计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202331005
61(6): 1370−1387, 2024

容错深度学习加速器跨层优化

张 青[1,2] 刘 成[1,2] 刘 波[3] 黄海同[1,2] 王 颖[1,2] 李华伟[1,2] 李晓维[1,2]

[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)

[3](北京控制工程研究所 北京 100094)

(zhangqing22s@ict.ac.cn)

Cross-Layer Optimization for Fault-Tolerant Deep Learning Accelerators

Zhang Qing[1,2], Liu Cheng[1,2], Liu Bo[3], Huang Haitong[1,2], Wang Ying[1,2], Li Huawei[1,2], and Li Xiaowei[1,2]

[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)

[3](*Institute of Beijing Control Engineering*, *Beijing* 100094)

Abstract Fault-tolerant deep learning accelerator is the basis for highly reliable deep learning processing, and is also critical to deploy deep learning in safety-critical applications such as avionics and robotics. Since deep learning is known to be both computing-intensive and memory-intensive, traditional fault-tolerant approaches based on redundant computing will incur substantial overhead including power consumption and chip area. To this end, we propose to characterize deep learning vulnerability difference across both neurons and bits of each neuron, and leverage the vulnerability difference to enable selective protection of the deep learning processing components from the perspective of architecture layer and circuit layer respectively for the sake of lower fault-tolerant design overhead. At the same time, we observe the correlation between model quantization and bit protection overhead of the underlying processing elements of deep learning accelerators, and propose to reduce the bit protection overhead by adding additional quantization constrain without compromising the model accuracy. Finally, we employ Bayesian optimization strategy to co-optimize the correlated cross-layer design parameters at algorithm layer, architecture layer, and circuit layer to minimize the hardware resource consumption while fulfilling multiple user constraints including reliability, accuracy, and performance of the deep learning processing at the same time.

Key words cross-layer optimization; fault-tolerant deep learning accelerator; vulnerability factor; hybrid architecture; selective redundancy

摘 要 容错深度学习加速器是保障高可靠深度学习的基石,也是深度学习应用于安全关键领域如宇航、机器人等面临的一个关键环节.然而,深度学习计算和访存都非常密集,传统基于冗余计算的容错方法直接应用于深度学习加速器的容错设计会导致严重的功耗、芯片面积等硬件资源开销.为此,从神经元计算任务和神经元的数据位宽2个维度挖掘深度学习模型对于故障的敏感度差异,并利用这些差异从架构和电路层分别对于敏感的部分提供更多的保护以降低容错代价.同时,利用深度学习自身的容错特性,通过

收稿日期: 2024-02-01; **修回日期**: 2024-03-14

基金项目: 国家重点研发计划(2022YFB4500405); 国家自然科学基金项目(62174162); 空间可信计算与电子信息技术实验室开放基金资助(OBCandETL-2022-07)

This work was supported by the National Key Research and Development Program of China (2022YFB4500405), the National Natural Science Foundation of China (62174162), and the Spatial Trusted Computing and Electronic Information Technology Lab Open Fund (OBCandETL-2022-07).

通信作者: 刘成(liucheng@ict.ac.cn)

限制量化缩小电路层需要保护的电路逻辑规模. 最后，利用贝叶斯优化协同优化算法、架构和电路的跨层设计参数，在保障深度学习可靠性、精度以及性能的前提下，最小化硬件资源开销.

关键词 跨层优化；容错深度学习加速器；脆弱因子；异构架构；选择性冗余

中图法分类号 TP391

深度学习不断地从传统的计算机视觉、自然语言处理等领域向着更加广阔的应用领域渗透，越来越多地涉及一些安全关键（safety-critical）的应用场景[1-8]，如自动驾驶、宇航、机器人等. 相对于常规的深度学习应用，安全关键的深度学习应用在精度、速度的基础上又增加了可靠性的要求，而且可靠性甚至成为这类深度学习应用的关键指标，直接决定其是否可以被应用接受，例如车载深度学习模块必须要满足ISO26262规定的可靠性标准才能应用于车载系统[1,3,9]. 深度学习加速器（deep learning accelerator, DLA）作为支撑高能效深度学习处理的核心专用处理器[10-11]，在先进纳米级工艺下，伴随着更高的晶体管集成度以及更低的阈值电压，不可避免地会受到软故障的影响，导致深度学习推理出现异常错误甚至引发安全事故[12-13]. 因此，设计容错DLA使其能够有效容忍故障、实现高可靠的推理是推动深度学习在安全关键领域应用的关键.

深度学习处理是典型的计算和访存密集型任务，传统的芯片可靠性设计方法如三模冗余保护等尽管可以解决故障导致的精度损失问题，但是大量的冗余计算会极大地影响深度学习的处理速度、芯片面积以及功耗等，与DLA最基础的设计目标如速度、能效等出现冲突，限制了其在安全关键领域的应用[1,9]. 另一方面，深度学习处理的输出通常是离散的，很多较小的计算错误对于深度学习推理结果的影响很小. 同时，深度学习处理中包含了很多非线性函数，可以过滤很多错误的中间计算数据，也会缓解软错误等对于推理结果的影响，这些特性使得深度学习的处理相对于通用计算具有更强的故障容忍能力. 很多DLA可靠性设计工作[14-16]利用深度学习这一天然的容错特性，降低深度学习的容错成本，实现精度、可靠性以及性能等多个设计目标的共同优化.

其中一大类方法是根据深度学习不同部分对于故障的敏感程度差异使用选择性的冗余保护，从而，在不损害深度学习计算可靠性和精度的前提下降低容错的成本. Schorn等人[17]与文献[12,18]提出将DLA的计算阵列分成不同的可靠性区域用于处理重要性不同的神经元，这样可以对不同的区域采用对应的容错保护以降低容错成本；Mahdiani等人[19]提出重点保护乘法器的高位计算部分来增强整体DLA的可靠性；也有一些工作直接从模型或者算法层进行选择性冗余计算[20-24]以提高深度学习的容错能力并减少容错的成本. 文献[17-24]工作表明我们可以从深度学习算法层、加速器的架构层、电路层等挖掘深度学习内在的容错差异，对于脆弱的部分给予更多的保护以降低容错成本. 由于不同的抽象层面涉及的信息不同，这使得对应的容错效果和容错成本存在显著的差异. 电路层可以准确提升整体设计的容错能力，但是灵活性差，难以直接利用应用层和架构层的信息，容错成本往往比较高；算法层可以更加充分地利用模型层的信息，针对特定模型定制容错选择、降低容错成本，但是缺乏底层细节，在故障率较高和可靠性需求较高的情况下，容错成本会急剧增加；加速器架构层容错的特点则介于电路层和算法层容错之间，仅有部分算法层和电路层信息，且仍然缺少统一的容错深度学习架构设计. 以往的DLA容错通常从某1个或者2个层面来提高DLA的可靠性，缺少系统的跨层协同优化，无法充分利用不同层容错设计的优势. 此外，跨层参数之间的关系复杂、设计空间大，手工设计不仅效率低、易错，而且难以达到全局优化，也难以快速地为不同的应用需求提供优化方案.

为了解决上述问题，我们提出了面向容错DLA的跨层优化设计框架，在算法层，我们从神经元和比特位2个维度分析了深度学习计算对于软故障的敏感性，将深度学习的计算任务分为重要的神经元计算和普通的神经元计算，将神经元数据的比特位又分为重要的比特位和普通的比特位，用于指导容错DLA的细粒度选择性保护. 在加速器架构层，提出了基于重计算的异构计算框架，分别用于处理普通的神经元计算和重要的神经元计算，支持神经元计算的选择性保护. 在电路层，提出了按位保护的电路冗余设计，仅仅保护计算单元的重要比特位逻辑，并且根据算法层的敏感性分析，对于2种计算阵列提供不同的电路层位冗余保护设计，进一步降低冗余成本. 此外，我们首次提出通过量化约束限制深度学习神经元重要比特位的直接关联逻辑规模，从而减少

电路层按位冗余保护的成本. 最后，根据目标应用的需求，自动化协同不同层的设计选择以及快速优化容错 DLA 设计的可靠性、资源开销和性能等目标.

本文的主要贡献包括 4 个方面：

1)从深度学习模型的不同神经元和比特位 2 个维度分析其对于软故障的敏感性，在此基础上跨越电路、架构以及算法 3 个层面进行 DLA 的容错设计，同时优化可靠性、资源开销以及性能等多个设计目标；

2)首次提出结合深度学习模型的量化策略和 DLA 计算单元的位冗余设计，通过限制模型的量化选择，在不影响精度的前提下，减少需要保护的高位电路规模、降低 DLA 电路层冗余设计成本；

3)设计了一个基于重计算的异构容错深度学习加速器架构，能够根据神经元对于软故障的敏感性差异实现神经元级别的细粒度选择性保护，并且可以容忍重要神经元分布的差异，在保证计算精度的前提下降低容错成本；

4)实验表明，相对于单独的电路层、架构层以及算法层选择性容错设计，提出的跨层优化设计方法在 DLA 的可靠性、资源开销以及性能总体上表现出显著的优势.

1 相关工作

DLA 容错设计是保障深度学习推理可靠性的基础，为了提高 DLA 的可靠性，很多前期工作从电路、架构、算法等不同的抽象层分别提出了很多容错设计方法，我们将分别介绍相关的研究工作.

在电路层，传统的容错编码技术如 ECC(error correction code)可以用于保护片上缓存，冗余可以保护控制和计算逻辑. 为了降低直接冗余的成本，Mahdiani 等人[19] 提出了对于深度学习计算单元的高位部分给予更高优先级的保护而忽略低比特位的逻辑以减少冗余保护的成本；文献 [25-27] 提出了采用随机计算、近似计算等逻辑电路代替传统二进制计算逻辑以增强容错能力或者提高计算能效. Reagen 等人[14] 提出采用 Razor 电路去检测由电压降低导致的时序故障，而后利用深度学习模型的自身容错特性增加字和位掩码去修复 SRAM 数据错误，从而在不降低精度的前提下提高计算能效. Zhang 等人[28] 提出在 DLA 的计算单元上增加常数 0 旁路，当相应的计算单元出现硬故障的时候，可以将其旁路置为常数 0，缓解故障导致的巨大数值波动以及精度损失. Clemente 等人[29] 提出在 Hopfield 神经网络加速器上增加冗余的连接，并利用投票逻辑实现纠错，这种策略相对于三模冗余可以显著降低容错设计开销. 文献 [14,19,25–29] 工作主要通过电路设计增强容错，但是也需要在算法或者模型层对电路层引入的数值偏差进行微调.

在架构层，Liu 等人[12,18] 针对 DLA 计算阵列上的任意的计算单元硬故障问题提出了异构计算架构，采用不同于二维脉动阵列的点乘计算阵列实现对任意计算单元上任务的重计算. Schorn 等人[17] 提出将 DLA 的计算阵列分成高可靠的计算区域和普通的计算区域，分别用于处理对于故障敏感的计算任务和对于故障不敏感的计算任务. 然而，对于故障敏感的计算和不敏感的计算的分布通常会随着模型甚至输入改变，仍然缺少详细的微结构支撑. Gao 等人[30] 提出在 DLA 的基础上增加集成学习单元，从而可以使用多个轻量级的深度学习模型并行计算容忍硬件故障，提高推理的可靠性，然而这种方法依赖于模型的重新设计. Ozen 等人[31] 提出在深度学习计算阵列上增加和校验单元，从而利用 ABFT(algorithm-based fault tolerance)技术，实现实时的故障检测和纠错，但受限于 ABFT 技术的容错和纠错能力，其主要应用于低故障率场景.

在算法层，很多工作通过挖掘深度学习模型自身的容错能力和参数冗余，在保证精度的前提下通过改变模型参数甚至结构来增强模型对于底层硬件故障的容忍能力，从而在不改变 DLA 硬件设计的情况下提高深度学习处理的可靠性. 文献 [4–5,15–16,18,22,32–33] 等工作通常借助容错训练改变深度学习模型参数或者架构，这类基于模型训练的容错设计往往依赖样本数据和大量的重新训练，且对于故障率比较敏感，难以在训练阶段被充分地考虑. 不同于这类严重依赖大量应用数据训练的容错方法，一些方法通过使用等效的计算方式[20,34]，或者引入新的激活函数或者数值限制[35-37]，或者利用 Checksum 机制纠错[34,38]，来提高模型的容错能力，这类方法一般仅需要少量数据微调或者甚至不需要应用数据，在故障率较低的情况下通过较小的计算代价以达到较高的预测精度，具有很大的吸引力. 在硬件故障率较高的情况下，可靠性急剧下降，容错的成本也显著增加. 与上述挖掘模型自身容错能力的容错方法正交的一类技术则是进行冗余保护，为了避免直接冗余带来的巨大计算成本，很多工作进一步分析了深度学习模型内部脆弱性的差异[4-5,17,39-41]，从而通过差异性的保护方法降低冗余保护的成本[16-17,23-24,42]. 由于深度学

习计算引擎普遍支持分层计算的模式，模型层间脆弱性差异可以方便地适用于各种 DLA 架构，相应的深度学习的脆弱性分析工作以层间脆弱性差异分析为主[4,39-41]，选择性保护也以层间冗余为主，其既可以利用空间冗余也可以利用时间冗余实现. 理论上，更加细粒度的脆弱性差异分析[17]有助于更加高效的选择性保护，但目前仍然缺少 DLA 架构和电路支持.

综上可见，大多数面向深度学习的容错技术重点主要在电路、架构或者算法的某一个层次上进行，不同的技术有着显著的优点和缺点，使用场景也各有限制. 尽管也有一些容错技术涉及了不同层容错的配合以提高推理精度或者降低容错代价. 例如，Zhang 等人[28]提出的计算单元旁路技术可以和模型的训练结合，使得模型可以匹配特定的故障旁路设置，从而减少旁路导致的精度损失. 然而，总体上目前仍然缺少系统的面向软故障的容错 DLA 跨层优化设计方法. 为此，本文基于跨层芯片优化设计的思想，首先在算法层提出从神经元和比特位 2 个维度的敏感性分析，在此基础上分别在架构层和电路层探索细粒度的选择性冗余保护技术，在保障可靠性和性能的前提下最小化容错设计成本.

2 DLA 跨层容错设计

为了应对不同场景下容错 DLA 的多目标设计需求，我们提出了一个系统的容错 DLA 跨层优化设计框架，该框架融合电路层、架构层以及算法层的容错优点，在满足推理精度和性能的前提下最小化容错成本. 我们将首先介绍面向容错 DLA 的跨层优化框架，然后展开介绍算法层、架构层以及电路层对应的容错设计策略，最后展示自动化的跨层参数设计空间探索方法.

2.1 总体架构

面向容错 DLA 跨层优化的总体架构如图 1 所示，首先需要用户确定设计目标和设计约束，如性能、可靠性、资源开销等，其中可靠性通常以故障场景下的精度作为指标，一定程度上与模型的精度指标重叠，资源开销主要包括容错设计引入的额外芯片面积等.

给定设计目标和设计约束之后，框架在算法层通过分析获得不同神经元对于软故障的敏感性，以此作为依据将深度学习的神经元计算分为重要部分和普通部分. 相对于普通神经元计算，重要神经元计算对于软故障会更加敏感并导致更大的模型精度损失，因此需要更强的容错设计. 在此基础上，我们又

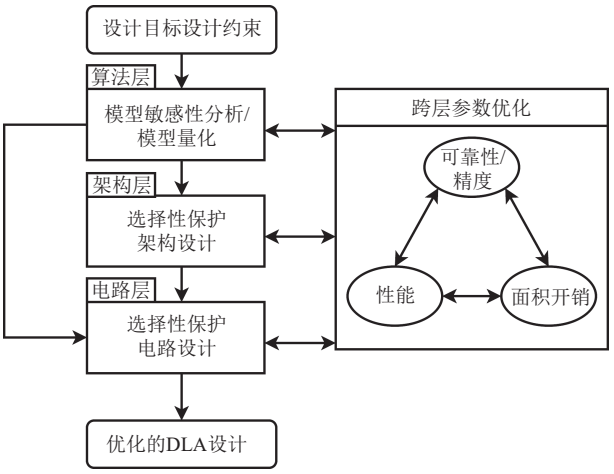


Fig. 1 Cross-layer fault-tolerant DLA design framework

图 1 深度学习处理器跨层容错设计框架

从神经元的比特位维度进一步细化了神经元计算对于软故障的敏感性，神经元的高比特位错误会导致更大的数值偏差，对模型的精度会产生更大的影响，因此高比特位的计算也需要相对更强的容错保护. 综上，针对算法层的容错设计，我们从神经元和数据比特位 2 个维度将深度学习模型计算分成重要部分和普通部分.

此外，模型量化是深度学习高能效处理的关键环节，使得模型可以采用更低的数据位宽进行计算. 为了支持定点量化，DLA 中基础的计算单元如乘加单元通常需要根据量化作出截断. 截断操作不仅会影响精度，也会直接影响计算单元输出数值不同比特位的计算逻辑规模，也会影响 DLA 电路层的选择性冗余保护成本，我们将在电路层冗余设计中详细介绍.

尽管我们在算法层将深度学习的处理分为重要的神经元计算和普通的神经元计算，但是由于通常的 DLA 架构设计采用流式计算方式，而且深度学习模型往往需要进一步分片(tiling)通过时分重用的方式在 DLA 上进行计算，这就使得我们很难将重要神经元计算和普通神经元计算分开以进行不同的容错保护. 此外，不同深度学习模型的重要性神经元计算的分布、同一个深度学习模型不同分块中重要神经元计算的分布都存在很大的差异，进一步增加了 DLA 架构上分开重要神经元计算和普通神经元计算的难度. 为此，我们扩展了 HyCA 架构[12,18]，提出了一个灵活可配置的异构 DLA 架构 FlexHyCA，FlexHyCA 采用常规的二维计算阵列处理普通神经元计算，同时，利用异构的点乘阵列单元(dot product processing unit, DPPU)并行处理少量对于可靠性有更高要求的重要的神经元计算. 通过实时加载重要的神经元在二维

阵列上的分布信息，分割 2 种不同神经元的计算. 此外 DPPU 可以根据重要神经元的比例，灵活选择重用二维阵列加载缓存的数据或者从 DRAM 直接读取需要的数据，以保证 DPPU 不会阻塞二维阵列的计算，消除对于 DLA 性能的影响.

针对深度学习模型中神经元的重要比特位和普通比特位对于故障敏感性的差异，在 FlexHyCA 的计算单元上进行了选择性的位保护电路设计，与电路层的区域保护和直接的 TMR 三模冗余保护策略不同，我们仅对于普通神经元的重要比特位以及重要神经元的重要比特位直接相关的逻辑电路进行了冗余保护，以减少容错设计的成本.

我们发现不同层的容错设计参数选择会互相影响，例如算法层重要神经元的比例会直接决定 FlexHyCA 架构中 DPPU 的尺寸，从而影响整体加速器的硬件资源开销. 类似的，神经元中重要的比特位选择也会对应需要保护的逻辑电路规模，最终影响电路层的位保护电路设计的成本. 模型的量化选择也影响计算单元的截断，从而影响位保护需要覆盖的电路规模以及容错成本. 重要神经元和普通神经元的比例也会影响各自重要比特位的数量，这些不同层之间的设计参数之间互相影响，导致容错 DLA 的参数手工优化非常困难，而且难以应对不同场景下用户在设计目标和设计约束上的差异性需求. 为此，我们引入了基于贝叶斯算法的设计空间探索机制，同时利用局部参数的相互关系对设计空间进行剪枝，实现快速的自动化跨层参数搜索.

2.2 算法层容错设计

我们首先分别从神经元计算和数据比特位 2 个维度分析模型对于软故障敏感性的差异. 神经网络模型可视为一个复杂的函数 $f(x)$，其中 x 代表输入数据. 对函数进行一阶泰勒展开，如式(1)所示，可以得到函数在输入数据附近受到的扰动造成的一阶误差 $f(x+\Delta x)-f(x)$ 正比于扰动大小 Δx 和梯度值 $f'(x)$，因此可以近似认为神经元敏感性与其在数据输入上的梯度相关. 梯度大的神经元，相应的数值扰动就可能会产生更大的数值波动，对于模型精度的潜在影响就更大，也可以认为梯度大的神经元对于故障的敏感性更高，我们将其定义为重要神经元.

$$f(x+\Delta x)-f(x)=\Delta x f'(x)+o(\Delta x). \tag{1}$$

类似的神经元重要性分析方法在文献 [17,43] 中也有应用. 首先将模型中神经元的梯度进行排序，将梯度值最高的 *S_TH*% 的神经元作为重要的神经元，详细的基于梯度的神经元重要性分析如算法 1 所示.

算法 1. 基于梯度的重要神经元选择算法.

输入：神经网络模型 *M*，数据集 *D*，重要神经元占比阈值 *S_TH*；

输出：重要神经元集合 *N*.

① *GetImportantNeurons*(*M*, *D*, *S_TH*);

② *gradients*←0; /* 初始化神经元的梯度 */

③ for each *input* in *D*

④ *output*←*inferenceModel*(*M*, *input*); /* 神经网络模型前向推理计算 */

⑤ *grad*←*backwardModel*(*M*, *output*); /* 神经网络模型返向传播计算梯度 */

⑥ *gradients*←*gradients* + *Abs*(*grad*); /* 累积计算输入数据在所有神经上的梯度 */

⑦ end for

⑧ *gradients*←*sortByDescent*(*gradients*); /* 按梯度总和大小从高到低排序 */

⑨*N*←*selectNeuronsByThreshold*(*gradients*,*S_TH*);/* 按 *S_TH* 比例选择梯度大的神经元作为重要神经元 */

⑩ return *N*.

由于高位数据位翻转造成的数值扰动大于低位数据翻转造成的数值扰动，同时神经元之间也区分为重要神经元和普通神经元. 为此，我们定义普通神经元的高 *NB_TH* 位为普通神经元的重要比特位. 重要神经元受到翻转的影响更大，需要保护的位数更多，重要神经元的高 *IB_TH* 位定义为重要神经元的重要比特位. 由于定点化深度学习模型的数据位宽有限，且高位比低位的重要性高，因此，对于相同类型的神经元我们总是优先保护高位. 对于重要神经元和普通神经元之间的折中则取决于模型的精度和位保护的成本. 在满足精度的前提下，选择保护代价最低的设置. 由于定点化模型的数据位宽有限，*IB_TH* 以及 *NB_TH* 设置的组合数量不大，采用简单的枚举算法确定 *IB_TH* 以及 *NB_TH*，其中精度通过故障注入实验来获得，位保护的成本也可以通过计算单元的逻辑综合获得，具体的位保护设计以及成本评估将在后续的电路层容错设计章节详细介绍，重要比特位的设置优化如算法 2 所示.

算法 2. 基于枚举的重要比特位评估.

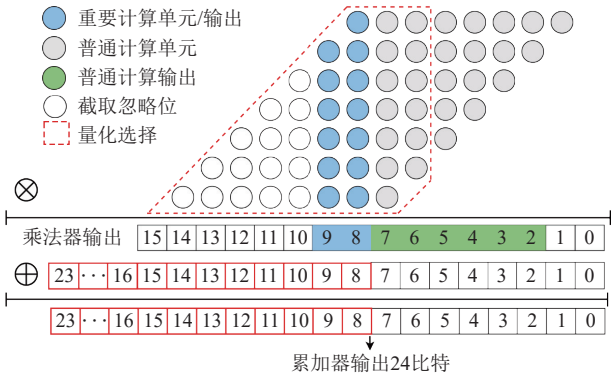
输入：神经网络模型 *M*，数据集 *D*，重要神经元 *N*，神经元数据位宽 *B*，模型精度目标 *ACC*；

输出：重要神经元的重要比特位为高 *IB_TH* 位，普通神经元的重要比特位为高 *NB_TH* 位.

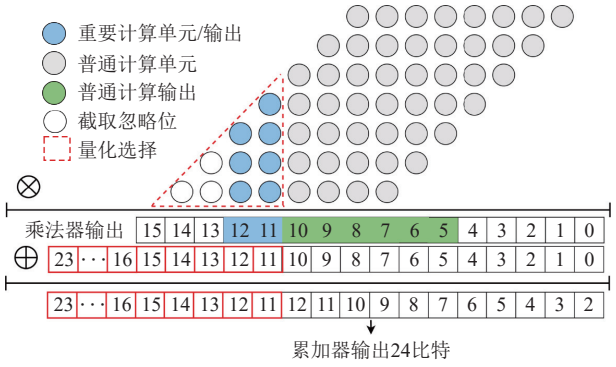
① *GetBitConfig(M, D, N, IB_TH, NB_TH, ACC)*;

② for *IB_TH* ranges from 1 to *B*
③ for *NB_TH* ranges from 1 to *B*
④ *acc*←*evaluateModel(M, N, D, IB_TH, NB_TH)*; /* 模拟评估重要神经元的高 *IB_TH* 位以及普通神经元的高 *NB_TH* 比特位完全无故障情况下模型的精度 */
⑤ *cost*←*getCost (M, N, IB_TH, NB_TH)*; /* 重要神经元的高 *IB_TH* 位以及普通神经元的高 *NB_TH* 比特位保护的面积开销 */
⑥ if *acc*>*ACC* 和 *cost*<*opt_cost* /* 如果精度满足需求 */
⑦ *opt_IB_TH, opt_NB_TH*←*IB_TH, NB_TH*;
⑧ *opt_cost* ←*cost*;
⑨ end if
⑩ end for
⑪ end for
⑫ return *opt_ib_th, opt_nb_th*.

量化是加速深度学习模型推理的关键优化技术，其不仅影响模型的精度，也会影响 DLA 计算单元中高位逻辑的区域. 对于定点化的 DLA，其计算单元的输出数据位宽通常比较大，以 8 比特 DLA 为例，计算单元的输出数据位宽至少为 16 比特，为防止溢出，很多 DLA 设计会选择大于 16 比特的设置，甚至使用 32 比特设置. 由于模型后续层的计算仍然是 8 比特，计算单元的输出就需要在计算过程根据模型的量化进行截断，这就意味着截断之外的数据位对模型的推理影响很小，相应地与这些比特位直接相关的逻辑电路的重要性也比较低. 类似地，输出数据中普通比特位对应的逻辑电路的重要性也相对较低. 然而，一般深度学习模型的量化选择没有限制，深度学习不同模型以及同一个模型不同层的量化选择都可能不同，这就导致计算单元中重要的逻辑电路是不同的. DLA 设计要保证最坏的情况，那么重要的逻辑电路比重就很大. 图 2 显示了不同量化选择对应的累加器以及乘法器的重要比特位的位置以及直接的计算逻辑区域. 假设输入数据位为 8 比特，乘法器输出数据位为 16 比特，累加器输出数据位为 24 比特，输出数据的高 2 位为重要的比特位. 当累加器截取的输出数据位置是第 2 到第 9 比特时，输出数据的高 2 位是重要比特位，其直接的计算逻辑对应于图 2(a)中的蓝色部分，其中乘法器对应的重要计算逻辑则包括一列 8 个 1 比特求和以及一列 7 个 1 比特求和. 在没有量化约束的情况下，乘法器输出的 6~15 比特都有可能是重要的比特位，需要保护的电路区域对应的逻辑是整个红色虚线标记的区域，就必然导致很高的冗余成本. 我们定义量化截取数据的最低位为 *Q_scale*，当我们设置 *Q_scale*=5，实质上限制量化使得截取的数据位范围缩小到第 5 比特和第 24 比特之间，乘法器的高 2 比特的范围是第 11 比特和第 15 比特之间，乘法器中需要保护的计算逻辑区域则可以大幅度缩小，如图 2(b)中的红色虚线区域. 当我们对重要的计算逻辑进行三模冗余保护的时候，量化限制可以显著减少冗余保护的成本.



(a) 无量化限制，累加器截断的数据可以位于24比特数据的任意连续位置



(b) 有量化限制，累加器截断的数据可以位于第5~23比特的任意连续位置

Fig. 2 The range of the important bits in accumulators and multipliers in DLA, as well as the directly correlated computational logic, varies substantially over the different quantization constraints

图 2 不同量化约束下，DLA 中累加器以及乘法器的重要比特位的范围以及直接关联的计算逻辑差异显著

2.3 架构层容错设计

为了支持选择性容错保护，DLA 架构设计面临的核心问题是如何在基础 DLA 架构上区分重要神经元计算和普通神经元计算并且为少量重要神经元的计算提供更强的容错保护. 特别是，架构设计还要能够容忍重要神经元分布的差异性. 重要神经元的分布差异主要体现在 3 个方面：1)不同深度学习模型的重要神经元的分布不同；2)同一个深度学习模型

不同层的重要神经元的分布也不尽相同；3)同一个深度学习模型同一个层的卷积也往往由于DLA资源受限需要进行切分并分时映射到DLA的计算阵列上去，而重要神经元在不同的任务切片上的分布仍然会出现差异.

为此，我们采用Liu等人[12]针对DLA任意位置计算单元硬故障容错提出的HyCA作为基础架构，实现重要神经元计算和普通神经元计算的分离. 由于大多数的神经元被分类为普通的神经元，我们使用HyCA的二维计算阵列处理普通的神经元计算，以更好地重用数据和权重. 对于少量稀疏分布的重要神经元，我们则将其动态加载到异构的点乘计算单元(dot product processing unit, DPPU)处理以充分利用神经元计算的内部并行. 由于HyCA的DPPU需要重用二维阵列专用缓存中的数据，当需要保护的重要神经元计算比例过高，DPPU会阻塞二维阵列的计算. 另一方面，重要神经元的计算比例存在较大的波动，最坏的情况设置DPPU尺寸会引入很大的面积开销. 为了解决这一问题，我们在HyCA的基础上为DPPU增加了更灵活的数据加载机制，在重要神经元计算比例过高的情况下，直接从DRAM加载需要的数据，避免了DPPU对于二维计算阵列阻塞导致的性能下降，而仅仅增加了少量的I/O；当重要神经元分布比较均匀以及FlexHyCA兼容原本的HyCA设计，优先复用二维阵列缓存中的数据，避免额外的I/O. 这样，新的FlexHyCA架构就能够更好地适应重要神经元分布差异性问题，具体的FlexHyCA架构如图3所示.

2.4 电路层容错设计

为了支持选择性容错保护，结合神经元不同比特位重要性的差异性，针对DLA中最基本的乘加计算单元提出了可配置的位保护设计，对于神经元重要比特位的逻辑计算提供三模冗余保护，避免软故障带来较大的计算偏差. 由于乘加计算单元的核心是乘法器部分，以基础的8比特乘法器为例介绍可配置的位保护乘法器. 如图4所示，乘法输出数据位宽为16比特，当累加器最终截断的8比特数据对应于乘法器输出的[m:n]位，那么乘法器最重要的高s位就对应于[m: m+s−1]位，相应地我们就将乘法器m-s+1列到m列的计算逻辑认为是需要保护的区域. 由于量化选择不同，m的位置是变化的，那么就需要保

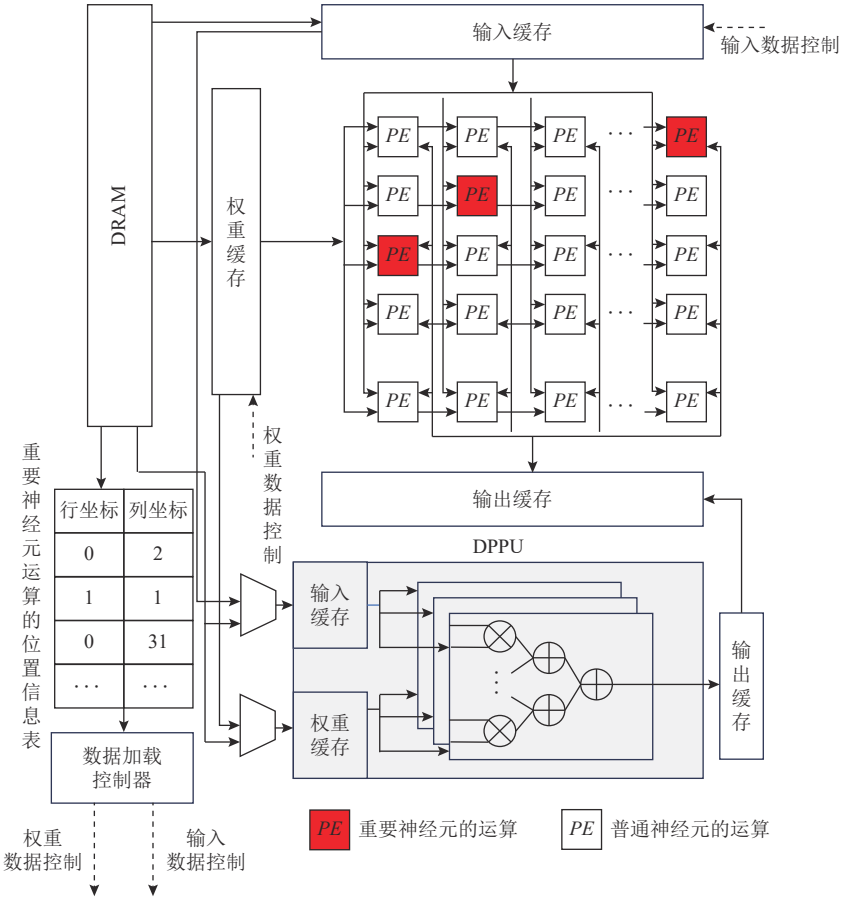
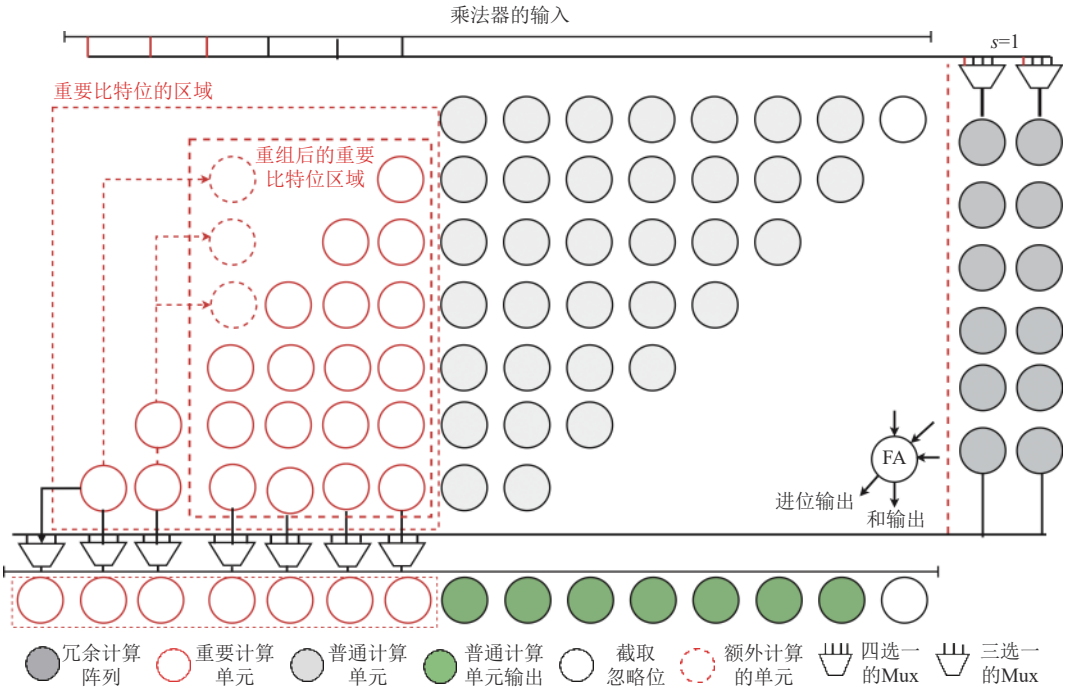
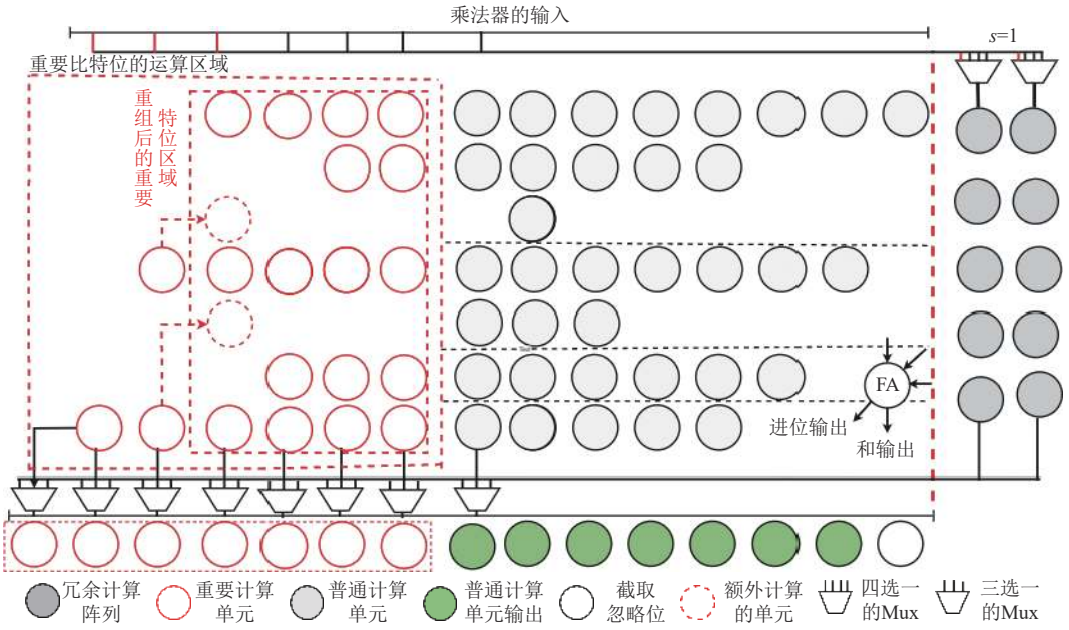


Fig. 3 FlexHyCA architecture

图3 FlexHyCA架构



(a) 移位乘法器位保护



(b) 华莱士树乘法器位保护

Fig. 4 Principle of configurable bit-protected multiplier

图 4 可配置的位保护乘法器原理

护图 4 中标识的整个红色区域. 然而, 对于一个具体的量化选择, 我们实际上仅仅用到了红色虚线区域中连续的 s 列计算. 为了降低开销, 仅需要对区域中最大的 2 列蓝色计算任务进行保护. 对于不同的量化选择, 需要对应地改变 Mux 的选择, 使得冗余单元可以保护对应的列. 然而, 红色虚线区域左侧的计算任务远远小于最大的 2 列蓝色计算任务. 当使用简单的 Mux 替换策略时, 在量化选择了左侧的两列计算时, 冗余阵列会出现很大的浪费, 而且由于冗余单元需要保护的计算单元数量很多, 冗余计算单元对应的信号都有很大的扇出, 也会引入额外的延迟以及面积. 为了解决这个问题, 对左侧部分相邻的计算单元进行合并, 这些单元中任何一列被选为重要的计算单元的情况下, 合并后的单元都会得到保护, 进一步提高了冗余单元的利用率, 并降低了冗余单元信号的扇出. s=1, Q_scale=2 时, 如图 4 所示, 最左侧的

3 列计算单元冗余单元的最大扇出从 6 降低为 4.当量化导致最重要的 1 列位于左侧区域时, 冗余计算也可以用于保护更多的计算列. 尽管图 4 仅仅展示了普通移位乘法器的效果, 实际上位保护策略也可以用于其他结构的乘法器. 图 4(b)展示了将位保护策略应用于华莱士树乘法器的示例.

2.5 跨层设计空间探索

由于不同层的设计参数互相关联, 各个层次分别优化难以达到最优的设计目标. 为此, 我们构建了统一的跨层优化空间以协同优化不同层的设计参数. 我们将指定故障率下模型的精度要求和性能损失作为用户的设计约束, 最小化冗余保护策略需要引入的额外芯片面积. 跨层优化框架的参数选择包括 3 部分:

1)算法层的设计参数. 模型中重要神经元的百分比 *S_TH%*; 重要神经元的选择策略 *S_policy*(层间统一比例、分片统一比例); 重要神经元的重要比特位数量 *IB_TH*; 普通神经元的重要比特位数量 *NB_TH*; 量化截断参数 *Q_scale*.

2)架构层设计参数. FlexHyCA 的二维阵列尺寸 *Array_size* 和点乘阵列尺寸 *Dot_size*、异构阵列数据重用与否 *Data_reuse* 等.

3)电路层设计参数. 计算单元的位冗余实现策略 *PE_policy*(直接冗余, 可配置冗余)

通过上述将不同层的优化参数进行解耦, 可以进一步帮助我们对设计空间进行参数化的描述.

面向容错 DLA 的跨层参数设计空间很大, 而且很多参数会对不同层都产生影响, 手工进行优化难以满足复杂的设计目标, 同时设计效率也比较低. 为了解决跨层参数优化问题, 首先对跨层参数优化问题进行形式化. 如式(2)所示, 所有的设计参数统一用向量 $\boldsymbol{V}$ 表示, 其各个分量分别对应不同层的设计参数. 我们将性能、可靠性、精度作为约束, 将典型故障率设置下模型的推理精度作为可靠性和精度指标, 将最小化冗余设计带来的芯片面积作为优化目标.

$$\begin{aligned}&\boldsymbol{V}=(S\_TH, IB\_TH, NB\_TH,\\&Q\_scale, S\_policy, Array\_size, \cdots),\\&\arg\min_{v\in\boldsymbol{V}} Area,\\&\text{s.t.}\quad Acc_{\text{high}}\geqslant 0.97\times Acc_0,\\&\quad Acc_{\text{low}}\geqslant 0.95\times Acc_0,\\&\quad Perf\leqslant 1.10\times Perf_0,\\&\quad Bandwidth\leqslant 1.10\times Bandwidth_0.\end{aligned}\tag{2}$$

为了解决面向容错 DLA 的跨层参数优化问题, 采用贝叶斯优化算法作为基础的空间探索方法, 实现如算法 3 所示, 我们额外增加了最大迭代次数 *ITER_MAX_ STEP* 限制, 每次迭代都需要评估对应配置的可靠性、精度、性能以及冗余芯片面积. 利用 Scale-Sim[44] 评估性能以及使用 Synopsys Design Compiler 评估基于位保护的计算单元面积, 由于设计空间中大量的位保护选择是重复的, 预先对于可能的位保护设计的代价进行了离线评估以构成面积开销表, 在贝叶斯优化过程中, 实际上是通过查表快速获得不同配置对应的 FlexHyCA 的面积以及通过自研的故障注入模拟器评估不同故障率下模型的精度. 为了加快设计空间的搜索, 也通过经验参数提前剪枝设计空间中的部分参数. 例如更多的位保护会提高模型的容错能力, 在同等故障率下其对应的精度更高, 同时也会引入更大的冗余保护成本. 本质上, 位保护参数对于精度和芯片冗余基本上都是单调递增的. 类似地, 其他设置不变的情况下, 重要神经元的比例与精度和冗余成本之间也是单调递增的关系, 这样, 经验信息可以用于快速剪枝不符合约束的设计选择. 例如, 当位保护设置违背了精度或者可靠性要求, 那么更少的位保护也一定会违背精度或者可靠性要求, 这样可以无需对这样的参数进行评估就可以直接剪枝这样的设计参数.

算法 3. 跨层设计空间探索算法.

输入: 跨层设计参数空间 $\boldsymbol{V}$, 设计约束 R;

输出: 最优参数选择 v.

① *BayesDesignOpt*($\boldsymbol{V}$, R);

② $v\leftarrow$*sample*($\boldsymbol{V}$), *step*$\leftarrow$0; /* 随机化初始设计参数 */

③ while *step* < *ITER_MAX_ STEP*

④ *Area*, *Acc*, *Perf*$\leftarrow$*getDesignVal*(v); /* 获取当前设计参数的性能和开销数据 */

⑤ $v\leftarrow$*bayesOptStep*(v, $\boldsymbol{V}$, *Area*), *step*$\leftarrow$*step*+1; /* 单步贝叶斯优化调整参数 v */

⑥ if not *meetRestriction*(*Acc*, *Perf*, *Bandwidth*, R)/* 如果不满足设计约束 */

⑦ $v\leftarrow$*bayesOptPurning*(v, $\boldsymbol{V}$);

⑧ end if

⑨ end while

⑩ return v.

3 实验结果及分析

3.1 实验设置

本节主要从用户优化目标与约束、数据集与模

型基准、硬件实现配置、软件仿真配置等方面介绍实验设置.

1)用户优化目标与设计约束. 参考文献 [4,14] 等工作, 使用位翻转率(bit error rate, BER)描述软故障概率, 尽管 BER 一定程度上代表片上每个存储单元含缓存与寄存器每个周期发生单粒子翻转的概率, 实际上是一个抽象的面向深度学习应用的故障率, 与具体的目标硬件存在一定的差异. 针对故障率 I(*BER*=1E–4)和 故障率 II(*BER*=2E–4)这 2 种场景, 我们分别设置了 2 种不同的可靠性/精度约束, 相对于无故障的深度学习模型, 在故障率 I 下精度损失小于 3%, 在故障率 II 下精度损失小于 5%、性能损失小于 10%、带宽损失小于 10%. 在满足精度、性能和带宽约束的前提下最小化芯片面积开销.

2)数据集与基准模型. 实验中我们使用 ImageNet 作为数据集, 使用 VGG16 和 Resnet50 作为典型的深度学习模型的测试基准, 这 2 个模型都采用了 8 比特整型量化, 量化后的模型精度分别为 72.95% 和 75.96%.

3)硬件实验配置. FlexHyCA 基础设计中二维计算阵列固定为 32×32, 权重缓存为 512 KB, 数据缓存为 256 KB, 其中基础计算单元使用华莱士树实现乘法, 累加器数据位宽为 24 比特. 针对用户约束和目标设计的 FlexHyCA 二维阵列参数固定不变, 一维阵列的缓存和计算阵列尺寸由优化工具决定. 采用 Verilog 建模 FlexHyCA, 在 TSMC 65 nm 工艺下, 使用 Synopsys DC 工具综合获得不同配置下的芯片面积.

4)软件实验配置. 在 SCALE-Sim[44] 的基础上针对 FlexHyCA 架构实现了性能模拟, 用于评估不同硬件设置下深度学习的性能. 我们在 PyTorch 框架下实现了故障注入模拟, 用于评估不同故障率下深度学习模型的精度损失. 为了方便故障注入在 PyTorch 上的实现, 我们参考文献 [4, 14, 17, 22, 45−48] 等工作, 主要针对神经元和权重进行随机的位翻转故障注入.

5)容错 DLA 对比实验设置. 为了验证提出的跨层优化 DLA 容错设计方法, 从硬件可靠性(精度)、硬件资源开销、性能 3 个维度, 将提出的设计方法与经典的不同层单独保护的方法进行了对比. 包括基础的 DLA 设计(Base)、电路层选择性三模冗余设计(TMR-CRT1, TMR-CRT2, TMR-CRT3)、架构层选择性三模冗余设计(TMR-ARCH)、算法层选择性三模冗余设计(TMR-ALG), 以及本文提出的跨层冗余设计(TMR-CL). 其中电路层三模冗余设计主要是针对 DLA 的所有基本计算单元, 文献 [19] 电路层的保护方法仅保护高比特位运算部分. 选择了高 1 比特三模冗余(TMR-CRT1)设计、高 2 比特三模冗余(TMR-CRT2)设计, 以及高 3 比特三模冗余设计(TMR-CRT3). TMR-ARCH 主要是根据层间敏感性差异, 对故障敏感的层利用三模冗余保护, 三模冗余通过空分复用的方式实现, 并实时进行投票选择, 仅仅需要将基础 DLA 分成三等分, 并增加投票逻辑, 对于故障相对不敏感的层则正常计算. TMR-ALG 则不需要改变 DLA 架构, 通过时分复用的方式重复执行对故障敏感的深度学习层, 可以在深度学习加速器上也可以在通用计算平台上便利地实现[22]. TMR-ALG 本质上和 TMR-ARCH 类似, 但 TMR-ARCH 也可以用于保护关键的控制通路[39]. 然而, 由于本实验主要关注数据通路的保护, 无法充分评估 TMR-ARCH. 本文提出的跨层优化设计将在 5.6 节详细介绍.

深度学习的不同层对于故障的敏感性差异是很多选择性冗余保护策略的关键依据[45-46], 首先通过故障注入实验, 获得了 VGG16 以及 ResNet50 不同层的敏感性对比. 在指定故障率下, 将一层深度学习模型在完全保护的情况下相对于整个模型都未保护的情况下获得的精度提升作为该层的敏感性. 为了简化分析, 将 ResNet50 的一个块(block)当做整体进行分析. 图 5 展示了在故障率 I 和故障率 II 情况下 VGG16 和 ResNet50 不同层的敏感性数据, 可以看到不同层的敏感性差异显著, 敏感性最高的层相对于最低的层相差超过 10 个百分点.

根据模型层间的敏感性差异, 进一步分析了按

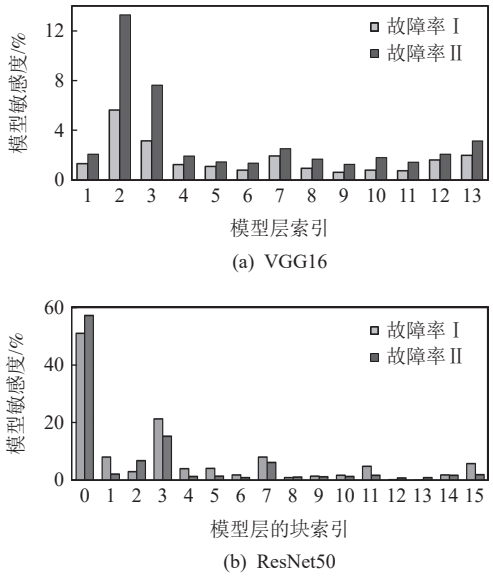
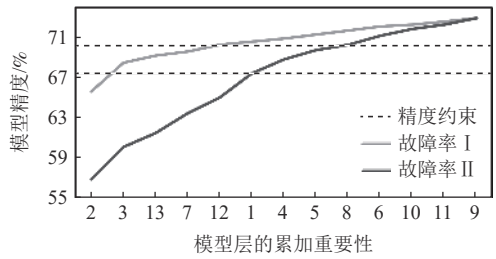


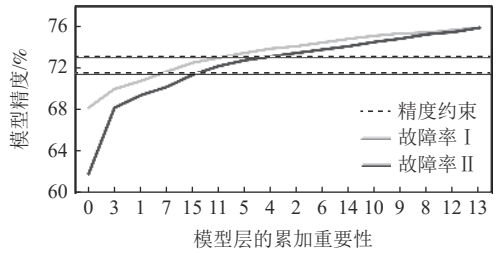
Fig. 5 Sensitivity of different layers/blocks of deep learning models under various fault rates

图 5 不同故障率下深度学习模型不同层/块的敏感度

照敏感性差异逐层保护带来的模型精度的提升. 精度提升曲线如图 6 所示，从中可以看到，精度提升速度开始比较迅速，后面则比较平缓，也进一步表明根据敏感性分析进行选择性容错的有效性. 此外，根据图 6 我们也可以根据用户可靠性/精度需求确定至少需要保护的模型层的集合. 实验中，TMR-ARCH 与 TMR-ALG 都是依据上述的敏感性来决定需要优先保护的深度学习模型层或者块的集合.



(a) VGG16



(b) ResNet50

Fig. 6 Mode accuracy variation caused by the gradual protection of layers/blocks at different fault rates

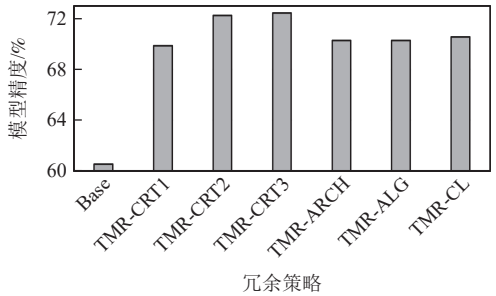
图 6 不同故障率下敏感度逐层/块保护带来的模型精度变化

3.2 实验总体分析

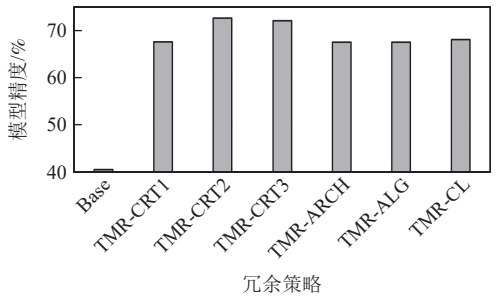
我们从电路、架构以及算法这 3 个层次分别选择不同的保护策略，并观察其在不同的故障率以及不同模型设置下的保护效果，然后，从可靠性/精度、性能、资源开销 3 个方面分别与基础设计以及本文提出的跨层设计进行对比.

1)可靠性/精度. 如图 7 所示，我们对比了实验设置中典型的容错 DLA 设计分别在故障率 I 以及故障率 II 下的精度，总体上，不同的容错策略都能满足精度需求. 但是 TMR-ARCH, TMR-ALG, TMR-CL 等设计相对比较灵活，因此可以搜索得到正好满足用户精度约束的设置. 相比之下，TMR-CRT 粒度比较粗、精度差异比较明显，TMR-CRT1 未能满足故障率 II 下的精度约束，TMR-CRT2 以及 TMR-CRT3 又超过了用户精度的需求.

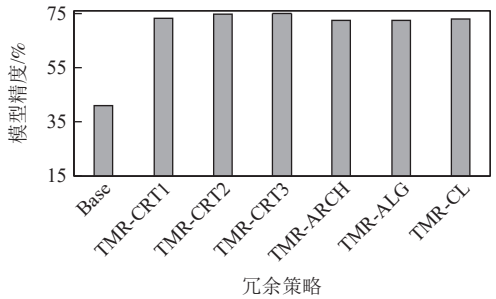
2)性能. 性能数据如图 8 所示. TMR-CRT 不改变 DLA 的架构以及上层软件，不会导致性能受损. TMR-CL 选择重要神经元在层间分布均衡的策略，因此重要神经元的重计算和 DPPU 的尺寸匹配，而分块的差异问题可以通过少量的 I/O 代价解决. 本质上，重要神经元比例高的分块可以被重要神经元比例低的分块所均摊，因此总体上对于性能的影响也可以忽略不计. 相比之下，架构层和算法层则分别是通过空分复用以及时分复用来进行选择性冗余，引入的硬件



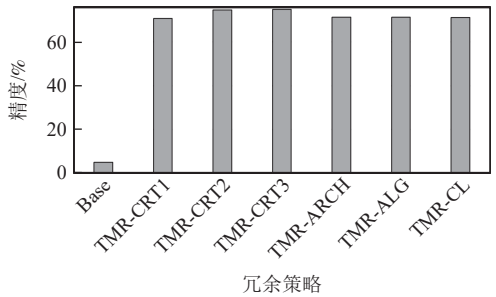
(a) 故障率 I 下VGG16精度



(b) 故障率 II 下VGG16精度



(c) 故障率 I 下RetNet50精度



(d) 故障率 II 下RetNet50精度

Fig. 7 Model accuracy after different fault-tolerant DLA design strategies optimization

图 7 不同容错 DLA 设计策略优化后的模型精度

成本少或者没有，但是敏感度高的层理论上性能会因三模冗余降低为原来的 1/3. 敏感的深度学习模型层占比较高，最终导致运行时间增加接近 1 倍，无法满足设计约束.

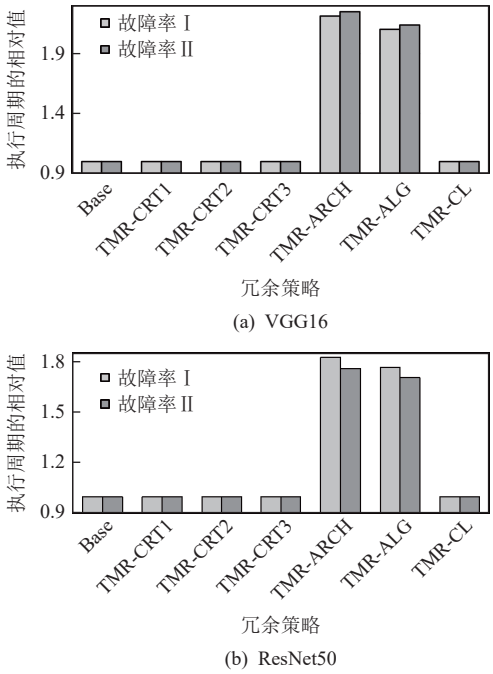


Fig. 8 Excutive time of the model after different fault-tolerant DLA design strategies optimization

图 8 不同容错 DLA 设计策略优化后的模型执行时间

3)硬件资源开销. 我们评估了典型的容错 DLA 设计引入的额外芯片面积开销，本文主要关注了计算阵列的容错，在没有特别说明的情况下，芯片面积开销也仅考虑计算阵列部分. 为了方便比较，额外的芯片面积开销都相对于原始未保护的计算阵列面积做了归一化，实验结果如图 9 所示. TMR-ALG 本质上采用时分复用方式对敏感的层进行冗余保护，不会额外引入硬件成本. TMR-ARCH 则主要是在架构层利用空分复用的方式对于敏感的层进行冗余，需

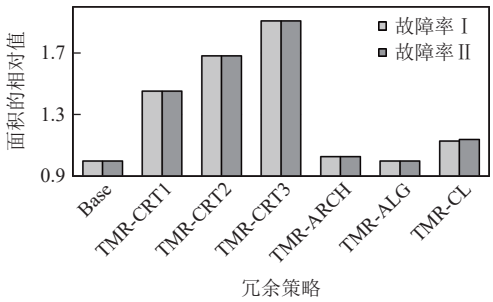


Fig. 9 Relative area of chip corresponding to different fault-tolerant DLA

图 9 不同容错 DLA 对应的芯片相对面积

要少量的控制器和选择器，因此其面积相比于 Base 有少量提高. TMR-CRT 尽管只需要保护高位，但是高位的具体位置和量化相关，在不限制量化的前提下，对应的电路逻辑区域很大，这就导致电路层的保护成本仍然很高. TMR-CL 综合考虑不同层参数之间的互相影响，一方面通过量化限制大幅度减少了电路层的成本；另一方面，通过 FlexHyCA 架构将少量重要神经元的计算单独分出来并进行更严格的保护，尽管面积比 Base 和 TMR-ARCH 大，但是相对来说仍然是可以接受的.

此外，TMR-CL 会受到重要神经元分布不均的影响，导致 FlexHyCA 需要为 DPPU 重复加载依赖的数据和权重，从而会引入额外的 I/O.由于其他的设计方案，没有额外的 I/O 增长，因此将单独分析额外的带宽消耗. 在 TMR-CL 中 I/O 的增长主要来源于 2 方面：1)当重要神经元在一个分块计算中占据的计算单元的比例超过 DPPU 的算力，我们不再重用二维阵列缓存的数据，而是直接从 DRAM 加载需要的数据，避免对于二维计算阵列的阻塞. 2)需要在编译阶段为重要神经元所在的二维阵列计算单元的位置以使得 FlexHyCA 可以选择性重计算. 最终，对于 VGG16 和 ResNet50 模型，TMR-CL 额外引入的 I/O 相对于模型自身权重数据的 8.2% 和 9.9%，不会对整体 DLA 设计的 I/O 产生严重的影响.

3.3 算法层参数分析

为了更好地理解算法层核心参数之间的关系，我们仅以 ResNet50 为例分析了算法层重要神经元的比例与神经元重要比特位的数量之间的关系，以及模型量化约束对于模型精度的影响.

对于重要神经元的比例与神经元重要比特位的数量之间的关系，我们将重要神经元的比例设置为 S_TH={0.02, 0.05, 0.10, 0.15, 0.20, 0.25, 0.3, 0.35, 0.4}等 9 种，对于重要神经元的重要比特位和普通神经元的重要比特位，我们设置{⟨2,1⟩,⟨3,1⟩,⟨4,1⟩,⟨3,2⟩,⟨4,2⟩,⟨4,3⟩}6 种组合. 实验结果如图 10 所示，位保护较少的时候，重要神经元的比例总体上对精度能够产生较大的影响，精度随着重要神经元比例的增加持续提高. 然而，在故障率 I 的场景下，精度的增加呈现出阶段性增长的特点. 重要神经元的比例从 2% 增长到 5% 的时候，精度有比较明显的提高，然而，当比例从 5% 增长到 20% 的过程中，精度增长不多；当比例增长到 25% 之后，精度开始有明显增长. 实质上，这表明模型只需要少量的高比特位和重要神经元就能达到比较高的精度，但是进一步的精度提高，就需

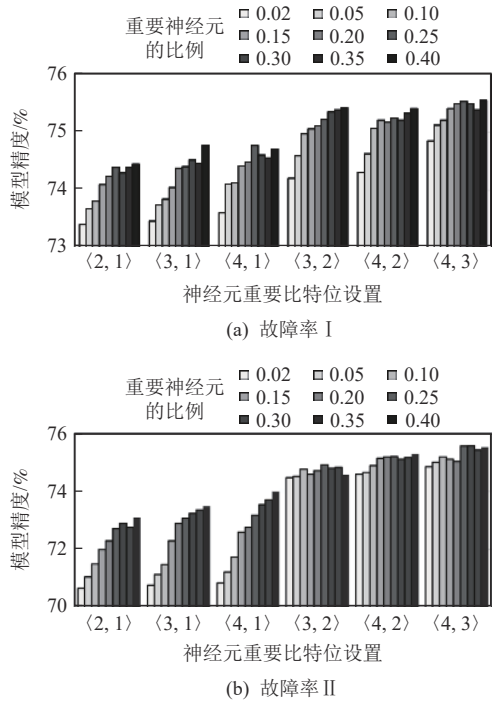


Fig. 10 Influence of the proportion of important neurons and the number of important bits of neurons on accuracy of ResNet50 model

图 10 重要神经元的比例与神经元重要比特的数量对 ResNet50 模型精度的影响

要更多的比特位以及重要神经元, 也意味着更多的保护成本. 当位保护较多的时候, 重要神经元比例的作用变得非常微弱, 本质上是由于深度学习模型的高比特位得到充分保护之后, 无论是重要神经元还是普通神经元, 低比特位部分的故障对于精度的影响都很小, 但是对于达到理想的精度仍然有着巨大的作用. 总体上, 当用户对于可靠性或者精度要求不是很高的情况下, 选择较少比例的重要神经元具有较高的性价比. 对于精度要求很高的情况下, 选择更多的比特位保护总体上价值更高, 相对来说我们仍然倾向于选择较少比例的重要神经元.

对于模型量化约束和模型精度之间的关系, 为了简化分析, 我们将整个模型做了统一的量化约束. 实验设置累加器数据位宽为 24 比特, 没有约束的情况下, 截取的数据区间为 0~23 比特的任意连续位置. 当增加了量化约束之后, 将截取数据的最低位设置为 *Q_scale*. 图 11 显示了 *Q_scale* 对于模型精度的影响, 随着 *Q_scale* 的增加, 量化选择的空间越来越小, 相应的模型精度也会受到影响. 从图 11 中可以发现, 当 *Q_scale*<7 时, 模型精度下降很少, 根据第 2.2 节的讨论可以看到, *Q_scale* 的提高可以有效地减少重要比特位对应的逻辑电路规模, 这就意味着, 在精度允许的范围内仍然存在很大的降低选择性位保护成本的空间.

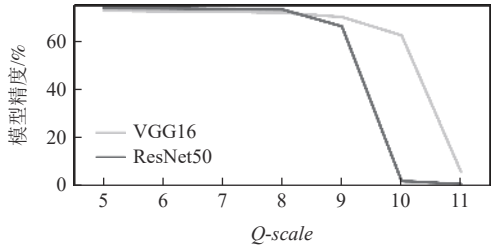


Fig. 11 Influence of *Q-scale* on model accuracy

图 11 *Q-scale* 对于模型精度的影响

3.4 架构层参数分析

为了帮助理解架构层的主要设计参数之间的关系, 我们将 FlexHyCA 的二维阵列固定为 32×32, 然后, 分析了不同 DPPU 尺寸以及重要比特位设置下芯片面积的变化, 芯片面积都相对于无保护的二维计算阵列芯片面积做了归一化. 实验结果如图 12 所示, 尽管 DPPU 采用了比二维阵列更多的位保护, 但是 DPPU 中的计算单元通常远小于二维阵列, 因此整体上引入的芯片面积比重比较低, 相比之下, 二维计算阵列计算单元数量多, 随着位保护的增加, 引入的芯片面积增加明显. 因此, 在参数优化过程中, 重要神经元的位保护设置可以更大一些, 这样也不会导致显著的芯片面积增加. 相比之下, 二维计算阵列的位保护设置要尽量低, 以控制整体的冗余保护成本. 此外, DPPU 过大就不可避免地会引入较大的芯片面积开销, 但 DPPU 过小又限制了重要神经元比例. 本质上, 只有当 DPPU 提供的选择性保护降低的二维阵列位保护的需求可以覆盖 DPPU 本身的面积开销的情况下, DPPU 的引入才会有比较好的收益.

我们为 DPPU 提供了直接的数据访存通路, 避免了 DPPU 只能重用二维阵列缓存中的数据导致的阻塞问题, 但这又会引入额外的访存. 同时, FlexHyCA

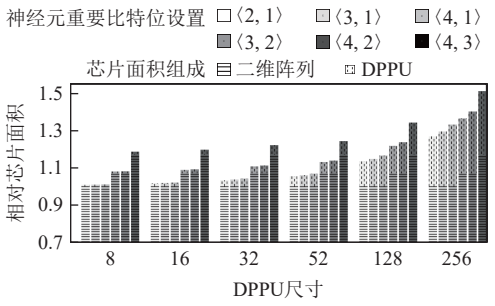


Fig. 12 Influence of DPPU size and bit protection configurations on DLA chip area

图 12 DPPU 尺寸和位保护设置对 DLA 芯片面积的影响

设计中需要较多的重要神经元位置信息表，也会导致额外的访存. 为此，我们在不同的重要神经元比例设置下进一步评估了 I/O 的增加，并相对于权重数据做了归一化，由图 13 中可以看出，额外的 DRAM 数据访问，是随着重要神经元的阈值设计线性变化的，因此，额外的数据访问记录该部分重要神经元的位置. 当神经元的阈值设计 S_TH=0.1 时，额外的数据加载超过了 10%，不符合我们最初的设计要求，因此，额外的数据加载也对神经元的阈值设置有一定的限制.

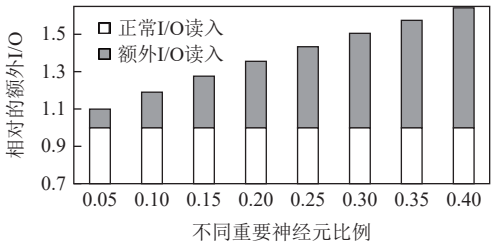


Fig. 13 Influence of the proportion of important neurons on I/O of FlexHyCA

图 13 重要神经元的比例对 FlexHyCA 的 I/O 影响

3.5 电路层参数分析

为了理解电路层位保护设计参数，我们分析了无量化约束下不同位保护对 8 比特整型乘法器芯片面积的影响，以及不同量化约束下不同位保护对于 8 比特整型乘法器芯片面积的影响. 对于量化约束，我们将 Q_scale 分别设置为 4 和 7，由于量化约束设置对于 ResNet50 以及 VGG 的精度影响很小. 对于带约束的位保护设计，我们进一步对比了直接冗余和可重构冗余实现，其中直接冗余的方法是直接对整个重要的逻辑区域进行三模冗余，而可重构冗余则是使用 Mux 根据量化的选择仅保护重要的比特位逻辑. 实验结果如图 14 所示，相对于没有约束的高比特位冗余设计，本文提出的带有约束的容错保护面积显著降低，相比于直接冗余实现，冗余面积平均下降了 71.4%，其根本原因在于乘法器中只有中间的几列计算比较多，相对的冗余成本比较高. 当我们引入量化约束，少量的约束即可以绕开这部分计算的冗余，从而显著降低冗余成本. 对比直接冗余，可重构冗余可以减少冗余计算单元的数量，进一步降低冗余成本. 此外，无约束的冗余策略在位保护数为 1 的情况下，也会引入很大的面积开销，这是因为无量化约束的情况下，高比特位可能的逻辑规模仍然很大. 相比之下，带约束的冗余成本随着位保护数量的增加更加平稳，有利于设计的可扩展性.

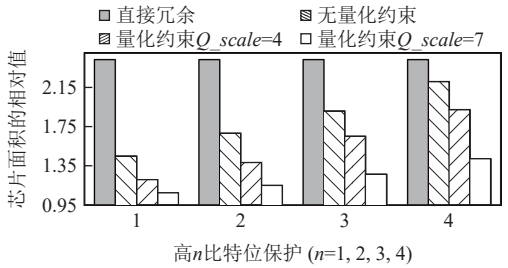


Fig. 14 Computing array area corresponding to different bit protection configurations

图 14 不同的位保护设置对应的计算阵列面积

3.6 跨层设计参数搜索

对于跨层设计参数搜索，我们主要在设计空间内，利用贝叶斯优化算法搜索跨层设计参数，在满足精度和性能约束的前提下最小化芯片面积的开销，主要的设计参数及其取值范围如表 1 所示. 图 15 分别展示了不同故障率下采用贝叶斯搜索的数据点以及对应的帕累托曲线. 总体上，可以看出不同的设计参数对应的芯片面积开销差异很大，面积最大的参数对应的芯片面积接近于 3 倍的优化参数对应的芯片面积，从一个侧面也表明跨层参数优化的必要性. 同时，我们从图 15 中可以发现，2 种故障率下，模型精度损失分别在 3% 和 5% 时，跨层优化中很多数据点相应的芯片面积普遍比较小，相对于传统的三模冗余具有显著的优势. 然而，随着模型精度要求的提高，跨层优化搜索得到的数据点对应的芯片面积大幅度增加，很多设计选择对应的芯片面积超过了原始芯片面积的 2 倍，这也表明容错 DLA 的精度要求提高之后，对应的容错成本急剧增长. 即使我们仅仅关注帕累托曲线上的设计选择，也可以发现类似的情况. 当可靠性/精度要求较低的情况下，相对的芯片面积开销可以控制在 5% 以内，然而，当可靠性/精度

Table 1 Search Space of the Cross Layer Design Parameters

表 1 跨层设计参数的搜索空间

参数	取值范围	含义
S_TH	{5%,10%,15%,20%,25%,30%,35%,40%}	重要神经元百分比
IN_TH	{2,3,4}	重要神经元重要比特位数量
NB_TH	{1,2,3}	普通神经元重要比特位数量
Q_scale	[1, 2, ⋯, 16]	截断约束
S_policy	{层间统一比例，分片统一比例}	重要神经元的选择策略
Dot_size	[8,16, ⋯, 256]	点乘阵列尺寸
Data_Reuse	{True, False}	异构阵列数据重用
PE_policy	{直接冗余可配置冗余}	计算单元位保护策略

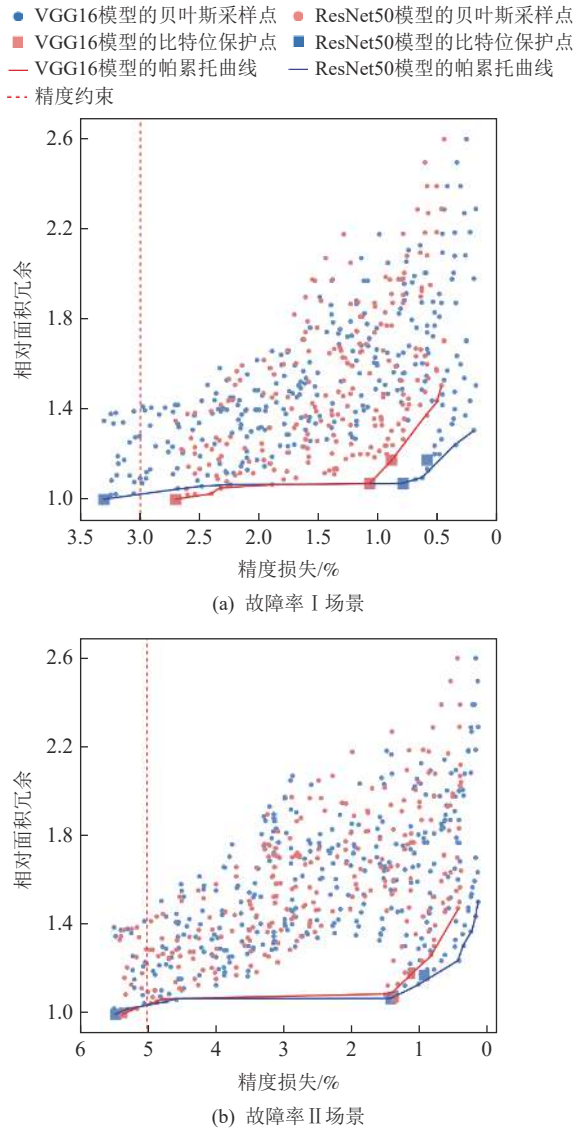


Fig. 15 Sampled data points in Bayesian optimization process

图 15 贝叶斯优化过程中采样的数据点

要求提高，优化的芯片面积开销则达到 40% 以上. 本质上，伴随着精度要求的提高，面积冗余成本开始的时候增加比较平缓，这是因为在该范围内的点，二维计算阵列采用高 1 位保护，大多数的重要神经元的阈值 *S_TH*% 设置得比较小(基本小于 10%)且采用的是可配置的冗余方式，在可配置的冗余方式下，保护重要神经元的不同比特位所带来的面积冗余成本差距不大，DPPU 的面积相比于二维计算阵列面积很小. 但是，随着精度要求的进一步提高，需要保护重要神经元的阈值或者数据位进一步提高，DPPU 的面积相比于二维计算阵列的面积不可忽略，所需要的面积冗余成本会显著增加. 因此，合理的深度学习可靠性和精度要求对于最终的容错设计选择有着巨大的影响.

此外，我们发现通过量化与电路层的协同设计使得电路层的容错成本大幅度降低，在不区分重要神经元和普通神经元的场景下，很大程度降低了容错成本并缩小了跨层优化的空间. 我们在图 15 中特别用方块标识出高 1 比特保护、高 2 比特保护以及高 3 比特保护对应的数据点，可见它们要么处于帕累托曲线上，要么非常接近帕累托曲线. 特别地，当模型精度要求位于高 1 比特保护和高 2 比特保护之间的很多设计参数被高 2 比特保护的设计所屏蔽，导致帕累托曲线中间出现很大的空白.

根据实验的不同设计目标，相应的优化参数结果如表 2 所示. 总体上，不同故障下由于很多设计参数存在比较强的相关性，例如重要神经元的比例(*IN_TH*)和点乘阵列的尺寸(*dot_size*)是一致的，过高的重要神经元比例会极大地增加芯片面积，这导致这 2 个参数选择空间比较小，也影响了一些相关的参数选择，同时正如前面分析的那样，一方面电路层和量化的协同设计极大地降低了电路位保护的成本，显著压缩了搜索空间. 另一方面重要神经元的比例以及 *Dot_size* 的参数都是离散化的，也一定程度限制了搜索空间，这些因素使得参数搜索的结果比较接近.

Table 2 Optimized Crossing Parameters Design of Different Models

表 2 不同模型的最优跨层参数设计

参数	故障率 I	故障率 II
S_TH	5%	5%
IN_TH	2	3
NB_TH	1	1
Q_scale	7	8
S_policy	分层统一比例	分层统一比例
Dot_size	52	52
Data_Reuse	True	True
PE_policy	可配置冗余	可配置冗余

4 结 论

深度学习加速器已经成为主流深度学习推理的计算引擎之一，其可靠性是保障高可靠深度学习的关键. 高可靠深度学习加速器在传统性能、能效、面积等基础指标之上进一步增加了可靠性的需求，设计空间大、优化困难. 针对高可靠深度学习加速器设计，本文提出从神经元计算以及神经元比特位 2 个

维度挖掘深度学习处理对于芯片软故障敏感性的差异，并根据这些差异从算法、架构以及电路设计3个层次提出对应的选择性保护方法；同时系统地探索了跨层设计参数之间的关系；最后，借助贝叶斯优化实现了跨层参数优化，可以为不同的用户需求提供自动化的跨层容错DLA设计，在满足可靠性/精度、性能等约束下最小化容错保护成本. 实验表明，相对于传统以单个层面为主的容错设计，跨层容错DLA更好地将深度学习特征与架构和电路设计结合，极大地降低了深度学习加速器的容错成本.

本文系统地探索了深度学习加速器在算法、架构以及电路层上的选择性容错保护方法，但仍存在3个方面的不足，未来继续完善.

1)本文提出的加速器架构中的二维计算阵列目前无法分割普通的神经元计算和重要的神经元计算，本质上重要的神经元在二维阵列和DPPU上都进行了处理，导致了计算的浪费，同时也限制了DPPU的规模. 未来可以借助稀疏化深度学习加速器架构从二维计算阵列中剔除重要神经元的计算，可以进一步降低容错成本.

2)目前我们采用了静态的方法切分重要的神经元计算和普通的神经元计算，缺乏对于输入数据差异性的考量，未来希望进一步增加对动态神经元重要性的支持，也有助于进一步降低容错成本.

3)算法层仍然存在很多其他的冗余保护方法，例如容错训练、基于Checksum机制的故障检测和纠错等，特别是在低故障率下与本文提出的容错设计策略存在一定的交叠区域，可以纳入到本文提出的跨层容错设计框架中，有望获得更好的容错设计方案.

作者贡献声明：张青完成了电路设计以及实验部分，并撰写论文；刘成提出了总体的设计方案并修改了论文；黄海同完成了重要性分析和跨层参数空间探索的算法设计；刘波、王颖、李华伟、李晓维提出指导意见并修改论文.

参 考 文 献

[1] Seo-Hyun J, Jin-Hee C, Yangjae J, et al. Automotive hardware development according to ISO 26262[C]//Proc of the 13th Int Conf on Advanced Communication Technology (ICACT). Piscataway, NJ: IEEE, 2011: 588−592

[2] Cheng Liu, Zhen Gao, Siting Liu, et al. Special session: Fault-tolerant deep learning: A hierarchical perspective[C]//Proc of the 40th VLSI Test Symp (VTS). Piscataway, NJ: IEEE, 2022: 1−12

[3] Rabe M, Milz S, Mader P. Development methodologies for safety critical machine learning applications in the automotive domain: A survey[C]//Proc of the IEEE/CVF Conf on Computer Vision and Pattern Recognition, Piscataway, NJ: IEEE, 2021: 129−141

[4] Reagen B, Gupta U, Pentecost L, et al. Ares: A framework for quantifying the resilience of deep neural networks[C]//Proc of the 55th ACM/ESDA/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE 2018: 1–6

[5] Neggaz M A, I. Alouani I, Lorenzo P R, et al. A reliability study on CNNs for critical embedded systems[C]//Proc of 36th Int Conf on Computer Design (ICCD). Piscataway, NJ: IEEE, 2018: 476–479

[6] Shafique M, Naseer M, Theocharides T, et al. Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead[C]//Proc of IEEE Design & Test. Piscataway, NJ: IEEE 2020: 30–37

[7] Sorin G, Bogdan T, Tiberiu C, et al. A survey of deep learning techniques for autonomous driving[J]. Journal of Field Robotics, 2020, 37(3): 362−386

[8] Mittal S. A survey on modeling and improving reliability of DNN algorithms and accelerators[J]. Journal of Systems Architecture, 2020, 104(10): 10−16

[9] Maksim J, Matteo S R, Aneesh B, et al. Challenges of reliability assessment and enhancement in autonomous systems[C]//Proc of IEEE Int Symp on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT). Piscataway, NJ: IEEE, 2019: 1−6

[10] Chen Yunji, Luo Tao, Liu shaoli et al. Dadiannao: A Machine-learning Supercomputer[C]//Proc of 47th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2014: 609−622

[11] Chen Y H, Emer J, Sze V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks[C]//Proc of 2016 ACM/IEEE the 43rd Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2016: 367−379

[12] Liu Cheng, Chu Cheng, Xu D, et al. HyCA: A hybrid computing architecture for fault-tolerant deep learning[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(10): 3400−3413

[13] Anand D, Alan W. The impact of new technology on soft error rates[C]//Proc of Int Reliability Physics Symp. Piscataway, NJ: IEEE, 2011: 5B.4.1−5B.4.7

[14] Reagen B, Whatmough P, Adolf R, et al. Minerva: Enabling low-power, highly-accurate deep neural network accelerators[C]//Proc of 2016 ACM/IEEE the 43rd Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2016: 267−278

[15] Li W, Ning X, Ge G, et al. FTT-NAS: Discovering FA ApproxABFT ult-tolerant neural architecture[C]//Proc of the 25th Asia and South Pacific Design Automation Conf (ASP-DAC). Piscataway, NJ: IEEE, 2020: 211−216

[16] He Xin, Ke Liu, Lu Wenyan, et al. AxTrain: Hardware-oriented neural network training for approximate inference[C]//Proc of the Int Symp on Low Power Electronics and Design. Piscataway, NJ: IEEE, 2018: 1−6

[17] Schorn C, Guntoro A, Ascheid G. Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators[C]//Proc of 2018 Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2018: 979−984

[18] Xu D, Chu Cheng, Wang Qianlong, et al. A hybrid computing architecture for fault-tolerant deep learning accelerators[C]//Proc of the 38th Int Conf on Computer Design (ICCD). Piscataway, NJ: IEEE, 2020: 478−485

[19] Mahdiani H R, Fakhraie S M, Lucas C. Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors[J]. IEEE Transactions on Neural Networks and Learning Systems, 2012: 1215−1228

[20] Xue Xinghua, Huang Haitong, Liu Cheng, et al, Winograd convolution: A perspective from fault tolerance[C]//Proc of the 59th ACM/IEEE Design Automation Conf (DAC). New York: ACM, 2022: 853−858

[21] Xu D, Xing K, Liu C, et al. Resilient neural network training for accelerators with computing errors[C]//Proc of the 30th Int Conf on Application-Specific Systems, Architectures and Processors (ASAP). Piscataway, NJ: IEEE, 2019: 99−102

[22] Xu D, He M, Liu C, et al. R2F: A remote retraining framework for AIoT processors with computing errors[J]. IEEE Transactions on Very Large Scale Integration Systems, 2021, 29(11): 1955−1966

[23] Ruospo A, Gavarini G, Bragaglia I, et al. Selective hardening of critical neurons in deep neural networks[C]//Proc of the 25th Int Symp on Design and Diagnostics of Electronic Circuits and Systems (DDECS). Piscataway, NJ: IEEE, 2022: 136−141

[24] Bertoa T G, Gambardella G N, Fraser N J, et al. Fault tolerant neural network accelerators with selective TMR[C]//Proc of the IEEE Design & Test. Piscataway, NJ: IEEE, 2022: 1−4

[25] Lee V T, Alaghi A, Pamula R, et al. Architecture considerations for stochastic computing accelerators[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 37(11): 2277−2289

[26] Ardakani A, Gross W J. Fault-tolerance of binarized and stochastic computing-based neural networks[C]//Proc of the 2021 IEEE Workshop on Signal Processing Systems (SiPS). Piscataway, NJ: IEEE, 2021: 52−57

[27] Qian W, Li X, Riedel M D, et al. An architecture for fault-tolerant computation with stochastic logic[J]. IEEE Transactions on Computers, 2011, 60(1): 93−105

[28] Zhang J J, Gu T, Basu K, et al. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator[C]//Proc of the 2018 IEEE 36th VLSI Test Symp (VTS). Piscataway, NJ: IEEE, 2018: 1−6

[29] Clemente J A, Mansour W, Ayoubi R, et al. Hardware implementation of a fault-tolerant hopfield neural network on FPGAs[J]. Neurocomputing, 2016, 171(1): 1606−1609

[30] Gao Z, Zhang H, Yao Y et al. Soft error tolerant convolutional neural networks on FPGAs with ensemble learning[J]. IEEE Transactions on Very Large Scale Integration Systems, 2022, 30(3): 219−302

[31] Ozen Z, Orailoglu A. Sanity-Check: Boosting the reliability of safety-critical deep neural network applications[C]//Proc of the 28th Asian Test Symposium (ATS). Piscataway, NJ: IEEE, 2019: 7−75

[32] Wang Hao, Feng Ruibin, Han Zifa, et al. ADMM-based algorithm for training fault tolerant RBF networks and selecting centers[J]. IEEE Transactions on Neural Networks and Learning Systems 2017, 29(8): 3870−3878

[33] Xue Xinghua, Liu Cheng, Liu Bo, et al. Exploring winograd convolution for cost-effective neural network fault tolerance[J], IEEE Transactions on Very Large Scale Integration Systems, 2023, 31(11): 1763−1773

[34] Zhao K, Di S, Li S, et al. FT-CNN: Algorithm-based fault tolerance for convolutional neural networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(7): 1677−1689

[35] Le H, Muhammad A H, Muhammad S. FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation[C]//Proc of the 2020 Design, Automation & Test in Europe Conf & Exhibition (DATE) . Piscataway, NJ: IEEE, 2020: 1241−1246

[36] Zhan Jinyu, Sun Ruoxu, Jang Wei, et al. Improving fault tolerance for reliable DNN using boundary-aware activation[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, 41(10): 3414−3425

[37] Chen Zitao, Li Guanpeng, Karthik P. A low-cost fault corrector for deep neural networks through range restriction[C]//Proc of the 51st Annual IEEE/IFIP Int Conf on Dependable Systems and Networks (DSN). Piscataway, NJ: IEEE 2021: 1−13

[38] Xue Xinghua , Liu Cheng, Huang Haitong, et al. ApproxABFT: Approximate algorithm-based fault tolerance for vision transformers[J]. arXiv preprint, arXiv: 2302.10469, 2023

[39] Xu D, Zhu Ziyang, Liu Cheng, et al. Reliability evaluation and analysis of fpga-based neural network acceleration system[J]. IEEE Transactions on Very Large Scale Integration Systems, 2021, 29(3): 472−484

[40] Xu D, Zhu Ziyang, Liu Cheng, et al. Persistent fault analysis of neural networks on FPGA-based acceleration system[C]//Proc of the 2020 IEEE 31st Int Conf on Application-Specific Systems, Architectures and Processors (ASAP). Piscataway, NJ: IEEE, 2020: 85−92

[41] Li M L, Ramachandran P, Sahoo S K, et al. Understanding the propagation of hard errors to software and implications for resilient system design[C]//Proc of the ACM Sigplan Notices. New York: ACM, 2008: 265−276

[42] Libano F, Wilson B, Anderson J, et al. Selective hardening for neural networks in FPGAs[J]. IEEE Transactions on Nuclear Science, 2018, 66(1): 216−222

[43] Mahmoud A, Siva K, Sastry H, et al. HarDNN: Feature map vulnerability evaluation in CNNs[J]. arXiv preprint, arXiv: 2020. 09786,2020

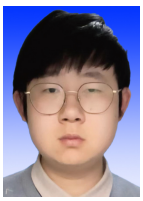
[44] Samajdar A, Zhu Y, Whatmough P, et al. Scale-Sim: Systolic CNN accelerator simulator[J]. arXiv preprint, arXiv: 1811.02883,2018

[45] Huang Haitong, Liu Cheng, Xue Xinghua, MRFI: An open source multi-resolution fault injection framework for neural network processing[J]. arXiv preprint, arXiv: 2306.11758, 2023

[46] Hanif M A, Hafiz R, Shafique M. Error resilience analysis for systematically employing approximate computing in convolutional neural networks[C]//Proc of the 2018 Design, Automation Test in Europe Conf Exhibition (DATE). Piscataway, NJ: IEEE, 2018: 913−916

[47] Xue Xinghua, Liu Cheng, Wang Ying, et al. Soft error reliability analysis of vision transformers[J], IEEE Transactions on Very Large Scale Integration Systems, 2023, 31(12): 2126−2136

[48] Pandey P, Prabal B, Koushik C, et al. GreenTPU: Improving timing error resilience of a near-threshold tensor processing unit[C]//Proc of the 56th ACM/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2019: 1−6



Zhang Qing, born in 1998. Master candidate. His main research interests include deep learning hardware acceleration and fault-tolerant chip design.

张　青, 1998 年生. 硕士研究生. 主要研究方向为深度学习硬件加速、容错芯片设计.

Liu Cheng, born in 1984. PhD, associate professor. His main research interests include domain-specific accelerator design, fault-tolerant computing, and near-data computing.

刘　成, 1984 年生. 博士, 副研究员. 主要研究方向为专用加速器设计、容错计算、近数据计算.

Liu Bo, born in 1977. PhD, professor. His main research interests include fault-tolerant computing and onboard computing.

刘　波, 1977 年生. 博士, 研究员. 主要研究方向为容错计算、星载计算.

Huang Haitong, born in 1999. Master candidate. His main research interests include deep learning algorithm and accelerator design, and fault-tolerant deep learning.

黄海同, 1999 年生. 硕士研究生. 主要研究方向为深度学习算法与加速器设计、深度学习容错.

Wang Ying, born in 1985. PhD, professor. His main research interests include approximate/error-tolerant computing, VLSI design, and energy-efficient accelerators .

王　颖, 1985 年生. 博士, 研究员. 主要研究方向为近似/容错计算、VLSI 设计、高能效芯片设计.

Li Huawei, born in 1974. PhD, professor. Her main research interests include EDA, approximate computing, fault-tolerant computing, and VLSI verification and test.

李华伟, 1974 年生. 博士, 研究员. 主要研究方向为集成电路设计自动化、近似计算、容错计算、设计验证与测试.

Li Xiaowei, born in 1964. PhD, professor. Member of IEEE. His main research interests include VLSI test, fault-tolerant computing, and hardware security.

李晓维, 1964 年生. 博士, 研究员. IEEE 会员. 主要研究方向为 VLSI 测试、容错计算、硬件安全.