速度逐渐放缓. 然而随着以卷积神经网络(convolutional neural network, CNN)模型为代表的人工智能应用的日益增长, 主要依靠集成电路工艺技术来提高计算能力的传统方法正逐步失效. 与此同时, 由于冯·诺依曼体系结构固有的“存储器墙”问题[1], 以及其对应的数据访存产生的功耗和增加的计算延时, 远远超过了计算单元完成对应计算任务产生的功耗和增加的计算延时[2], 如在 20 nm 的 CMOS 工艺下, 片外访存的功耗及延迟是数据运算的 800 多倍[3]. 因此, 在具有复杂数据访问的 CNN 处理器中, 通常会因未能对各级存储资源进行合理分配和优化, 以及远离计算单元的数据访问数量过多, 从而产生更多的功耗的计算延迟, 导致智能计算平台的效能无法满足终端系统要求.

已有基于软硬件协同设计方法[4-7]采用可支持灵活数据重用模式的可重构 CNN 处理器架构, 通过设计可适应 CNN 模型计算特性的片上数据缓存器结构[8-9]和数据访问策略[10], 提高片上数据复用率, 以降低数据移动代价. 虽然 CNN 计算任务之间具有较高的数据局部性, 但采用未经充分优化的分块和映射方法并不利于实现数据重用, 往往会增加数据在计算单元和存储单元之间的移动次数, 从而影响 CNN 处理器的整体性能和功耗[11]. 已有采用基于循环枚举的策略可在理论上找到最优的映射方法, 但是需要花费更多的搜索时间和极大的计算代价, 导致其很难满足实际应用需求. Yang 等[12]提出的优化方法加快了优化模型的收敛, 但是对于规模更大的 CNN 以及调度情况更为复杂的可重构 CNN 处理器, 此类基于枚举的方法依然无法达到很高的计算效率. 因此, 需要设计一种自动并且高效的 CNN 最优数据流模式搜索策略.

与此同时, 已有研究通常假设 CNN 处理器中缓存单元采用固定结构, 如双缓存(Buffer)、便笺存储器(scratchpads)和高速缓冲区存储器(cache)等[13]. 因此, 虽然可通过优化数据访问策略的方法提高片上数据复用率, 进而降低数据访问代价及计算延迟, 但由于未将映射策略和片上存储单元划分方法相结合, 仅通过权衡针对不同参数的缓存区容量来适应大多数应用情况. 因此, 对于优化循环调度后, 所需的存储量大于分配存储量的运算层, 使存储容量不能满足计算块内运算的数据缓存需求, 只能通过调整循环调度策略将部分数据缓存于片外存储单元, 从而降低了处理器的计算性能. 同时, 对于所需存储量小于分配存储量的运算层, 会因分配的存储量较大, 相应的单次访问功耗也随之增大, 从而降低了处理器的计算效能.

为了解决因固定映射策略和片上数据缓存模式导致较大数据访问产生的延迟和功耗, 进而降低可重构 CNN 处理器计算效能的问题, 本文基于深度强化学习(deep reinforcement learning, DRL)算法[14], 提出了一种以减少数据访存代价为目标的自适应优化模型. 该模型根据处理器的存储结构以及 CNN 运算层的特性, 自动搜索最优循环调度策略; 并且, 在已有弹性存储结构[10,15]的基础上, 提出可根据循环调度策略动态划分存储区并调整访存模式的数据缓存方法. 本文从设计方法、映射策略和缓存划分机制 3 个角度出发, 提出一套有效地提高 CNN 处理器中数据访问效率的方法, 以提高智能计算平台的整体效能, 具体如下.

(1) 提出一种软硬件协同优化的 CNN 处理器设计方法, 通过量化分析 CNN 模型计算过程中基于循环调度的数据流特性, 确定 CNN 处理器中各级存储器的容量和结构、处理器性能和功耗的量化关系, 从而建立一种将 CNN 计算任务到可重构处

理的可自适应优化映射模型.

(2) 在基于量化分析的映射模型根据上, 提出一种基于深度强化学习的自适应数据访存机制优化策略, 针对不同 CNN 的计算和访存特征以及处理器的片上缓存单元架构, 通过构建可平衡处理器性能和功耗的奖励函数, 获得一种高效能的计算任务映射和数据访问方法.

(3) 为适应自适应数据访存机制, 提出一种基于可重构片上互联结构和逻辑地址映射的动态划分方法, 通过灵活的动态划分片上缓存单元, 并针对不同的循环调度策略自动地调整数据缓存模式, 以减少片外数据访问数量并降低数据移动的代价.

1　相关工作和研究动机

1.1　面向 CNN 运算的功耗和性能模型

CNN 模型中最主要的卷积层运算通常可采用基于多层循环的计算模式表示[3], 因此可以通过分析各种基于循环的数据流特性确定处理器片上的数据流和计算模式, 进而确定 CNN 处理器中各级存储器的容量和结构, 以及处理器性能和功耗的量化关系. 假设 CNN 处理器的存储单元由 L 级缓存单元组成, 其中最高级存储单元(0 级)为最接近计算单元(processing element, PE)的寄存器, 最低级存储单元($L-1$ 级)为片外存储单元——DRAM. 采用各级数据存储量的计算方法[16], 即

$$s_i = \prod_{d\in D_{\dim}} \prod_{j=0}^{i} d_j \tag{1}$$

以确定各级存储中不同类型的数据块的尺寸. 其中, d_j 为 j 级缓存的维度子集, 表示该缓存级别中相应维度的数据量; $D_{\dim}$ 为所有维度的集合. 依照各级缓存所需的存储量, 即可得到相邻缓存级别所需要的带宽. 由于每级缓存都使用双 buffer 的结构, 因此需要将平均带宽作为衡量性能的标准, 而缓存级别的平均带宽为当前缓存级别下各类数据的带宽总和. 假设当前缓存级别中存储的数据被所有循环重用后才会被替换, 那么 i 级缓存的平均带宽可表示为

$$w_i^{\mathrm{b}} = s_i / c_i \tag{2}$$

其中, s_i 为第 i 级缓存所需存储的数据量; c_i 为完成 s_i 对应计算任务所需的运算时钟周期数, 其计算公式为

$$c_i = o_i \Big/ \prod_{u^l\in U^L} \prod_{j=0}^{i-1} u_j^l \tag{3}$$

即为理论上完成 c_i 表示的运算时钟周期与处理器的并行度的比值. 其中, o_i 表示 s_i 对应计算任务在理论上需要的计算量; $\prod_{u^l\in U^L} \prod_{j=0}^{i-1} u_j^l$ 表示计算单元可提供的并行处理计算能力的总和; U^L 为针对 s_i 的循环展开空间; 而 u_j^l 表示在共有 L 级缓存单元的处理器中, 针对第 l 级的第 j 层缓存单元中数据的并行计算度.

本文采用了类似于 Eyeriss 的功耗分析模型, 即当前缓存级别的总功耗为访存次数 A_i^{acc} 与每次访问功耗 e_i 的乘积, 公式为

$$E = \sum_{i=0}^{L-1} A_i^{\mathrm{acc}} \times e_i \tag{4}$$

其中, $A_i^{\mathrm{acc}} = \prod_{j=i}^{L-1} R_j^{\mathrm{if}} + \prod_{j=i}^{L-1} R_j^{\mathrm{of}} + \prod_{j=i}^{L-1} R_j^{\mathrm{w}}$.

因为数据移动方向是从高级别缓存到低级别缓存, 所以 i 级缓存的访存次数 A_i^{acc} 取决于 i 级以外 $(i, i+1, \cdots, L-1)$ 缓存级别的访存次数. 其中, R_j^{if}, R_j^{of} 和 R_j^{w} 分别表示输入特征图、输出特征图和权重在第 j 级缓存中的重用次数. 假设某种数据从加载到第 i 级缓存到被替换的时间为 T, 则在 T 中第 $i-1$ 级缓存对此数据的访问次数可定义为数据重用率 R_i. 因此, 当调度策略确定时, 在第 i 级缓中不同类型数据的重用率[16]可表示为

$$R_i = \begin{cases} \prod_{v\in V} v_i, \text{ 对于输入特性和权重} \\ 2\sum_{v\in V} v_i, \text{对于输出特性} \end{cases} \tag{5}$$

其中, V 表示各种循环的最大值. 各级缓存中输入特征、输出特征以及权重重用次数分别为

$$R^{\mathrm{if}} = F_X F_Y \prod_{j=i}^{L-1} k_j \tag{6}$$

$$R^{\mathrm{of}} = 2F_X F_Y \prod_{j=i}^{L-1} c_j \tag{7}$$

$$R^{\mathrm{w}} = \prod_{j=i}^{L-1} x_j \prod_{j=i}^{L-1} y_j \prod_{j=i}^{L-1} b_j \tag{8}$$

由于常用数据流并不会对输入特征的行方向(f_x)及列方向(f_y)的循环次数进行拆分, 因此这 2 个维度的重用次数为卷积核在行方向和列方向上的尺寸, 即为 F_X 和 F_Y. 与文献[16]对于 PE 间的数据移动次数和对应功耗模型描述类似, 本文也采用与式(4)所示类似方法, 完成对 PE 间数据访问的

功耗的建模.

1.2 面向 CNN 处理器的映射策略及研究动机

在面向 CNN 模型的计算系统中, 映射是将 CNN 计算所需的每个操作(包括提供适当的操作数)分配并调度到处理器的硬件单元, 以完成运算的过程. 由于映射会对计算单元的利用率和数据访问数量等产生较大的影响, 因此优化映射策略对于提高 CNN 计算系统的效能和性能具有重要意义[17]. 基于类似于屋顶线模型分析方法, Zhang 等[18]优化了 $L-1$ 级缓存的性能和功耗, 却忽略了片上其他缓存级别对于存储结构性能和功耗的影响. Suda 等[19]以最小化推理时间为优化策略, 利用回归模型来预测各级缓存的最优化配置, 但是没有关注各级缓存对功耗的影响; Yang 等[12]提出了一种基于循环枚举的优化框架, 将功耗和性能模型作为反馈输入, 通过遍历循环拆分、循环排序以及循环展开方法的组合, 以获得最优的循环调度方法; 但是由于 CNN 巨大的循环调度空间, 其遍历过程可能需要很长时间才能够收敛. 以 AlexNet[20]中第 3 个卷积运算层的输入通道维度为例, 其包含 192 个输入通道, 那么 0 级缓存中对于此维度的拆分方法共有 192 种, 同时根据文献[12]的分析可知, 各缓存级别中循环的重排列顺序有 7!种, 假设待优化处理器为一个 3 级存储结构, 那么对于输入通道维度, 分析模型需要遍历 $(192\times 7!)^3$ 种组合; 另外, 若处理器的计算分块可支持并行运算, 循环展开维度会继续扩大遍历的次数. 虽然通过过滤方法[12]可加快优化模型的收敛, 但是对于规模更大的 CNN 以及需要用更多配置信息表示不同调度方式的可重构 CNN 处理器, 枚举法依然无法达到很高的计算效率.

虽然可重构 CNN 处理器具有很强的可扩展性和可定制性[15], 可根据被加速的 CNN 模型的数据流特性, 通过动态重构片上缓存单元架构和互联模式, 以及数据访问模型, 以提高计算效率; 但是采用确定的片上缓存资源划分模式和数据重用策略, 仍无法充分地利用片上缓存资源[15]. 针对 ResNet50 模型, 采用 Eyeriss 架构的 CNN 处理器, 其前 12 层需要的块随机存取存储器(block ramdom access memory, BRAM)数量如图 1 所示. 因为 ResNet50 由多个相似的卷积块组成, 所以其他层的情况与前 12 层类似. 从统计结果可以看出, 不仅各卷积层的参数总量具有较大差异, 而且同层不同类型的参数存储量也差异巨大. 如采用传统的双缓存片上缓存结构, 共需要 290 个存储块(假设一个存储块的大小为 36 Kbit). 因此如果缓存区不能根据映射策略以动态调整缓存模式, 则会导致浪费片上缓存资源或降低计算效率. 为了提高处理器计算效率并降低功耗, 以满足嵌入式平台的应用要求, 本文在文献[12]的优化框架上, 提出结合基于智能算法的自适应映射策略和可重构弹性缓存单元划分的 CNN 处理器.

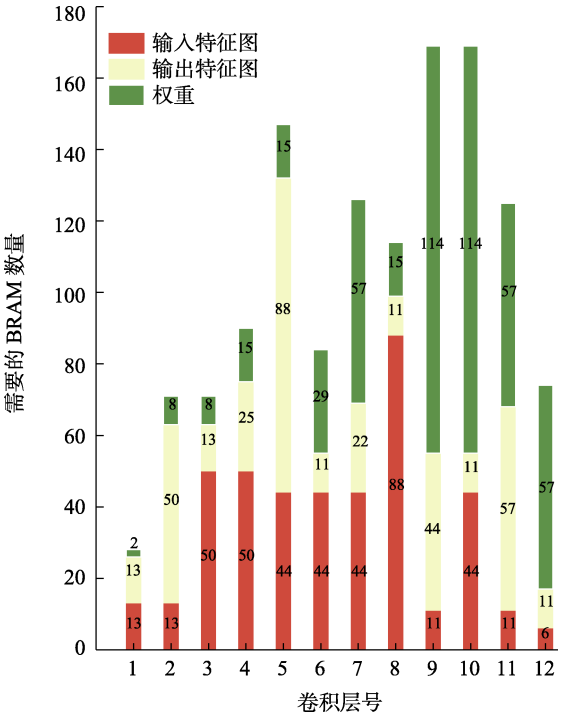


图 1 基于 Eyeriss 架构的 ResNet50 模型前 12 层卷积层中各类数据需要的存储容量

2 本文系统整体结构

为了降低可重构 CNN 处理器中访问数据的延迟和功耗, 本文基于 DRL 的智能优化算法, 提出一种可自动搜索最优循环调度方法和片上缓存资源划分策略, 通过配置相应的片上存储结构和互连模式, 以提高片上数据的重用率. 本文系统结构如图 2 所示, 采用智能算法[21]自动搜索最优的循环调度策略, 映射模块根据确定的循环变换组合实现 CNN 处理器的数据流以及存储配置, 存储结构的功耗及性能分析结果将反馈给智能搜索模型, 智能搜索模型则根据反馈信息更新优化策略. 同时, 片上弹性缓存单元根据调度策略动态地划分片上存储资源, 并通过配置可重构互连网络来适应不同的调度策略.

当计算平台的运算单元和存储结构固定, 并

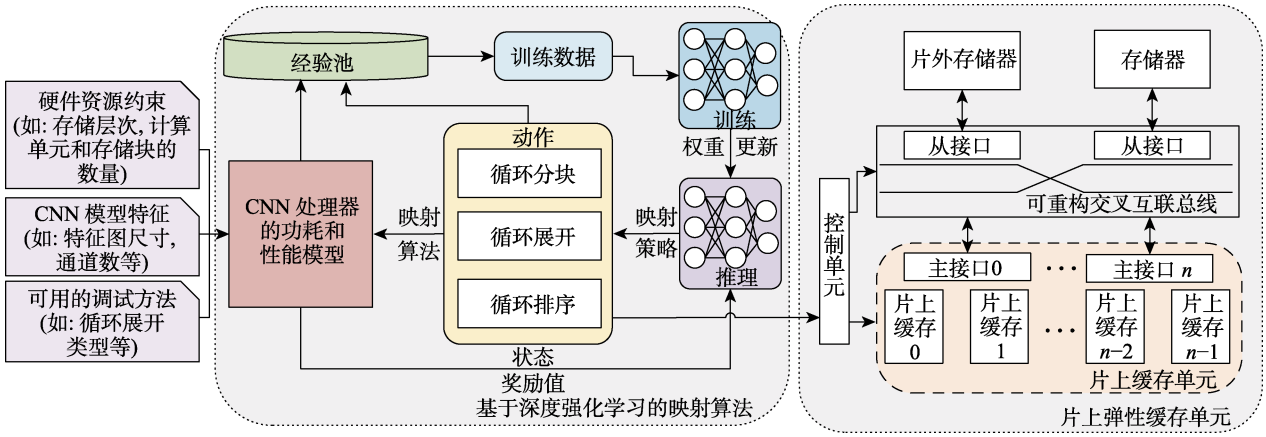


图 2　基于 DRL 和片上弹性缓存的自适应映射优化模型

且 CNN 模型的计算任务确定, 则采用第 1.1 节的分析方法估算针对特定映射策略的功耗和计算效率. 因此, 本文在训练基于 DRL 的自适应映射算法时, 将功耗和性能模型作为训练的“环境”. 同时, 本文采用基于优先经验回放的数据采样方法, 以及基于双网络架构的深度强化学习算法, 以提高训练的效率和稳定性.

3　基于深度强化学习的访存机制优化模型

本文将面向可重构 CNN 处理器的模型加速任务映射问题建模为一个基于强化学习问题, 通过自动搜索循环调度方法, 得到可重构 CNN 处理器中最优的访存机制以及相应的存储结构配置, 如图 2 所示的映射算法. 本文采用基于 Actor-Critic 框架的算法[22]作为智能体, 通过基于第 1.1 节中功耗和性能模型设计的 CNN 处理器模型交互, 自动探索到最优的循环拆分、循环排序以及循环展开策略.

3.1　奖励函数

由于可重构 CNN 处理器数据访问的效能由访问存储的延迟和功耗共同决定, 因此智能体的奖励函数应同时考虑延迟和功耗.

如式(3)所示, 由于运算周期 c_i 决定了第 i 级缓存的访问延迟, 访问延迟与第 i 级缓存的循环展度和运算量 o_i 相关, 然而循环拆分方法既影响第 i 级缓存的数据量 s_i 又影响运算量 o_i. 因此, 为了消除因各优化变量之间相互影响导致搜索方法无法收敛问题, 无法将运算周期 c_i 直接作为智能体的奖励函数. 另外, 数据访问的功耗也与第 i 级缓存的数据量 s_i 有关, 因此为了提高优化模型的工作效率, 同时解耦估算数据访问延迟和功耗中各参数量之间的相关性, 本文使用计算资源利用率 p 表示 CNN 处理器计算性能, 即

$$p=\prod_{u^l\in U^L}\prod_{j=0}^{i-1}u_j^l\Big/\mathrm{MAC}_i \tag{9}$$

其中, MAC_i 表示第 i 级缓存内 $(i,i-1,\cdots,0)$ 乘加单元的数量; 其分子项表示第 i 级缓存内各级可以并行处理的乘加运算量乘积; 并采用如式(4)所示的功耗模型表示访问数据的功耗. 因此, 在式(9)定义的计算资源利用率基础上, 本文智能搜索策略中的奖励函数定义为

$$R(s,a)=p(s,a)-\lambda E(s,a) \tag{10}$$

其中, s 表示状态; a 表示动作; λ 为超参数.

3.2　动作集合

DRL 的动态集合即为 CNN 的循环调度空间, 根据文献[12], 循环拆分以及循环展开共用同一数据维度 D_{dim}. 两者的不同点在于对批处理大小 B 的处理, 循环拆分不涉及对于 B 的拆分; 而循环展开可以应用可重构数据流的复制方法并行计算多张特征图. 对于特征数据, D_{dim} 维度下包括特征图尺寸 X 和 Y, 批处理大小 B, 以及输入通道数 C 或者输出通道数 K; 对于权重, D_{dim} 维度下包括卷积核尺寸 F_X 和 F_Y, 以及对应通道数. 表 1 给出了 L 级存储结构中循环拆分、循环排序以及循环展

表 1　L 级存储结构中动作集合

循环调度策略	可能的动作数量
循环拆分	$\prod_{i=0}^{L-1}\prod_{d\in D_{\mathrm{dim}}}d_i$
循环展开	$C_7^2\prod_{j=0}^{L}U_j^L$
循环排序	$(7!)^L$

开的动作集合.

对于一个 L 级的存储结构, 循环拆分的动作集合为 $\prod_{i=0}^{L-1}\prod_{d\in D_{\text{dim}}} d_i$. 其中, d_i 为每级缓存的维度子集, 表示该缓存级别中相应维度上的最大值, 其值根据 CNN 运算层的参数确定. 对于一般的二维阵列计算结构, 如果每级缓存中都包含 7 层循环, 且运算阵列支持二维方向展开, 那么各级缓存中循环展开的动作集合为 $C_7^2\prod_{j=0}^{L}U_j^L$. 其中, C_7^2 表示从 7 层循环中找出 2 个循环进行展开; U_j^L 表示该展开维度的最大值. 同样前提下, 各级缓存的循环排序调度空间为 7!, 而对于一个 L 级缓存架构, 循环排序的调度空间会增加为 $(7!)^L$.

由于双延迟深度确定性策略梯度(twin delayed deep deterministic policy gradient, TD3)算法的动作空间是连续的, 而循环调度策略的分布是离散的, 因此需要使用

$$a_{\text{discrete}}=\text{round}\left(a_{\min}-0.5+a_{\text{continuous}}\times\left(a_{\max}-a_{\min}\right)\right) \tag{11}$$

将连续的动作转换为离散的调度策略选择. 其中, a_{discrete} 为实际的调度策略; $a_{\text{continuous}}$ 为智能体做出的动作, 取值范围为 $[0,1]$; $a_{\min}$ 和 $a_{\max}$ 分别为各调度策略的最小和最大数值, 如针对循环拆分, $a_{\min}=1$, $a_{\max}=\prod_{i=0}^{L-1}\prod_{d\in D} d_i$. 因此, 最终的动作函数用一个各维度取值范围在 $[0,1]$ 的三维矢量表示.

3.3 状态空间

如图 2 所示, CNN 处理器模型的状态空间包括架构参数、访存成本模型、CNN 模型参数、可选的调度约束以及存储资源重配置. 其中, CNN 模型参数并不会随着调度策略的改变而发生变化. 为了进一步加快优化模型的收敛速度, 可人为地对可能的调度进行约束. 由于约束不会在调度策略搜索过程中发生变化, 因此 CNN 模型参数和调度约束参数均为静态状态. 静态状态对各个调度策略的影响一致, 即在智能体搜索最优策略时不需要参考静态状态. 然而, 在 CNN 处理器中的存储区可分为基于寄存器的 0 级存储单元、基于片上存储器的 1 级存储单元和基于片外存储器的 2 级存储单元, 针对 3 级存储单元的存储容量、缓存功耗和并行度, 则共为 9 维矢量. 各级缓存所需存储量、各级缓存的功耗以及各级缓存的并行乘加运算量均属于动态状态, 会随着循环调度策略的改变而发生变化. 因此本文输入到智能体的状态空间仅由动态状态组成, 具体为

$$O_i(s)=\left(R_i^{\text{capacity}},T_i^{\text{cost}},M_i^{\text{parallel}}\right) \tag{12}$$

其中, 根据智能体针对第 i 层的调度策略完成本层卷积运算任务, R_i^{capacity}, T_i^{cost} 和 M_i^{parallel} 分别表示各级缓存所需的存储容量、功耗和可提供的并行度. 将各级存储所需的存储容量、功耗和可提供的并行度, 通过线性转换算法将各维参量值均匀映射到[0,1]区间, 即 $N=\dfrac{D-D_{\min}}{D_{\max}-D_{\min}}$. 其中, N 表示映射到[0,1]区间后的值; D, $D_{\max}$ 和 $D_{\min}$ 分别表示存储容量、功耗和可提供的并行度 3 类参量及其最大值和最小值.

4 自适应片上弹性缓存态划分机制

在执行卷积等计算任务前, 可重构 CNN 处理器将各卷积运算层的循环调度策略发送到所如图 2 所示的片上弹性存储控制单元[10,15], 控制单元接收并译码产生各级缓存的地址空间, 以指导交叉互联网络完成地址映射. 弹性缓存的具体结构如图 2 所示, 主要由数据存储区、交叉互联网络、弹性存储控制单元以及可配置缓存接口组成. 其中, 数据存储区由多个 SRAM 组成的存储块组成, 因此存储块是弹性存储结构的基本存储单元, 每个存储块具有相同的数据位宽和存储容量; 可重构交叉互联网络负责存储块的地址映射以及动态划分; 总线接口接收来自控制单元的配置信息, 并向控制单元提供状态信息; 控制单元通过接收当前运算层的循环调度策略, 确定各级缓存的存储容量, 运算开始前完成对交叉互联网络的配置, 并在运算过程中检查交叉互联网络的工作状态. 假定本文设计的弹性缓存器包含 L 个存储级, 其中 0 级缓存为寄存器文件, $L-1$ 缓存一般指片外 DRAM, 并且各级存缓存器之间采用基于 AXI 协议总线进行互联. 本节将分别介绍循环拆分、循环排序以及循环展开对地址空间的影响.

4.1 针对循环拆分的缓存划分方法

各级缓存需要的存储容量由循环拆分策略决定, 本文通过划分弹性片上缓存区满足各类数据存储需求. 由式(1)可知, 在循环拆分后的第 i 级缓存中, 某一类型所需存储量为 i 级和更高级别 $(i,i-1,\cdots,0)$ 缓存区中该类型数据在各循环维度上拆分因子的乘积. 由于本文的高级可扩展接口

(advanced extensible interface, AXI)地址空间按照字节编址, 为了完成地址空间的计算, 因此需要在式(1)的基础上增加数据位宽, 从而得到以字节为单位的地址空间分配信息, 表示为

$$s_i = w_{\text{Byte}} \prod_{d \in D_{\text{dim}}} \prod_{j=0}^{i} d_j \tag{13}$$

其中, w_{Byte} 表示以字节为单位的数据位宽. 交叉互联网络在接收到地址空间的配置信息后, 会优先针对各级缓存容量粗粒度划分存储块, 其划分策略为

$$\begin{cases} L_{\text{Bank}} \geqslant \sum_{i=1}^{L-1} i_{\text{Bank}} \\ i_{\text{Bank}} = s_i / s_{\text{Bank}} \end{cases} \tag{14}$$

其中, L_{Bank} 表示弹性存储结构内存储块的数量; i_{Bank} 表示 i 级缓存分配的存储块数目; s_{Bank} 表示存储块的容量. 在得到各级缓存所需存储块数目后, 交叉互联网络利用类似于式(14)的方法计算各类型数据需要的存储块数量, 进而实现对于地址空间的划分.

4.2　针对循环排序的缓存划分方法

由于不同类型数据在存储中的组织方式与循环排序相关, 如果直接将循环拆分后各类型数据顺序存储, 则在不同的循环排序策略下, 访存地址会经常发生大幅度变化, 导致降低数据总线在连续突发传输过程中的读写效率; 因此还需要结合循环排序, 进一步分析各类型数据在存储中的组织方式. 由于本文采用类似于访问 Cache 的数据缓存方式, 即访存的优先级从 0 级到 $L-1$ 级逐级递减, 并且对于同类型的数据, 高级别缓存会将低级别的数据作为基本存储块进行排列, 因此只需要考虑该级别缓存中基本数据块的排列方式. 为了将按连续地址访问各数据块, 本文按照底层循环优先的原则排列基本数据块, 即从循环的最底层开始向上遍历, 从而能够提高总线的突发传输效率.

例如, 以表 2 中列出针对典型卷积运算过程, 以及对应的 3 级缓存组织方式为例, 本文针对循环排序的缓存划分方法可分别在 3 级缓存中进行调度. 在 0 级缓存中, 对于权重数据优先对卷积核尺寸维度的数据进行存储, 并以 3×3 的卷积核为单位存储 2 组输入通道, 然后以 2×3×3 为单位存储 4 组输出通道; 对于输入特征数据则优先存储宽度为 18 的特征行, 再以特征行为单位存储 18 个特征行, 最后以一张特征图为单位存储 2 组输入通道; 对于输出特征数据则优先存储宽度为 16 的特征行, 再以特征行为单位存储 16 个特征行, 最后以一张特征图为单位存储 4 组输出通道. 在 1 级缓存中, 以 0 级缓存作为基本数据块, 权重在该级别无循环, 故只存储一个基本数据块的量; 对于输入和输出特征, 优先存储行方向上的 2 组拆分, 再存储列方向上的 2 组拆分; 在 2 级缓存中, 1 级缓存作为基本数据块, 在不考虑批处理维度情况下, 3 种数据都将优先存储输入通道上的 4 组拆分, 再存储输出通道上的 8 组拆分.

表 2　卷积层示例

3 级存储	卷积计算过程

level 2	Alloc I[32][8][34][34]
	Alloc O[32][32][32][32]
	Alloc W[32][8][3][3]
	for (int b_2=0; b_2<32; b_2++)
	for (int k_2=0; k_2<8; k_2++)
	for (int c_2=0; c_2<4; c_2++)

level 1	Alloc I_1[1][2][34][34]
	Alloc O_1[1][4][32][32]
	Alloc W_1[4][2][3][3]
	for (int y_1=0; y_1<2; y_1++)
	for (int x_1=0; x_1<2; x_1++)

level 0	Alloc I_0[1][2][18][18]
	Alloc O_0[1][4][16][16]
	Alloc W_0[4][2][3][3]
	for (int k_0=0; k_0<4; k_0++)
	for (int c_0=0; c_0<2; c_0++)
	for (int y_0=0; y_0<16; y_0++)
	for (int x_0=0; x_0<16; x_0++)
	for (int f_y=0; f_y<3; f_y++)
	for (int f_x=0; f_x<3; f_x++)
	$O_0[b_0][k_0][y_0][x_0]$+=$I_0[b_0][c_0][y_0+f_y][x_0+f_x]\times W_0[k_0][c_0][f_y][f_x]$

4.3　针对循环展开的缓存访问方法

各级缓存间的通信使用突发传输(如 AXI-Stream 接口), 单次传输突发数据量及突发长度共同决定了突发传输的并行输出能力; 同时, 鉴于循环展开方法决定了当前缓存级别的并行运算能力, 当前缓存级别的输入数据量需要满足并行运算的需求. 结合以上 2 点, 循环展开策略决定了缓存级别间总线通信过程的相关参数配置. 如采用 $f_x \mid f_y$ 的循环展开策略, 则计算块分别在 PE 阵列的行方向和列方向上进行循环展开. 当 0 级缓存向 1 级缓存发送读取数据请求时, 需要配置突发数据量和突发长度, 以满足当前运算并行度的需求. 受到不同硬件平台资源的限制, 总线能提供的最大突发数据量也不同. 如以 AlexNet 的第 1 和第 3 卷积层为例, 第 1 卷积层的卷积核尺寸为 11×11, 第 3 卷积层的卷积核尺寸为 3×3, 假设权重的数据位宽为 16 bit, 第 1 卷积层所需的并行输出量为 242 bit.

由于 AXI 可以提供的突发数据量最大值为 128 bit, 因此需要将突发长度配置为 2 才能满足需求; 第 3 卷积层所需的并行输出量为 18 bit, 则只需将突发数据量配置为 32 即可满足需求.

5 实验及结果分析

5.1 实验建立

分别采用学术界和工业界具有代表性的 DNN 处理器架构——Eyeriss[6,8]和 TPU[7]进行实验, 以验证本文提出的映射策略和片上储存划分和调度方法. 2 种典型的可重构 CNN 处理器架构参数如表 3 所示. 其中, 访问寄存器、片上存储单元及片外存储器的功耗等参数的设置均与文献[12]相同, 并通过功耗和性能模型来评估采用 ASIC 设计方法实现的处理器效率. 本文选取了 3 种典型 CNN 中的卷积层作为待加速运算层: 拥有多种卷积核尺寸的 AlexNet, 面向嵌入式的轻量化网络 MobileNet 以及具有残差结构的 ResNet50. 因为 ResNet50 由多个相似的卷积块组成, 其中前 12 层中包含只由一层卷积及由多层卷积和跨层连接的情况, 可以代表 ResNet50 模型的计算特性, 所以本文在性能分析中只展示前 12 层的结果. Eyeriss 和 TPU 均为 3 级缓存结构, 然而两者中各级缓存容量不同, Eyeriss 的 0 级缓存和 1 级缓存的容量都要大于 TPU, 因此其单次访存功耗也更大; 而 TPU 拥有更大的 PE 阵列, 因此可以提供更灵活的循环并行展开度. 为了探索更多的循环调度空间, 本文使用了 16×16 的 PE 阵列作为运算单元.

表 3 Eyeriss 和 TPU 架构参数

参数	Eyeriss	TPU
缓存级别数	3	3
各级容量/KB	[512, 131.072, 1 073 741.824]	[3 265.536, 29 360.128]
单次访存功耗/pJ	[0.96, 20, 200]	[0.015, 13.5, 150]
可并行 PE 数	256	16 384
PE 间单次访问功耗/pJ	0.035	0.035
运算精度/bit	16	16

为了评估本文提出的可重构 CNN 处理器架构和映射策略, 根据文献[15]对于弹性存储硬件架构的描述, 在基于 FPGA 的可重构计算平台上实现了本文提出的弹性存储单元. 为了能够使弹性存储应用在存储资源较少的嵌入式平台上, 在对硬件电路进行综合时选择 Xilinx Ultra96-V2 开发平台作为资源约束. 为了提高计算效率, 计算单元采用的混合数据位宽[23]——(MP5.98, MP7.8), 即各层特性值和权重的平均数据位宽分别为 5.98 bit 和 7.80 bit; 并将数据的传输位宽统一为 16.00 bit. 基于 Eyeriss 以及 TPU 中存储结构, 本文分别设计了多种资源配置的弹性存储电路, 表 4 给出了对于 3 种典型的神经网络在 2 种架构的弹性存储电路中资源使用情况.

表 4 3 种网络在不同架构下弹性存储电路资源使用情况

处理器架构及网络模型	Eyeriss			TPU		
	AlexNet	MobileNet	ResNet50	AlexNet	MobileNet	ResNet50
存储器(SRAM)/Kb	3 564	7 164	6 084	2 700	7 380	6 804
查找表(LUTs)	13 260	27 640	22 490	10 150	28 650	24 930
寄存器(FFs)	22 590	47 080	38 300	17 300	48 800	42 460

5.2 功耗影响分析

分别采用本文提出的弹性划分方法、Eyeriss 和 TPU 中采用的片上双缓存方法(由于主流 DNN 加速器通常采用双缓存结构和调度策略, 具有广泛的代表性, 所以本文将其作为测试基准), 分别实现基于 Eyeriss 和 TPU 的可重构 CNN 处理器. 其中, 针对 AlexNet 各卷积层的功耗对比如图 3 所示, 可以看出弹性存储电路的整体功耗要低于双缓存结构. 基于 Eyeriss 的弹性存储功耗最大为 4.45×10^7 pJ, 最小为 0.94×10^7 pJ, 整体功耗比双缓存结构下降了 17.72%; 基于 TPU 的弹性存储的功耗最大为 6.56×10^7 pJ, 最小为 0.71×10^7 pJ, 整体功耗比双缓存结构下降了 18.43%.

由于第 1 卷积层和第 2 卷积层中双缓存所需的存储量较小, 因此弹性存储对于 2 个运算层的单次访问功耗要大于双缓存结构, 即导致了第 1 卷积层和第 2 卷积层中弹性存储的访存功耗略大于双缓存结构. 根据式(4), 访存功耗等于单次访存功耗和访存次数的乘积, 因此并不是存储量越大, 访存功耗就越大. 根据式(5), i 级缓存中某一类型的访存次数取决于 i 级以外 $(i,i+1,\cdots,L-1)$ 缓存对于该数据的重用次数, 因此弹性存储级别中某类型数

据的访存次数不仅取决于弹性存储中对于该类型数据的重用次数，还取决于片外 DRAM 中对于该类型数据的重用次数.

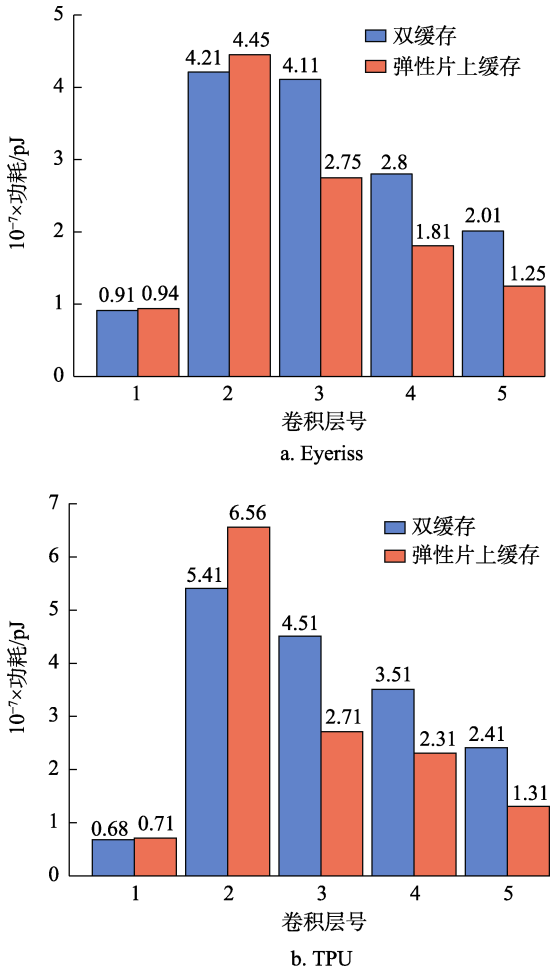


图 3　AlexNet 模型中各卷积层的运算层功耗

针对 AlexNet，在基于 Eyeriss 的弹性存储中，第 4 卷积层的循环调度方法在片外 DRAM 级别对输出通道数使用了因子为 4 的循环拆分，第 2 卷积层的循环调度方法在片外 DRAM 级别对输出通道 K 使用了因子为 32 的循环拆分，导致第 2 卷积层对于弹性存储的访存次数大于第 4 卷积层. 因此在图 3 中，虽然第 4 卷积层占用的存储量最大，但是其功耗要低于占用存储量较少的第 2 卷积层；同理，在基于 TPU 的弹性存储中，第 3 卷积层的功耗要低于存储量远小于它的第 2 卷积层.

采用相同的循环调度策略，MobileNet 的运算层在弹性存储与普通双缓存中的功耗对比如图 4 所示. 基于 Eyeriss 的弹性存储功耗最大为 2.89×10^8 pJ，最小为 0.73×10^8 pJ，整体功耗双双缓存结构下降了 27.13%；基于 TPU 的弹性存储功耗最大为 2.95×10^8 pJ，最小为 0.56×10^8 pJ，整体功耗相较于双缓存结构下降了 13.52%，两者功耗的最大值均出现在第 10 卷积层中.

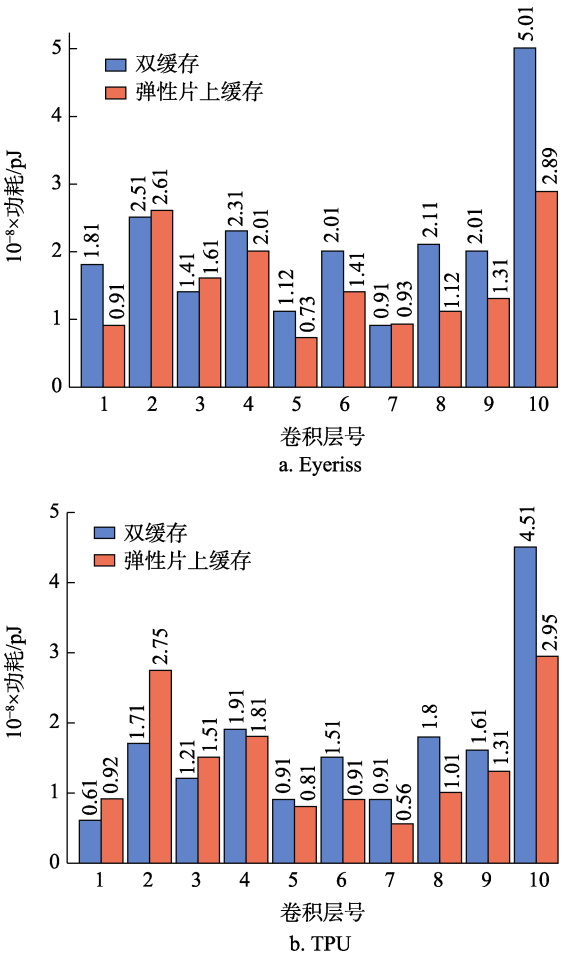


图 4　MobileNet 中各卷积层的运算层功耗

在基于 Eyeriss 的弹性存储中，第 10 卷积层的循环调度方法在弹性存储级别上对特征列，特征行以及输入通道维度分别进行了因子为 7，7 和 2 的循环拆分，同时在片外 DRAM 级别对输出通道 K 使用了因子为 4 的循环拆分，从而增加了各类型数据访问弹性存储的次数，产生了更多功耗. 而在基于 TPU 的弹性存储中，第 10 卷积层的循环调度方法在弹性存储级别上对输入通道进行了因子为 1 024 的循环拆分，同时在片外 DRAM 级别对输出通道 K 使用了因子为 4 的循环拆分，从而增加了输入特征以及输出特征访问弹性存储的次数，也产生了更多功耗.

基于相同的循环调度策略，ResNet 的运算层在弹性存储与普通双缓存中的功耗对比如图 5 所示. 基于 Eyeriss 架构的弹性存储功耗最大为 1.82×10^8 pJ，最小为 1.10×10^7 pJ，整体功耗比双缓存结构下降了 30.28%；基于 TPU 的弹性存储功耗最大为 6.21×

10^7 pJ, 最小为 1.16×10^7 pJ, 整体功耗相较于双缓存结构下降了 17.33%, 两者功耗的最大值均出现在第 8 卷积层.

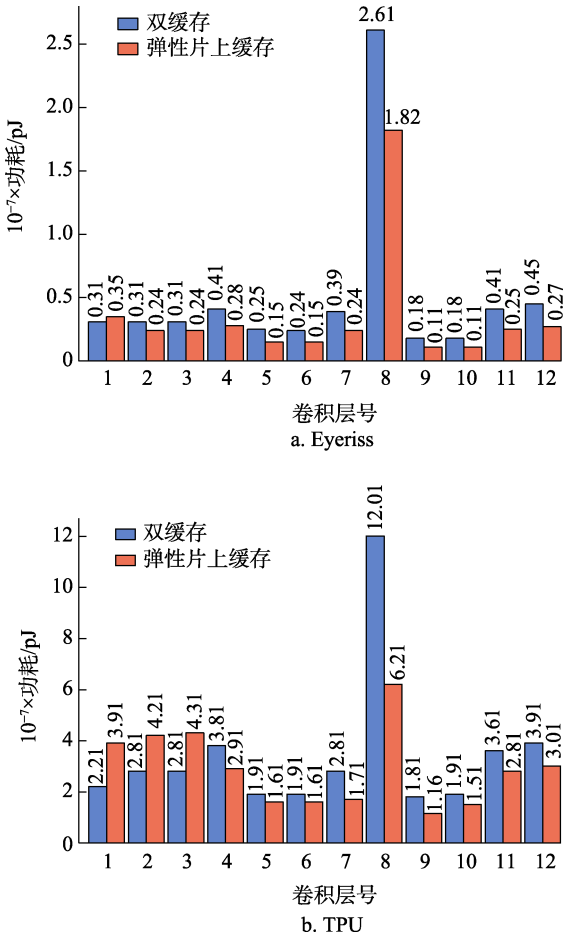


图 5 ResNet50 前 12 个运算层的功耗

在基于 Eyeriss 的弹性存储中, 第 10 卷积层的循环调度方法在弹性存储级别上对特征列和特征行维度分别进行了因子为 14 和 28 的循环拆分, 同时在片外 DRAM 级别对输出通道和特征列分别使用了因子为 16 和 2 的循环拆分, 从而增加了各类型数据访问弹性存储的次数, 产生了更多功耗. 而在基于 TPU 的弹性存储中, 第 10 个卷积层的循环调度方法在弹性存储级别上对特征列和特征行分别进行了因子为 28 和 28 的循环拆分, 同时在片外 DRAM 级别对输出通道 K 使用了因子为 4 的循环拆分, 从而增加了输入特征以及输出特征访问弹性存储的次数, 也产生了更多的功耗.

5.3 计算效能的测试与分析

针对 3 种典型的 CNN 模型, 本文首先对 Eyeriss 和 TPU 采用与文献[12]相同的调度约束下搜索, 以 Eyeriss 和 TPU 在计算块内的数据流作为调度约束, 采用与原始 Eyeriss 和 TPU 类似的映射策略, 并将其作为针对本文工作的对比基准. 本文以 Eyeriss 和 TPU 的架构为基础, 通过改变 1 级缓存的容量拓展循环调度空间, 同时启用存储资源重配置功能, 搜索其他可能的高效数据流. 为了方便比较, 假设 PE 间的访问功耗设定为相同的值, 同时根据功耗和性能的计算方法, 以处理器的计算效能(tera operations per second/Watts, TOPs/W)作为优化结果的比较标准. 针对 AlexNet 的卷积层, 为了方便与调度约束下的优化结果比较, Eyeriss 架构使用批处理大小为 1 的网络, 而针对 TPU 架构则采用与经典调度模式一样的批处理大小为 16 的执行模式. 图 6 所示为本文调度模型对于 2 种处理器的优化结果与原始调度的效能比.

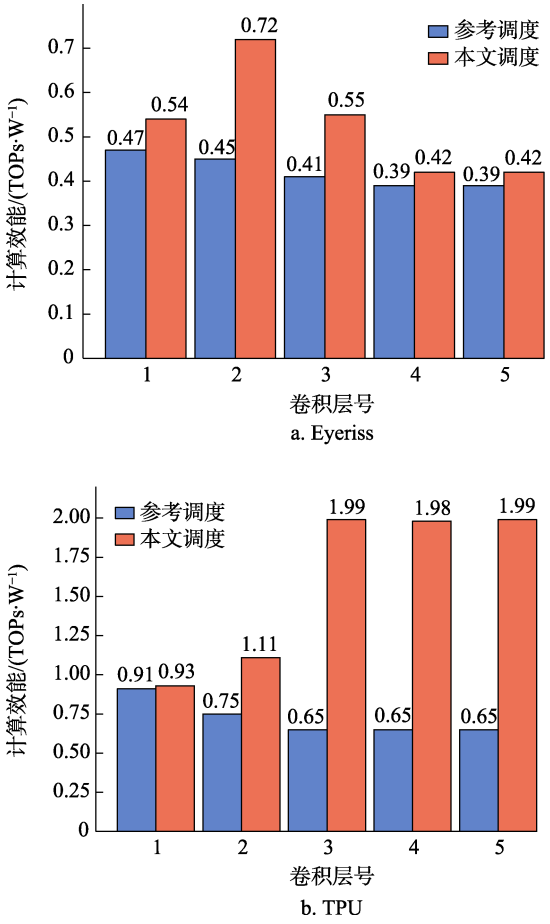
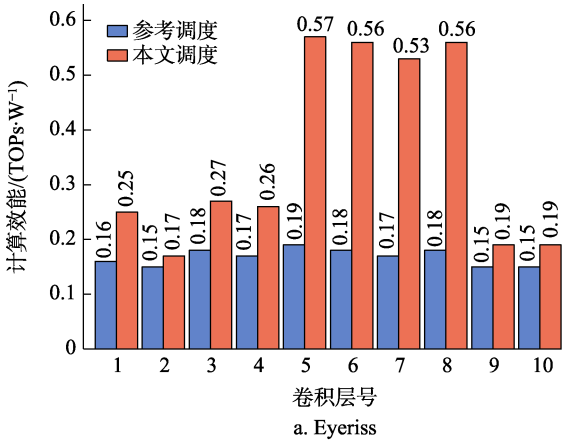


图 6 针对 AlexNet 优化结果

由于 $f_x \mid f_y$ 数据流对于数据的重用性更好, 因此 Eyeriss 的效能相比调度约束下的优化结果并没有很明显的提升, 只能通过改变 1 级存储容量的方式来探索更多的循环拆分方法, 以降低各级访存功耗. 实验结果显示, 针对 Eyeriss 架构的效能比调度

约束下的优化结果提升了 61%，性能峰值达到了 0.72 TOPs/W；如图 6 所示，改变数据流使得处理器性能不再受输入输出通道的限制，有效地提升了 TPU 架构的效能，同时使用批处理更大的网络使优化后的效能要优于 Eyeriss. 实验结果还显示，TPU 的效能比调度约束下的优化结果提升了 206%，性能峰值达到了 1.99 TOPs/W.

针对 MobileNet 和 ResNet50 的卷积层，2 种处理器均采用批处理大小为 1 的网络，优化模型对于 2 种处理器的优化结果与调度约束下的优化结果的效能比分别如图 7 和图 8 所示.



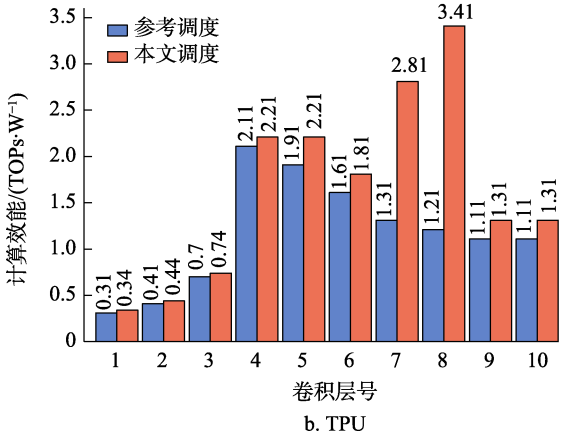
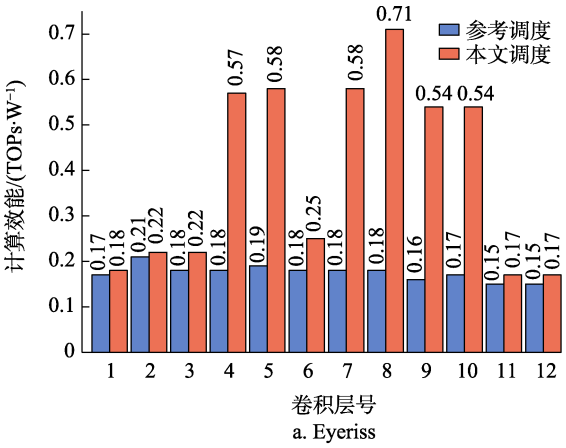


图 7　针对 MobileNet 优化结果

图 7 中，MobileNet 属于轻量化网络，其数据的重用率不高，因此采用原始映射策略，导致 2 种处理器的效能受到了限制. 图 8 中，它们效能的分布方式与 MobileNet 类似，但是由于 ResNet50 的计算密集性更高，其循环调度空间也更大，因此该网络下两者的效能峰值很接近，优化效果更好. 针对 2 种 CNN 模型在 Eyeriss 和 TPU 处理器架构中，采用本文提出的调度策略可达到的效能峰值，与基于调度约束下的优化结果相比的效能提升如表 5 所示.



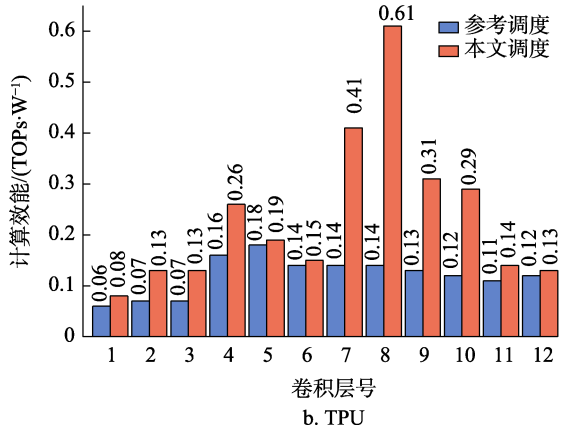


图 8　针对 ResNet50 前 12 个运算层的优化结果

表 5　针对 2 种 CNN 的效能及其提升比

网络模型	架构	效能峰值/(TOPs·W^{-1})	效能提升/%
MobileNet	Eyeriss	0.57	200
	TPU	0.03	50
ResNet50	Eyeriss	0.70	233
	TPU	0.60	215

5.4　与相关研究的对比

按照本文提出的映射策略的片上弹性缓存访问方法，基于 Ultra-96 平台上实现的可重构 CNN 处理器，与其他基于 FPGA 平台的最新相关研究的对比结果如表 6 所示，所有相关研究的实验平台均采用 Xilinx 公司推出的类似 FPGA 平台实现. 针对基于 FPGA 的可重构计算平台中片上存储资源(如 BRAM)和计算资源(如 DSP)的特性，采用和片上缓存单元结构已有相关研究，将片上缓存单元设计为可通过可重构互连网络和本文提出的片上弹性缓存访问策略，以动态缓存数据. 与此同时，通过限制对 LUT 等资源量提高 FPGA 布局布线的难度，以提高系统时钟频率. 计算单元效率采用如文

表 6 与相关研究的功耗、计算性能和效率对比

研究工作	FPGA 平台	运行频率/MHz	数据存储位宽/bit	DSP		BRAM 使用量/Kb	功耗/W	计算性能/GOPs	计算效率/(GOPs·DSP^{-1})
				使用量/个	使用率/%				
RP[24]	Virtex-7 690T	100	32	3 166	88	25 930.80	10.20	113.90	0.036
PM[10]	Virtex-7 690T	100	32	3 204	89	19 580.40	11.30	127.70	0.040
本文	Ultra96-V2	**150**	16	314	87	3 564.00	**5.40**	**134.20**	**0.427**

注. 粗体表示最优值.

献[23]示例的混合精度计算模式, 数据存储和传输采用 16 bit 的模式.

可以看出, 与采用固定片上缓存结构和访问策略的 CNN 处理器[24]以及与本文相似弹性缓存的 CNN 处理器[10]相比, 本文方法的计算效率和效能分别提高 11 倍和 2 倍以上. 基于深度强化学习的自适应算法探索最优数据流模式, 提高了片上数据重用率, 减少了计算延迟和功耗均较大的片外数据访问数量, 提高了访存效率, 验证了本文提出的自适应映射策略的有效性. 为了测试本文提出的调度策略对计算效率的影响, 当计算数据位宽和时钟频率均与文献[10]相同时, 计算性能可达到 28.5 GOPs, 计算效率为 0.09 GOPs/DSP. 与文献[10]相比, 虽然实验平台中可重构资源的限制导致计算效率较低, 但是计算效率却提高了 125%, 说明本文提出的自动缓存单元划分方法可有效地提高可重构处理器的性能.

6 结 语

为了缓解可重构 CNN 处理器因“存储墙”而降低计算效率的问题, 本文在已有量化分析 CNN 计算模式与处理器功耗和效率的模型基础上, 采用软硬件协调设计的策略, 设计并实现了一种基于深度强化学习的自动映射方法, 通过自动调整 CNN 计算中不同的调度策略, 以探索 CNN 中不同卷积层运算的最优计算模式; 同时, 在已有片上弹性缓存结构的基础上, 设计了一种可根据不同计算模式动态调整的存储资源划分及数据访问方法. 采用典型的可重构 CNN 处理器架构, 以及经典的 CNN 模型计算任务, 分别测试和评估了本文提出的映射策略和弹性缓存重构方法. 实验结果表明, 与较新相关研究中提出的调度策略相比, 本文方法可有效地提高处理器的效能; 同时, 提出的弹性缓存划分方法可降低处理器的计算功耗, 从而满足嵌入式系统对计算应用需求. 然而, 由于本文采用离线训练的方法以搜索最优调度策略, 并通过静态重构 CNN 处理器以适应当前的应用场景, 因此当应用场景变化范围的较大时, 无法达到理想的效能. 未来可通过探索基于在线训练和动态重构的方法, 以扩大本文方法的适应范围.

参考文献(References):

[1] Hennessy J L, Patterson D A. A new golden age for computer architecture[J]. Communications of the ACM, 2019, 62(2): 48-60

[2] Sze V, Chen Y H, Yang T J, *et al*. Efficient processing of deep neural networks: a tutorial and survey[J]. Proceedings of the IEEE, 2017, 105(12): 2295-2329

[3] Krishna T, Kwon H, Parashar A, *et al*. Data orchestration in deep learning accelerators[M]. Cham: Springer, 2020: 1-164

[4] Farabet C, Martini B, Corda B, *et al*. NeuFlow: a runtime reconfigurable dataflow processor for vision[C] //Proceedings of the CVPR Workshops. Los Alamitos: IEEE Computer Society Press, 2011: 109-116

[5] Chen T S, Du Z D, Sun N H, *et al*. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM SIGARCH Computer Architecture News, 2014, 42(1): 269-284

[6] Chen Y H, Emer J, Sze V. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks[C] //Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture. Los Alamitos: IEEE Computer Society Press, 2016: 367-379

[7] Jouppi N P, Young C, Patil N, *et al*. In-datacenter performance analysis of a tensor processing unit[C] //Proceedings of the 44th Annual International Symposium on Computer Architecture. Los Alamitos: IEEE Computer Society Press, 2017: 1-12

[8] Chen Y H, Yang T J, Emer J, *et al*. Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices[J]. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2019, 9(2): 292-308

[9] Kwon H, Samajdar A, Krishna T. MAERI: enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects[J]. ACM SIGPLAN Notices, 2018, 53(2): 461- 475

[10] Azizimazreah A, Chen L Z. Polymorphic accelerators for deep neural networks[J]. IEEE Transactions on Computers, 2022, 71(3): 534-546

[11] Verhelst M, Shi M, Mei L Y. ML processors are going multi-core: a performance dream or a scheduling nightmare?[J]. IEEE Solid-State Circuits Magazine, 2022, 14(4): 18-27

[12] Yang X, Gan M Y, Liu Q Y, *et al*. Interstellar: using halide's scheduling language to analyze DNN accelerators[C] //Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2020: 369-383

[13] Gudaparthi S, Narayanan S, Balasubramonian R, *et al*. Wire-aware architecture and dataflow for CNN accelerators[C] //Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. New York: ACM Press, 2019: 1-13

[14] Mnih V, Kavukcuoglu K, Silver D, *et al*. Playing Atari with deep reinforcement learning[OL]. [2023-07-14]. https://arxiv.org/abs/1312.5602

[15] Chang L B, Zhang S B, Du H M, *et al*. A reconfigurable neural network processor with tile-grained multicore pipeline for object detection on FPGA[J]. IEEE Transactions on Very Large Scale Integration Systems, 2021, 29(11): 1967-1980

[16] Yang X. A systematic framework to analyze the design space of DNN accelerators[D]. Stanford: Stanford University, 2019

[17] Sze V, Chen Y H, Yang T J, *et al*. Efficient processing of deep neural networks[M]. Cham: Springer, 2020: 1-341

[18] Zhang C, Li P, Sun G Y, *et al*. Optimizing FPGA-based accelerator design for deep convolutional neural networks[C] //Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. New York: ACM Press, 2015: 161-170

[19] Suda N, Chandra V, Dasika G, *et al*. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks[C] //Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. New York: ACM Press, 2016: 16-25

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

[21] Lillicrap T P, Hunt J J, Pritzel A, *et al*. Continuous control with deep reinforcement learning[OL]. [2023-07-14]. https://arxiv.org/abs/1509.02971

[22] Dankwa S, Zheng W F. Twin-delayed DDPG: a deep reinforcement learning technique to model a continuous movement of an intelligent robot agent[C] //Proceedings of the 3rd International Conference on Vision, Image and Signal Processing. New York: ACM Press, 2019: 1-5

[23] Huang Zhengzhe, Du Huimin, Chang Libo. Mixed-clipping quantization for convolutional neural networks[J]. Journal of Computer-Aided Design & Computer Graphics, 2021, 33(4): 553-559(in Chinese)
(黄钲喆, 杜慧敏, 常立博. 卷积神经网络混合截断量化[J]. 计算机辅助设计与图形学学报, 2021, 33(4): 553-559)

[24] Shen Y M, Ferdman M, Milder P. Maximizing CNN accelerator efficiency through resource partitioning[C] //Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture. Los Alamitos: IEEE Computer Society Press, 2017: 535-547