中国科学: 信息科学 2019 年 第 49 卷 第 3 期: 314–333
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
面向智能应用的定制计算加速器技术专题 • 论文

低功耗神经网络计算芯片技术研究

严佳乐[1], 张颖[1], 涂锋斌[1], 杨建勋[1], 郑时轩[1], 欧阳鹏[1], 刘雷波[1], 谢源[2], 魏少军[1], 尹首一[1*]

1. 清华大学微电子学研究所, 北京 100084, 中国
2. Department of Electrical and Computer Engineering, University of California, Santa Barbara CA 93106, USA
* 通信作者. E-mail: yinsy@tsinghua.edu.cn

收稿日期: 2018–10–18; 接受日期: 2019–02–21; 网络出版日期: 2019–03–20

国家自然科学基金 (批准号: 61774094) 和国家科技重大专项 (批准号: 2018ZX01031101-002) 资助项目

摘要 当前人工智能引发了全球的热潮, 它涵盖了图像识别、视频检索、语音识别、自动驾驶等各类智能应用. 在人工智能算法中, 神经网络算法扮演着举足轻重的作用, 也成为了当前的研究热点. 但是神经网络算法本身具有灵活性高、计算复杂、数据量大的特点, 这也对计算平台提出了高性能、低功耗、高灵活性及高存储等方面的需求. 针对神经网络专用芯片, 本文提出了可重构硬件架构来满足神经网络的灵活性需求, 以可重构架构为基础的 Thinker 系列可以执行多类神经网络运算. 在该架构基础上, 本文探究了相应的数据访存优化方案来降低功耗. 在存储系统优化方面, 基于 eDRAM 的神经网络加速方案和计算存储一体化 ReRAM 方案可以满足神经网络计算在存储性能及低功耗方面的需求, 它们配合可重构硬件架构可以实现全新的神经网络加速框架. 在高效计算方面, 本文针对低比特神经网络的标准卷积计算提出基于积分和基于滤波器拆分特征重建的优化方案, 以此满足高性能需求.

关键词 人工智能, 神经网络算法, 神经网络专用芯片, 可重构架构, 低功耗

1 人工智能与神经网络计算芯片

人工智能 (artificial intelligence, AI) 是一门融合了计算机科学、统计科学、脑神经学和社会科学的前沿综合性学科. 在 2016 年 AlphaGo 击败李世石赢得人机围棋大战后, 人工智能引发了全球的热潮. 目前, 深度神经网络是人工智能最为重要的技术之一, 它模仿了人脑神经元的机制来提取数据特征 (如图像、文本、声音信息), 并做出相关预测推断. 如今, 海量数据的收集、深度学习算法的革新、硬件技术的变革、互联网生态的基础等助力人工智能爆发式发展. 而其中以核心处理器芯片为基础的计算力发挥着至关重要的作用.

引用格式: 严佳乐, 张颖, 涂锋斌, 等. 低功耗神经网络计算芯片技术研究. 中国科学: 信息科学, 2019, 49: 314–333, doi: 10.1360/N112018-00282
Yan J L, Zhang Y, Tu F B, et al. Research on low-power neural network computing accelerator (in Chinese). Sci Sin Inform, 2019, 49: 314–333, doi: 10.1360/N112018-00282

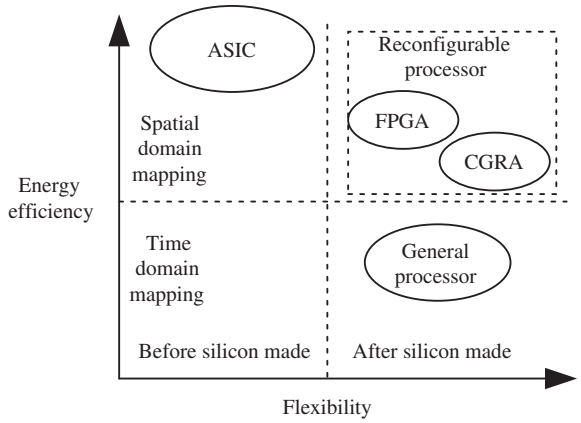

图 1 神经网络计算平台能效和灵活性的权衡

Figure 1 Balance of energy efficiency and flexibility of neural network computing platform

面向神经网络计算的主要硬件平台有: 基于 von Neumann 结构通用处理器 (CPU, GPU, DSP 等)、专用集成电路 (application-specific integrated circuit, ASIC)、现场可编程门阵列 (field-programmable gate array, FPGA)、粗粒度可重构结构 (coarse grained reconfigurable architecture, CGRA) [1] 等. 由于神经网络本身计算量大、数据量大、结构特点多样, 以 von Neumann 为结构的处理器很难在这样的算法上展现出高效的执行效率; ASIC 针对特定的计算网络结构采用了硬件电路实现的方式, 能够在很低的功耗下实现非常高的能效比 (100∼10000 GOPs/W). 在网络模型算法和应用需求固定的情况下, 它是一个不错的选择; 可重构技术在 von Neumann 结构和 ASIC 之间做了一定的折中和权衡, 它包含 FPGA 和 CGRA, 如图 1 所示. 这类可重构计算芯片具备硬件可编程能力, 可以兼顾智能应用算法中的高性能、低功耗、高灵活度的特点, 并且满足研究者针对算法特性进行定制化硬件的需求, 可以说它是未来通用 AI 芯片的技术发展路线.

2 神经网络计算芯片所面临的挑战及优化方案

2.1 神经网络计算的高性能需求

神经网络计算对硬件平台的算力有极高的要求. 在实际应用中, 以针对视频流的检测任务为例: 对于 1080P (像素点 1920×1280) 的 4 路频率为 30FPS 的图像, 选用 ResNet-152 [2] 作为主干网络提取特征信息, 需要 133 TOPS 的算力. 以 GPU 或者 FPGA 来说, NVIDIA 的 GeForce GTX 1080Ti 的算力是 11340 GFLOPS, Altera Stratix 10 的算力为 10 TFLOPS, 实际仍需要近 12 块 GPU 或 13 块 FPGA 才能完成实时任务; 而以 CPU 来说, Intel i7980XE 的算力为 79.92 GFLOPS, 相比于前两者更低. 这也意味着, 对于神经网络计算, 通用计算平台无法取得令人满意的性能, 必须设计专用的计算芯片, 才能满足神经网络计算中的高性能需求.

2.2 神经网络计算的低功耗需求

神经网络计算过程中产生的功耗一方面来源于数据访问, 另一方面源于数据运算. 大量的权重数据以及输入数据需要从片外搬运到片上, 并进行运算, 神经网络中数据访问造成的功耗甚至超过了计算所消耗的功耗 [3]. 现有的通用计算器件, 以 CPU, GPU 和 FPGA 这些现有的通用计算器件为例,

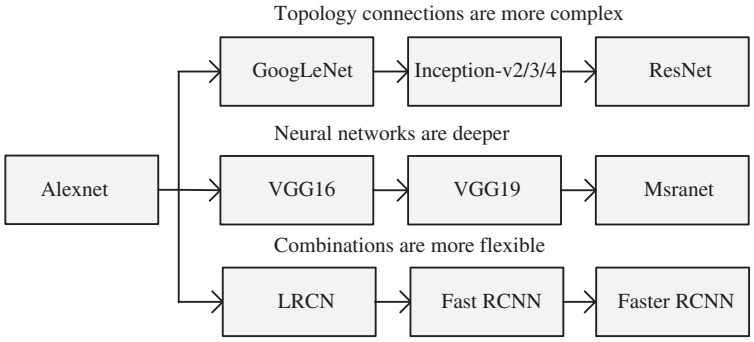

图 2　神经网络算法的演进

Figure 2　Evolution of neural network algorithms

它们在执行网络算法时能效均不理想. 以运行一个含 256 个神经元的单层 GRU-RNN 网络为例 [4], CPUi7-8700K 需要消耗 35.6 W, 能效仅为 0.53 GOPS/W. NVIDIA 的 GTX 1080Ti [4], 其峰值算力是 CPUi7-8700K 的 6.6 倍, 但消耗的功耗为 95.9 W, 能效仅为 1.29 GOPS/W. FPGA 相比于 CPU 和 GPU 有着功耗方面的优势, XC7Z100 [4] 运行该网络消耗 7.3 W, 能效为 164.11 GOPS/W. 而在实际场景如自动驾驶等领域, 需要更低的功耗和更强的算力, 神经网络专用计算芯片就有着这样的优势. 如 Mobileye 公司推出的 EyeQ4 芯片, 典型功耗为 3 W, 算力达 2.5 TOPS, 能达到自动驾驶 Level 3 的需求. 由此可见, 正因为现有通用计算器件的低能效, 令它们无法在神经网络计算及智能应用场景中符合要求, 越来越多的研究专注于低功耗、高能效的神经网络计算芯片设计.

2.3　神经网络计算的灵活性需求

神经网络算法的数据量大小不等, 网络结构多样, 同时数据本身又具备稀疏性、低比特位宽的特点. 新型的网络结构更是层出不穷, 如图 2 所示, 包含 VGG16/19 [5], GoogLeNet [6], ResNet [7] 等. 它们有的引入了含有跳转的残差级联结构, 有的加深了网络层数, 有的在已有的网络结构上增加了更多的模块, 使网络更加复杂. 同时每一层输入的特征尺寸、卷积核的大小、步长都不相同, 还含有 ReLU, Sigmoid 等激活函数单元及门控单元. 这就对神经网络计算芯片的灵活性提出了要求.

2.4　神经网络计算的存储性能需求

存储系统在深度学习加速的计算平台中扮演着非常重要的角色. 带宽、容量一直是存储的重要指标. 由于神经网络算法需要用到大量的存储资源, 包括权重数据及特征图数据. 这些数据需要几十兆甚至几百兆的存储空间. 举例来说, 如 AlexNet 拥有 6 千万个参数, 约 228 M, VGG16 拥有 1.3 亿个参数, 约 527 M, 而 Google NMT 神经网络含有 37 亿个参数, 约 13 G. 由于这些参数量过于庞大, 拥有有限空间的片上 SRAM (static random access memory) 无法全部存入它们, 绝大部分数据就被存入片外 DRAM (dynamic random access memory). 在计算时, 从片外搬运数据到片内不仅要消耗功耗, 而且 DRAM 与计算阵列之间的带宽有限, 如果芯片上的计算资源很庞大, 系统的性能往往会受到带宽的限制. Google 曾对其数据中心中运行的网络进行统计, 利用 TPU-1 [8] 针对各类神经网络做了性能分析, 以 LSTM 算法为例, Roofline 模型显示 LSTM 算法的运行强度很低, 最终只有 3 TOPS 左右, 仅为峰值性能 (92 TOPS) 的三十分之一. 原因是内存带宽限制其性能, 30 GB/s 的 DDR3 带宽成了系统性能的瓶颈. 因此, 存储系统的优化成为低功耗、高能效系统的关键所在.

2.5 神经网络计算芯片设计的优化方案

我们提出的可重构计算芯片可以针对不同类型的网络层 (如卷积层、全连接层、递归层、反卷积层等) 进行加速, 以满足灵活性需求. 同时, 基于可重构计算架构, 我们提出相应的数据访存优化方案, 它会针对不同卷积层计算特点选用不同的数据复用方式, 进而来满足低功耗需求. 在存储系统优化方面, 我们提出基于 eDRAM (embedded DRAM) 的神经网络加速方案和计算存储一体化异构 RAM 方案, 它们可以满足神经网络计算在存储性能及低功耗方面的需求, 并且配合可重构硬件架构实现全新的神经网络加速框架. 最后, 我们针对低比特神经网络运算提出基于积分和基于滤波器拆分特征重建的卷积优化方案, 以满足高性能需求.

3 神经网络计算芯片的计算架构

3.1 可重构计算芯片架构

神经网络算法不断地演变, 催生出了不同的网络结构层 (包括: 卷积、全连接、递归、残差结构等), 以及不同类型的神经网络运算 (如卷积和反卷积) 等. 我们将包含不同网络结构层的神经网络定义为混合神经网络, 如一个神经网络既包含卷积层又包含全连接层等. 大部分最新的研究工作主要集中在如何提高卷积层的性能和能效上, 但是不能高效地执行混合神经网络. 一方面, 混合神经网络相比于传统的神经网络, 含有较多的非卷积运算, 这些非卷积运算在传统的神经网络加速器中容易成为性能的瓶颈. 另一方面, 在混合神经网络中, 不同类型的网络在计算密度上有较大的差异, 而传统的神经网络计算芯片采用逐层计算的方式来计算神经网络, 但应对复杂的混合神经网络模型时, 这类方案的处理效率并不高. 在片上硬件资源有限的情况下, 如何设计通用性的神经网络计算架构来满足神经网络计算的灵活性需求是一个非常有价值的研究方向.

3.1.1 针对混合神经网络的可重构计算芯片架构

以一个混合神经网络 LRCN [9] 为例. 第一, 全连接结构和递归结构的各类操作在混合神经网络总计算量中占很大比例. 第二, 不同层具有多样化的操作密度. 卷积结构计算是计算密集型的, 全连接和递归结构计算是存储密集型的. 第三, 卷积和全连接、递归结构中的数据访问模式和数据可重用性也不一致. 根据这些特性, 我们提出了通用性的解决方案 —— 一款高效的可重构神经计算芯片 [10] (代号 “Thinker”). 它的整体架构如图 3 所示, 系统整体是由状态控制机控制, 并且通过可配置的 IO 和解码器单元来加载相应的权重数据和配置信息. 它的主要计算架构单元由 256 个可重构的处理单元 (process element, PE) 构成. 缓存控制器管理多块片上缓存, 用来为 PE 计算阵列提供数据.

为了在 PE 阵列上映射混合神经网络, 如果采用逐层计算架构, 可以依次执行所有神经网络层. 但是针对这种逐层方式, 计算密集型卷积运算结构将无法充分利用 DRAM 带宽, 而全连接和递归运算结构的大量 DRAM 访问将导致 PE 的利用率不足. 为此我们提出了新的策略, 将不同网络层对计算和存储访问的互补需求相结合, 以最大限度地利用资源. 这类适配架构的数据流具体形式是, 其中大部分 PE 分配给计算密集型的卷积结构, 大部分带宽分配给存储密集的全连接和递归结构. Thinker 的多块片上缓存 (Bank)、分布式 IO 端口, 以及不同操作的独立计算流使得 PE 阵列可以灵活地划分为 4 个子阵列, 分别处理卷积、全连接、池化 (Pooling), 以及递归结构门控操作. 卷积/全连接阵列的大小以及卷积/全连接缓存中的 Bank 数量等最佳划分参数, 会随着不同的混合神经网络拓扑结构而变化.

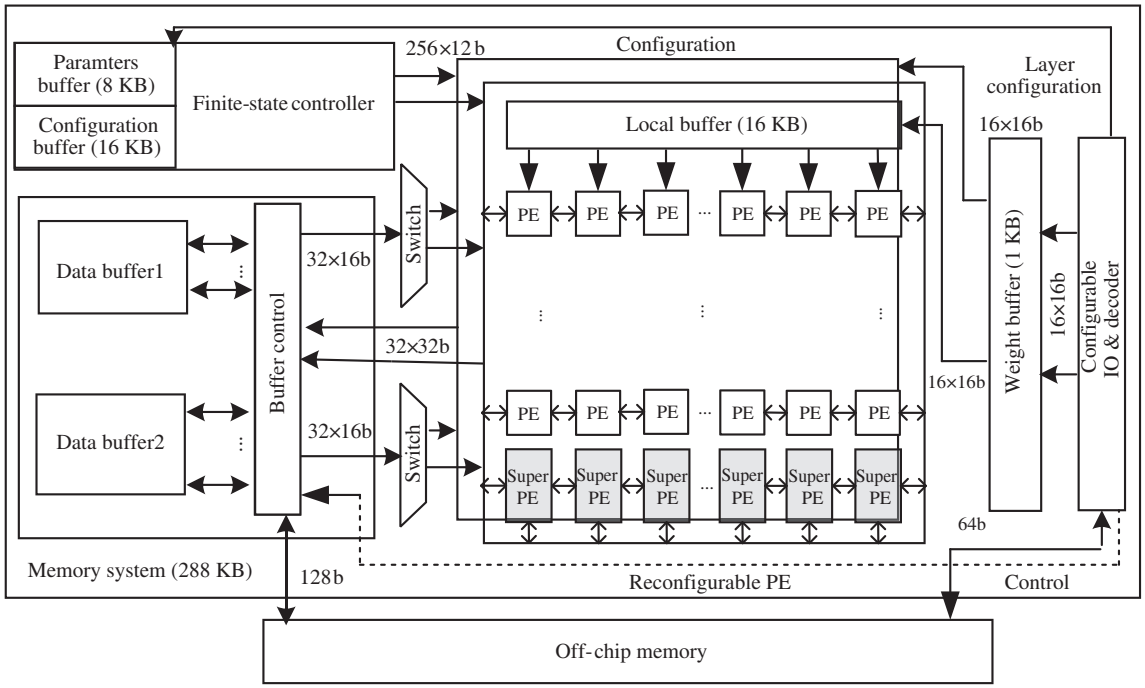

图 3 Thinker 整体架构

Figure 3 Thinker overall architecture

其中位宽自适应可配置异构 PE 阵列和多内存块的存储系统发挥了很大的作用.

(1) 位宽自适应可配置异构 PE 阵列. 为了应对不同比特位宽操作数以及卷积、全连接、池化等不同操作, 我们提出了位宽自适应的可配置异构 PE 阵列. 在每个异构 PE 中都包含位宽自适应的设计. 这样的位宽自适应设计, 可以是单比特位串行 (bit serial) 乘法器的结构, 也可以是多比特位并行 (subword parallel) 乘法器的结构. 本文以多比特位并行乘法器为例在此进行说明, 如图 4 所示. 它基于数字逻辑设计, 支持两个 8×16 比特或 1 个 16×16 比特的运算任务. 其中, s11 是控制选择器的选择信号, 其余信号包括权重、神经元, 以及时钟信号. 当执行两个 8×16 比特任务时, 两个 8 比特权重数据分别被送入两个独立乘法器中进行计算, 最后将两个 25 比特位宽结果拼接成 50 比特位宽的结果, 且 s11 置为 0. 当权重位宽大于 8 比特时, 它会将 16 比特数拆分成两个 8 比特部分, 并将两个 8×16 比特计算结果通过移位拼接的方式完成部分和的相加, 将 s11 置为 1 来进行输出. 同时 PE 分为通用 PE 和超级 PE. 超级 PE 用来完成 Pooling、Tanh、Sigmoid、标量乘法和加法, 它是由通用 PE 扩展而成的. 通用 PE 用来计算卷积、全连接, 以及递归结构的乘累加操作. 同时每一个 PE 支持遇到零值跳过操作 (zero-skipping), 用来支持网络的稀疏性.

(2) 多内存块的存储系统. 在该架构中设置了两个 SRAM 作为数据缓存来存储网络层的中间数据. 在计算过程中, 一个缓存用来提供 PE 输入数据, 另一个缓存用来存储来自 PE 阵列的输出数据. 当运算完该层以后, 对于下一层网络结构, 它们会交换两个缓存功能. 通过这种方式使得 PE 阵列、片上缓存之间的数据交换与片上缓存、片外 DRAM 之间的数据预取同时发生. 如图 5 所示, 在多层并行计算数据流中, PE 阵列被划分成若干独立的子阵列, 包括卷积阵列 (CONV array) 和全连接阵列 (FC array), 它们用于同时执行相应运算. CONV 阵列和 FC 阵列的输入特征点需要从多块 Bank 的片上缓存中并行加载. 由于卷积和全连接以及递归结构具有完全不同的数据访问模式, 为了确保并行数据访

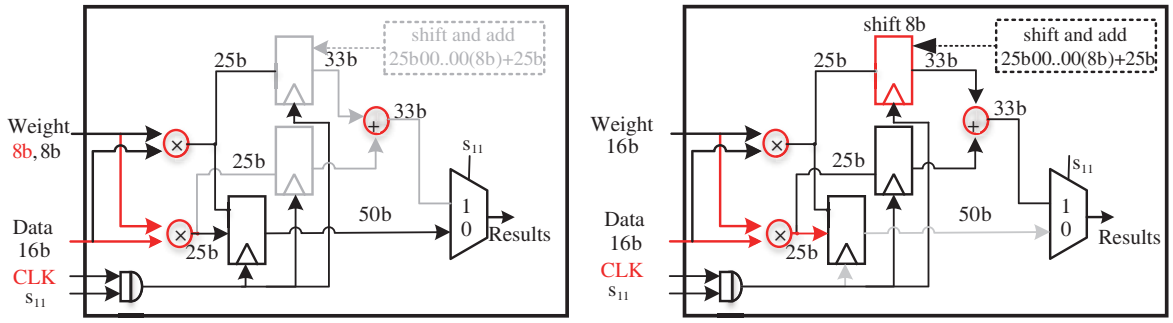

图 4　(网络版彩图) 自适应位宽计算单元

Figure 4　(Color online) Adaptive bit width computing unit

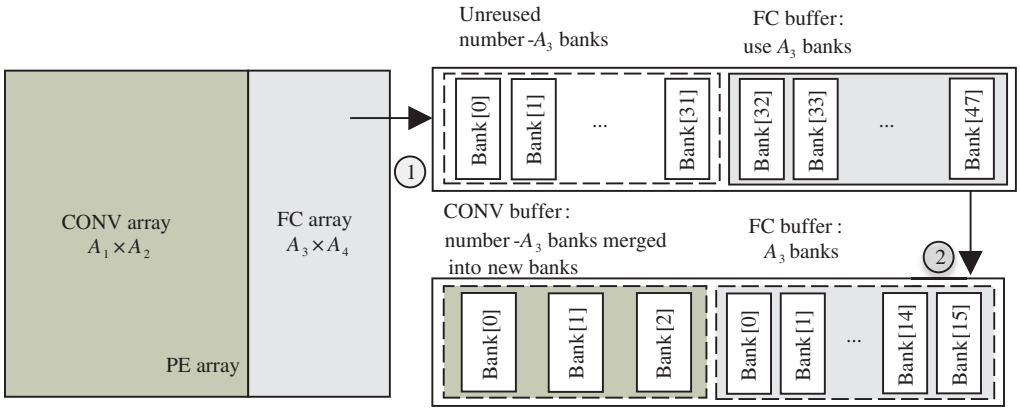

图 5　(网络版彩图) 多内存块的存储划分

Figure 5　(Color online) Multi-bank partition

问, 我们将 Bank 划分为两组: CONV 缓存 (CONV buffer) 和 FC 缓存 (FC buffer). 它们分别为 CONV 阵列和 FC 阵列提供相应的输入数据. 以图 5 中示例进行说明, 卷积阵列的大小是 $A_1 \times A_2$, 全连接阵列的大小是 $A_3 \times A_4$, 第 1 步先为 FC 阵列的长宽配置相应的 Bank 数, 以 A_3 为配置参数, 分配相应的 Bank 数. 第 2 步, 将剩余的 Bank 数根据 CONV 阵列的情况进行重新配置.

3.1.2　针对生成模型网络的硬件架构

在 3.1.1 小节中, 我们提到 Thinker 可重构计算芯片用来高效地运行混合神经网络. 随着神经网络的发展, 神经网络结构中出现了含有反卷积网络层 (Deconvlution, DeCONV) 的生成对抗网络. 同时在越来越多的计算机视觉智能应用中, 比如风格迁移、超分辨、检测技术等, 以卷积、反卷积, 以及残差模块 (residual blocks) 构成的生成网络模型扮演着重要作用. 它的结构如图 6 所示.

反卷积有着和卷积相反的运算过程. 如图 7 所示, 对于卷积运算来说, 输入特征图是 5×5 的尺寸, 输出特征图是 3×3 的尺寸, 卷积核大小为 3×3. 在卷积核遍历整个输入特征图的过程中, 每个输入值都会经历不同的乘法操作次数, 我们用不同的颜色来标记不同的操作数. 反过来看反卷积操作, 它在一开始针对输入特征图片插入了零值 (padding), 扩大了原先要做正常卷积运算的输入特征图. 原始输入特征图是 3×3 的尺寸, 输出特征图是 5×5 的尺寸, 卷积核大小为 3×3. 因为加入了零值之后, 输入特征图从原来的 3×3 变化到了 7×7, 卷积滑动窗滑动到不同位置时, 不同的输出特征图所

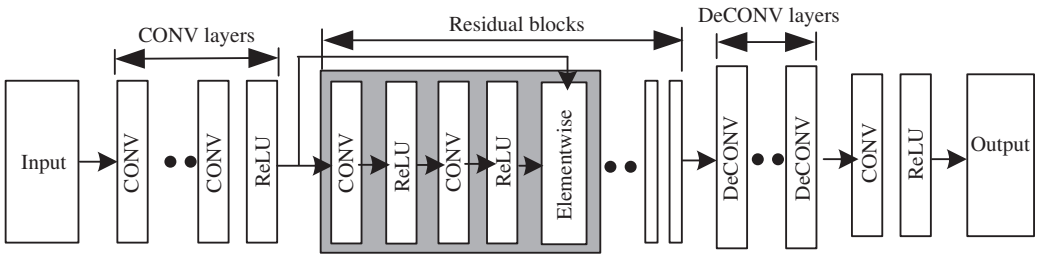

图 6 生成网络模型

Figure 6 Generative network model

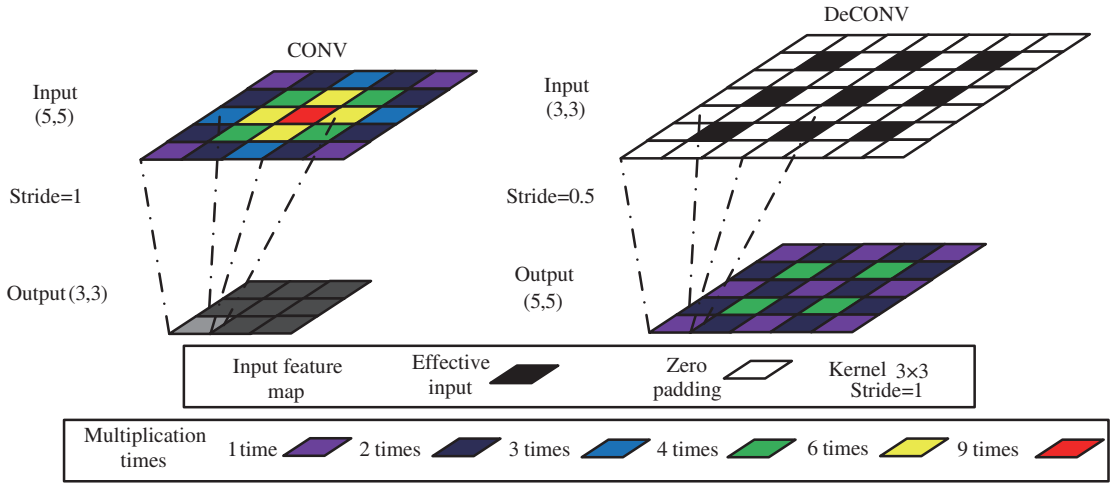

图 7 (网络版彩图) CONV 运算与 DeCONV 运算对比

Figure 7 (Color online) CONV vs. DeCONV

需的乘法操作次数不一样, 也就意味着无效值与有效值运算次数不同. 如果我们仍然用之前的数据映射方式, 将每一个输出特征点的计算任务映射到每一个计算单元 PE 上, 每个 PE 的负载会由于零值的介入而不同, 硬件的有效利用率也会大大降低.

传统的卷积神经网络加速器如 Eyeriss [3], Envision [11] 针对卷积做了很多研究如数据复用或并行计算, 但是针对反卷积模块却有着极低的 PE 利用率. 我们基于 Thinker 的架构模型, 提出了针对生成网络模型的可重构硬件架构 [12] (代号 “GNA”). GNA 的架构如图 8 所示, 它和 Thinker 架构有着类似的 2D-MAC 阵列以及输入、权重数据缓存. 针对计算单元部分, 它同样具备位宽自适应性的特点, 支持 16 比特以及 8 比特位宽的卷积运算. 所不一样的是, GNA 架构添加了冷缓存 (cold buffer) 模块. 它是用来暂存卷积乘累加的部分和数据以及 resnet 模块中的输入数据.

我们基于 GNA 的架构提出了对偶映射的技术. 针对卷积运算, 将每一个输出特征图上的点的计算映射到 PE 上去, 这样每一个 PE 的负载相对来说都比较均衡; 而针对反卷积运算, 由于插入了零值, 仍然按照上述的处理方式会带来负载不均衡的现象. 但反过来看, 针对反卷积运算, 实际上每一个输入值都会经过同样的乘法操作次数, 如果我们将每一个输入特征图上的点的计算任务映射到 PE 上去, 这样每一个 PE 的负载也会均衡.

对于残差网络模块, 传统的逐层运算方式对于点加操作 (elementwise addition) 会消耗额外的访存, 因为初始输入会在最后做一个逐点的叠加操作, 本质上逐层运算方式并没有把第 1 层的数据用尽.

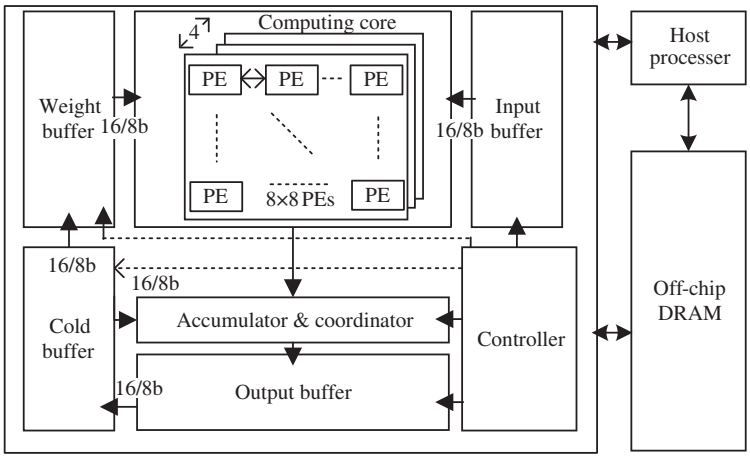

图 8 生成网络加速器架构

Figure 8 Generative network accelerator

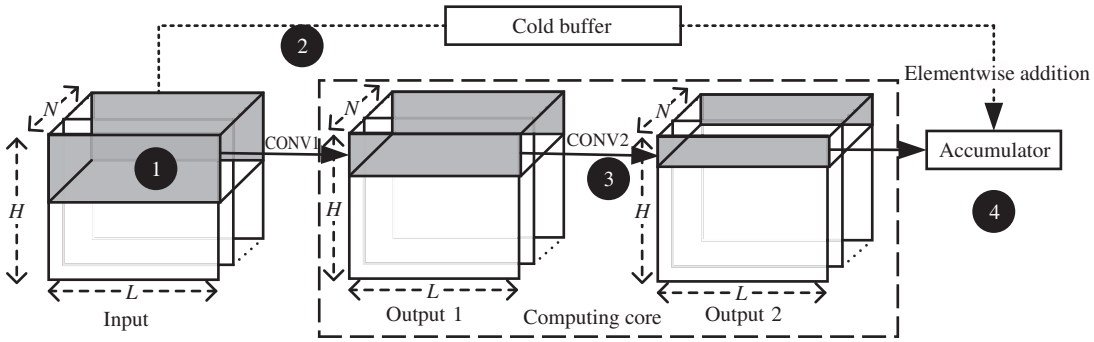

图 9 残差网络的优化调度

Figure 9 Optimization of residual network scheduling

我们提出混合卷积运算的方式, 通过流水线的操作, 如图 9 所示, 让一开始的输入特征数据持续在片上完成卷积操作如: 标记 1 和标记 3 所示, 直到最后一层; 同时片上的 Cold buffer 缓存存储下一开始的输入数据如标记 2 所示. 在完成了第 1 至 3 步后, 在第 4 步时, 在累加器上完成叠加运算. 这样所有的数据都只在片上就处理完毕, 无需额外的片外访存. 这样一方面减少了数据搬运所带来的功耗; 另一方面也减少了 PE 的等待时间, 使得运算速度更快.

3.2 神经网络计算的数据访存优化

基于上述可重构神经网络计算芯片, 我们提炼出一套基本架构模型如图 10 所示. 它具备上述提到的可重构架构模型的特点. 它由片上部分和片外部分组成, 片外的主处理器用来发送配置信号, 片外访存用来存储大规模的权重、特征图数据. 片上的存储部分由权重、输入、输出 3 部分构成, 计算核心负责从片上缓冲中读取权重和特征数据来运算, 其中包含有卷积模块和池化模块. 我们在本小节讨论在此架构基础上的数据访存优化. 因为卷积层的计算几乎占据了卷积神经网络的计算总量, 其他比如池化层、全连接层占据的计算量较少, 它们的计算模式和卷积类似, 都可以用同样的一组参数描述出来. 我们认为对卷积层计算的研究具有非常重要的意义. 因此本小节, 以它为研究对象, 重点分析在

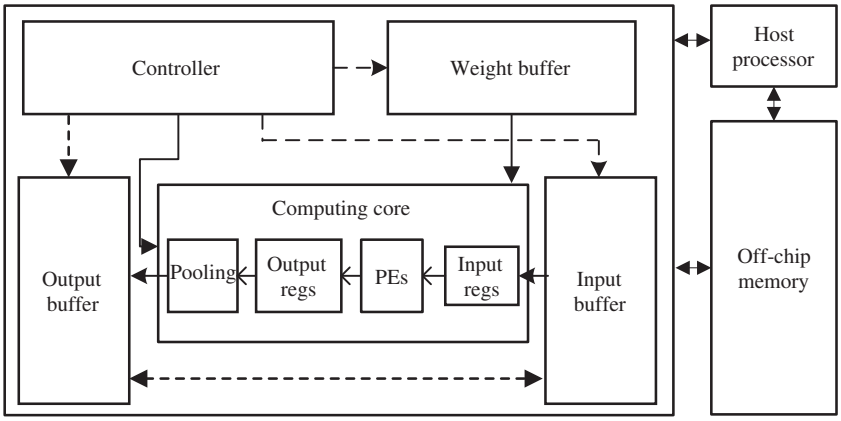

图 10　可重构神经网络计算架构

Figure 10　The architecture of reconfigurable neural network acceleration

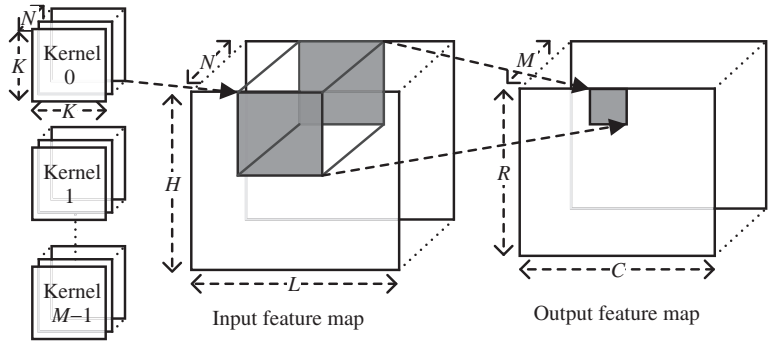

图 11　卷积层计算示意图

Figure 11　Graph of a CONV layer

神经网络加速器执行过程中, 对片上缓存和片外 RAM 的访问问题, 并提出相应的数据复用技术 [13].

许多工作如 Diannao [14] 以及 Eyeriss [3] 采用固定的数据复用方式来处理不同大小的卷积运算. Diannao 最小化了输出数据的访问量, 而 Eyeriss 最小化了输入数据的访问量, 导致了不同的性能和能效. 由于卷积神经网络本身含有不同的权重参数, 如果硬件平台能针对不同特征的网络层采用不同的数据复用方式, 将大大改善性能. 有些工作如 FPGA'15 [15] 提出了一个通用的分析方法来处理卷积在 FPGA 硬件平台上的执行方式, 但还是单一的输出数据复用的遍历顺序, 并没有涉及到权重和输入数据复用的探究.

3.2.1　数据复用技术

卷积的运算如图 11 所示. 输入是一组 N 通道的三维特征图, 输出是一组 M 通道的三维特征图, 同时含有 M 个 N 通道的卷积核. 同样卷积层的计算可以表示成伪代码如图 12 所示, 其中 I, O, W 代表的是输入数据、输出数据, 以及卷积核权重, R 和 C 代表输出图片的尺寸.

以输入数据复用技术为例, 图 13 展示了输入数据复用的情况. 我们可以将这个过程分为 3 步: 第 1 步计算核心把输入特征图读入局部的输入寄存器. 第 2 步计算核心充分复用这些输入数据, 更新输出缓存中的所有相关的输出部分和. 第 3 步更新后的输出结果重新写回到输出缓存中. 当新一批次的

```
for(in = 0; in < N; in ++)                    // Loop R
for(om = 0; om < M; om ++)                    // Loop C
for(or = 0; or < R; or ++)                    // Loop M
for(oc = 0; oc < C; oc ++)                    // Loop N
  O[om][or][oc] +=                            // Convolution
    Σ_{i=0}^{K-1} Σ_{j=0}^{K-1} W[om][in][i][j] × I[in][or × S + i][oc × S + j];
```

图 12 卷积层计算的循环表示

Figure 12 Pseudo code of a CONV layer

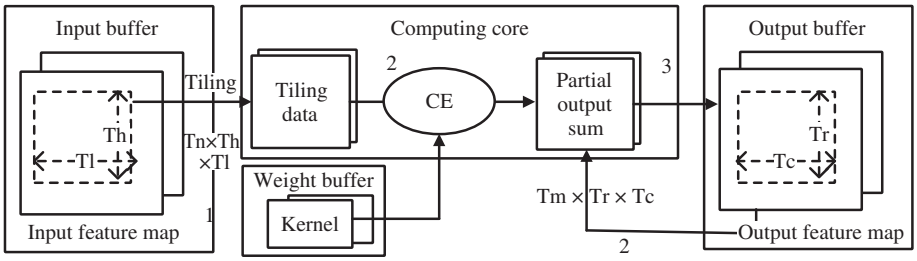

图 13 输入数据复用模式

Figure 13 Input reuse model

```
for(or = 0; or < R; or += Tr)                          // Loop R
for(oc = 0; oc < C; oc += Tc)                          // Loop C
for(in = 0; in < N; in += Tn)                          // Loop N
for(om = 0; om < M; om += Tm)                          // Loop M
//Computing core
  for(tm = om; tm < min(om + Tm); tm + +)              // Loop Tm
    for(tn = in; tn < min(in + Tn); tn + +)            // Loop Tn
      for(tr = or; tr < min(or + Tr, R); tr + +)       // Loop Tr
        for(tc = oc; tc < min(oc + Tc, C); tc + +)     // Loop Tc
          O[tm][tr][tc] +=                             // Convolution
            Σ_{i=0}^{K-1} Σ_{j=0}^{K-1} W[tm][tn][i][j] × I[tn][tr × S + i][tc × S + j];
```

图 14 输入数据复用伪代码

Figure 14 Pseudo code of input reuse

数据被读入计算核心时, 再重复上述 3 个步骤. 而卷积的计算模式也如图 14 所示. 我们采用统一的公式来表示片上缓存和片外 DRAM 的访问次数, $\mathrm{MA} = \mathrm{Tl} \times \alpha_i + \mathrm{TO} \times \alpha_o + \mathrm{TW} \times \alpha_w + \mathrm{TPO}$. 其中, Tl, TO, TW 分别表示当前层的输入、输出、权重总数; α_i, α_o, α_w 是数据复用参数, 对应输入、输出、权重在计算过程中的重复访问次数. TPO 是输出数据经过池化操作后的个数. 在这种情况下, 计算核心对于片上缓存来说, 数据复用参数为 $\alpha_i = 1$, $\alpha_o = 2\lceil \frac{N}{\mathrm{Tn}} \rceil - 2$, $\alpha_w = \lceil \frac{H}{\mathrm{Th}} \rceil \lceil \frac{L}{\mathrm{Tl}} \rceil$. 如果数据缓存需求超过了片上缓存容量, 那么就会产生额外的片外 DRAM 访问. 在这种情况下, 神经网络计算芯片对片外 DRAM 的数据据复用参数计算如下:

$$\alpha_i = 1, \quad \alpha_o = \begin{cases} 0, & M \times \mathrm{Tr} \times \mathrm{Tc} \leqslant B_o, \\ 2\left\lceil \dfrac{N}{\mathrm{Tn}} \right\rceil - 2, & M \times \mathrm{Tr} \times \mathrm{Tc} > B_o, \end{cases} \quad \alpha_w = \begin{cases} 1, & \mathrm{TW} \leqslant B_w, \\ \left\lceil \dfrac{H}{\mathrm{Th}} \right\rceil \left\lceil \dfrac{L}{\mathrm{Tl}} \right\rceil, & \mathrm{TW} > B_w, \end{cases}$$

这其中, B_w 和 B_o 分别表示权重缓存和输出缓存的容量.

输出及权重数据的复用方式与输入数据复用方式类似. 图 15 和 16 分别展示了输出数据复用和

```
for(or = 0; or < R; or += Tr)                          // Loop R
for(oc = 0; oc < C; oc += Tc)                          // Loop C
for(om = 0; om < M; om += Tm)                          // Loop M
for(in = 0; in < N; in += Tn)                          // Loop N
//Computing core
  for(tm = om; tm < min(om + Tm); tm + +)              // Loop Tm
    for(tn = in; tn < min(in + Tn); tn + +)            // Loop Tn
      for(tr = or; tr < min(or + Tr, R); tr + +)       // Loop Tr
        for(tc = oc; tc < min(oc + Tc, C); tc + +)     // Loop Tc
          O[tm][tr][tc] +=                             // Convolution
          Σ_{i=0}^{K-1} Σ_{j=0}^{K-1} W[tm][tn][i][j] × I[tn][tr × S + i][tc × S + j];
```

图 15　输出数据复用伪代码

Figure 15　Pseudo code of output reuse

```
for(om = 0; om < M; om += Tm)                          // Loop M
for(in = 0; in < N; in += Tn)                          // Loop N
for(or = 0; or < R; or += Tr)                          // Loop R
for(oc = 0; oc < C; oc += Tc)                          // Loop C
//Computing core
  for(tm = om; tm < min(om + Tm); tm + +)              // Loop Tm
    for(tn = in; tn < min(in + Tn); tn + +)            // Loop Tn
      for(tr = or; tr < min(or + Tr, R); tr + +)       // Loop Tr
        for(tc = oc; tc < min(oc + Tc, C); tc + +)     // Loop Tc
          O[tm][tr][tc] +=                             // Convolution
          Σ_{i=0}^{K-1} Σ_{j=0}^{K-1} W[tm][tn][i][j] × I[tn][tr × S + i][tc × S + j];
```

图 16　权重数据复用伪代码

Figure 16　Pseudo code of weight reuse

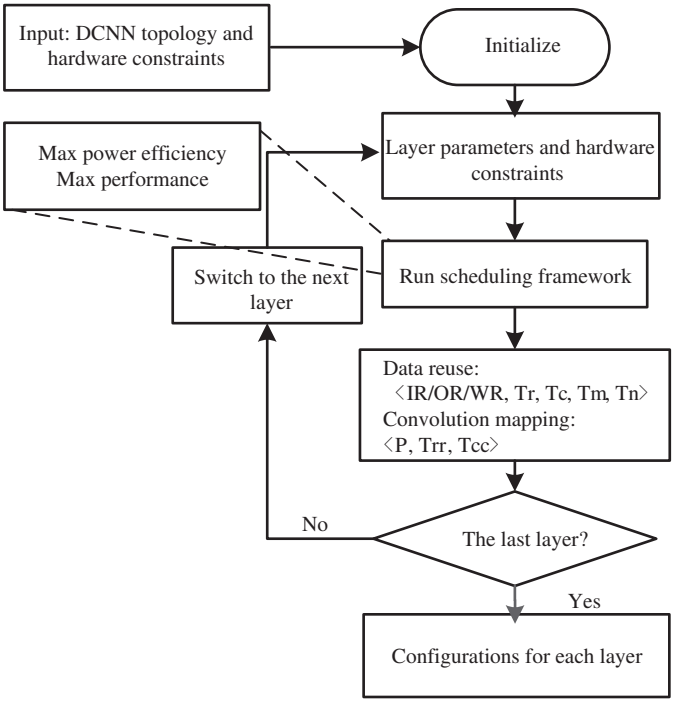

图 17　混合数据复用模式的工作流程

Figure 17　Workflow of mixed data reuse patterns

权重数据复用的方式.

3.2.2　混合数据复用模式的工作流程

在可重构的深度神经网络计算架构下, 灵活的数据传输通路支持 3 种不同的数据复用模式. 在这种工作流程下, 我们可以根据神经网络每层参数特性的不同, 选用不同的数据复用模式. 具体可以分为两个阶段: 第 1 阶段是编译阶段如图 17 所示, 它会给神经网络的每一层分配一个最优的数据复用模式及分块参数, 并编译成硬件架构的配置信息; 第 2 阶段是执行阶段. 在第 1 阶段的优化任务中, 我们以能效 (energy efficiency) 和性能 (performance) 为两个最大化的优化目标, 其中能量的计算公式

$\text{Energy} = \text{Energy}_{\text{MA}} + \text{Energy}_{\text{compute}} = \text{MA}_{\text{DRAM}} \times E_{\text{DRAM}} + \text{MA}_{\text{buffer}} \times E_{\text{buffer}} + \text{Operations} \times E_{\text{Operation}}$. 这其中, MA_{DRAM} 代表 DRAM 的访问次数, $\text{MA}_{\text{buffer}}$ 代表 buffer 访问次数, E_{DRAM} 和 E_{buffer} 分别代表单位 DRAM 和 buffer 访问能耗, Operations 和 $E_{\text{Operations}}$ 分别代表计算操作数和单位操作能耗. 对于给定的 CNN 模型, 计算操作数是固定的, 因此影响能耗的关键因素是 DRAM 访问次数和 buffer 访问次数, 我们针对不同的网络配置参数挑选出最适合它的数据复用模式来降低访存的访问次数. 性能的计算公式 $\text{Performance} = 2\text{MAC} \times \text{PEUtilization} \times \text{Frequency}$. 其中 MAC 代表硬件乘加单元数, PEUtilization 代表计算单元利用率, Frequency 代表计算核心频率. 对于给定的硬件架构, 影响性能的关键因素是计算单元在整体工作过程中的利用率. 设置好数据复用过程中的循环参数也会影响到利用率. DNA 架构会根据配置信息重构片上硬件资源, 并以最高的能效执行神经网络.

3.3 实验结果

上述代号为 Thinker 的可重构计算芯片, 最终采用了 65 nm LP CMOS 工艺流片. 在 200 MHz 的标称频率和 1.2 V 电压下, Thinker 处理器的峰值吞吐率达到 409.6 GOPS, 功耗为 386 mW, 对应的能效为 1.06 TOPS/W. 当电压缩小至 0.67 V 时, 吞吐率和功耗在 10 MHz 时降至 20.4 GOPS 和 4 mW, 对应的能效为 5.09 TOPS/W. 我们将 Thinker 平台和其他加速器的性能进行分析对比. 在 AlexNet 的测试网络下, 在峰值性能方面, Thinker 相比于 Eyeriss [3], ENVISION [11] 有 5∼6 倍的优势; 在能量效率指标方面, Thinker 芯片是 Eyeriss 的 8 倍之多. 我们将 GNA 芯片和其他硬件平台进行对比, 以风格迁移网络 (style transfer net) [16] 和超分辨网络 (super resolution net) [17] 为测试网络, 该架构在运行时的峰值能效比传统的神经网络加速器如 Eyeriss 高 14.8 倍. 我们将数据访存方案运用在可重构神经网络计算芯片上, 以 AlexNet 为测试网络, 在能效指标方面, 搭载此方案的计算芯片为 152.9 GOPS/W, 它比 VirtexVX485t 的硬件 [15] 高了 46 倍, 比 Eyeriss [3] 高了 1.9 倍.

4 神经网络计算芯片存储系统优化

随着神经网络的进一步发展, 为了获得更高的精度, 网络的规模及参数量也不断的增大, 甚至导致数据存储的容量达到了几兆或几十兆. 以 VGG 网络为例, 在输入图片尺寸为 $224 \times 224 \times 3$ 时, 容量已经达到了 6.27 MB (精度为 16 位宽数据). 并且伴随着输入图片的分辨率以及批量数据的增大, 这个值会不断的增大. 如 Eyeriss [3] 等以静态存储器 SRAM 为片上存储器的传统神经网络加速器, 因为其有限的存储空间, 不可避免需要产生额外的 DRAM 访问来进行神经网络的计算, 这也就导致了系统额外的能耗开销. 利用这类存储介质的硬件架构需要不断地根据网络参数来配置相应的卷积运算循环顺序以及数据复用模式. 如果能从对存储系统进行优化, 就能大大改善能耗开销. 因此, 在 AI 计算芯片设计过程中, 如何优化存储是一个急需解决的问题.

4.1 片上 DRAM 优化

嵌入式 DRAM 存储器 (eDRAM) 作为片上存储器, 与传统的 SRAM 对比, 它具有更高的存储密度, 并且大幅减少片外访存. 嵌入式 DRAM, 其结构如图 18 所示, 每个 eDRAM 存储单元由一个晶体管 (access transistor) 和一个电容 (storage capacitor) 构成. 数据的逻辑值会以电荷的形式存储在电容中. 但是, eDRAM 中存储的电容电荷会随着晶体管的漏电流而随着时间逐渐消失, 因此需要通过周期性的刷新操作来维持数据存储的正确性, 而刷新能耗是 eDRAM 总能耗的主要来源. 但是我们发现,

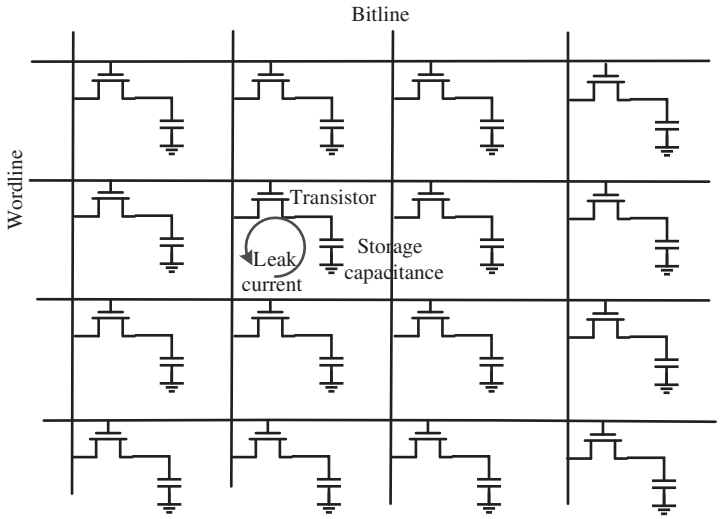

图 18　eDRAM 结构

Figure 18　eDRAM structure

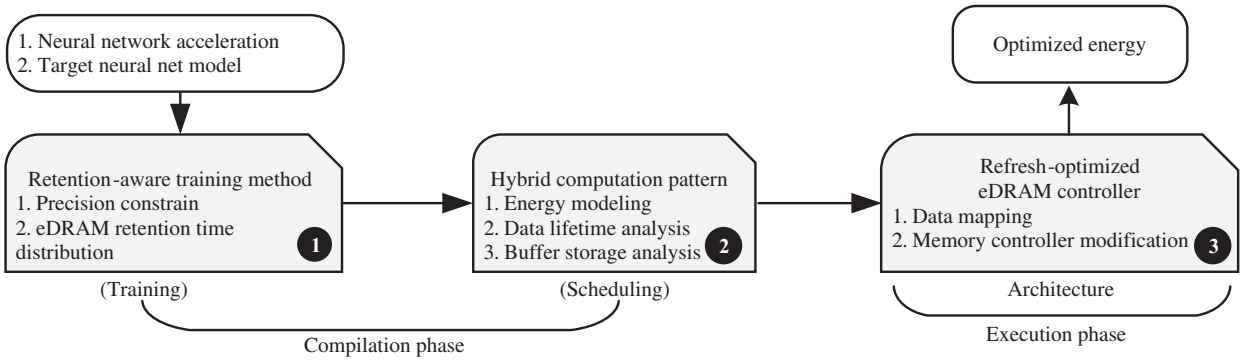

图 19　数据生存时间感知的神经网络加速框架

Figure 19　Retention-aware neural acceleration framework

如果数据在 eDRAM 中的生存时间 (data lifetime) 小于 eDRAM 的数据维持时间 (retention time), 那么系统将不再需要对于此数据的刷新操作. 由此可以得到两个优化方向: 减少数据生存时间和增大数据维持时间.

结合 eDRAM 存储器的特性, 我们提出了一种新型神经网络加速框架 [18], 如图 19 所示. 它以神经网络计算芯片和目标神经网络的具体参数为输入, 通过 3 个层次 (数据生存时间感知的训练方法, 神经网络分层的混合计算模式和刷新优化的 eDRAM 控制器) 的技术, 分别从训练、调度和架构 3 方面来降低 eDRAM 刷新能耗, 进而大幅度优化系统能耗.

在训练层次, 我们使用了数据生存时间感知的训练方法, 重新训练网络, 提高其对数据维持错误 (retention failure) 的容错能力, 如图 20 所示. 本方法在训练的正向过程中对输入和权重数据加入一个掩膜 (layer mask) 以引入误差. 这个掩膜会以一定的错误率对每个比特注入误差. 经过反复的重训练, 如果最终的精度损失可以接受, 那么就认为网络可以承受当前的错误率, 这种错误会在训练的正向过

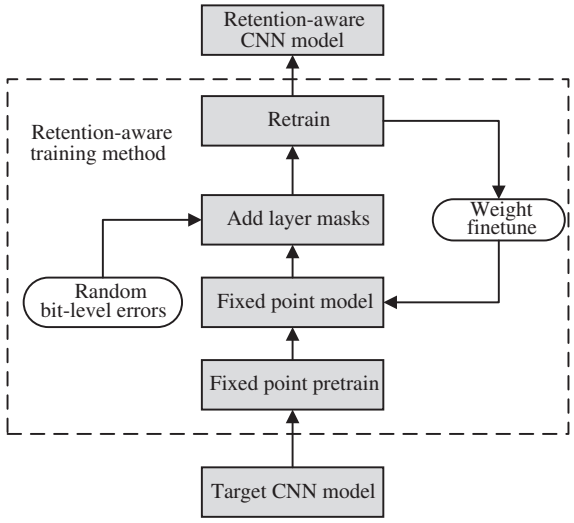

图 20 训练层次优化：数据生存时间感知的训练方法

Figure 20 Training hierarchy optimization: retention-aware training method

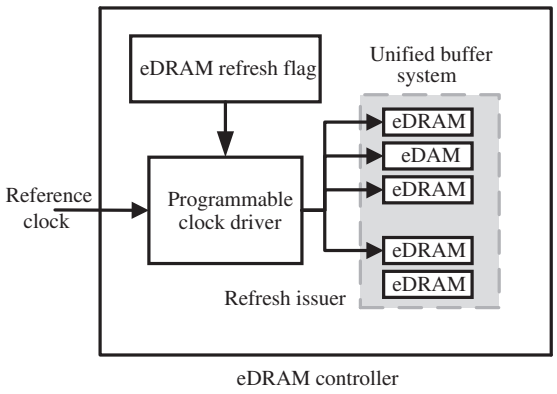

图 21 架构层次优化：刷新优化的 eDRAM 控制器

Figure 21 Architecture level optimization: refresh the optimized eDRAM controller

程中以一定的错误率注入到输入和权重数据中. 在给定的精度约束下, 本文的训练方法将得到最大可容忍的错误率. 根据 eDRAM 的维持时间特性曲线, 我们得到对应的最大可容忍的维持时间.

在调度层次, 本文提出了混合计算模式, 它可以根据芯片参数及 DNN 网络参数, 对网络每一层的计算分别调度, 并分配一个最优的计算模式. 计算模式的探索过程被抽象为一个目标为系统能耗最小化的优化问题. 其中, 系统能耗的建模考虑了计算能耗、片上访存能耗、刷新能耗和片外访问能耗. 调度结果会被编译成分层的配置信息 (包括可容忍的数据维持时间、每层的计算模式及刷新标志), 以用于执行阶段的硬件配置.

在架构层次, 本文设计了一个 eDRAM 控制器如图 21 所示, 它的内部含有一个可编程的时钟分频器, 各个 eDRAM 分区独立地进行刷新触发器和刷新标志位的操作. 这样的目的在于实现对每个存储分区独立的刷新控制, 并且进一步降低刷新能耗. 前两个阶段为编译阶段, 任务是生成配置信息; 最后一个阶段是执行阶段, 它会根据配置信息来重构芯片的执行方式, 具体决定每个分区分别存储什么数据类型、是否需要刷新, 以及刷新周期. 这个技术从存储优化和软硬件协同设计的角度大幅提升了芯片能量效率, 给面向智能应用的定制计算加速器的架构演进提供了新思路.

4.2 计算存储一体化优化技术

4.1 小节介绍了采用 eDRAM 作为片上存储器的优化方案, 我们将在本小节介绍计算存储一体化技术的解决方案, 并以一项可计算存储器设计: PRIME [19] 为实例进行陈述. 它是将计算单元转移到存储单元中. 这样一来, 数据处理将在数据存储的本地进行, 完全消除了数据搬移所带来的开销; 同时, 由于储存单元内部的带宽远大于通过储存总线在处理器端看到的带宽, 所以不仅在能效上, 在性能上也会有很大提升.

在介绍 PRIME 之前, 先介绍一下 ReRAM 的基础知识. ReRAM 是一种新型的存储介质, 被认为在未来代替 DRAM 成为主流内存介质. 不同于 DRAM, ReRAM 用电阻值来存储 0/1 (即高阻值代表 0、低阻值代表 1), 而非电荷. 对于写入数值, ReRAM 则通过加入正反电压来至 1 或至 0. 与 DRAM

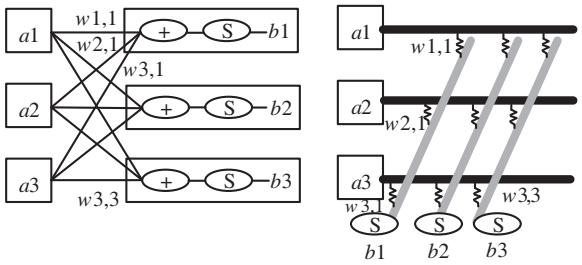

图 22 ReRAM 进行计算的示意 [19]

Figure 22 ReRAM for computing [19]

不同的是, ReRAM 还可以实现多位存储, 即在一个存储介质单元中, 存入多位比特. 这是通过不同的电阻值来表示的. 在存储阵列的结构设计上, ReRAM 有它的特点: 它采用了无晶体管的 crossbar 模式, 即是由字线 (wordline, WL) 和位线 (bitline, BL) 交叉连接, 其连接点上是 ReRAM 存储单元. 这样的存储阵列结构提供了更小的面积占用: DRAM 的单位面积为 $6F^2$, 而 ReRAM 则可以达到 $4F^2$. 更加吸引人的是, ReRAM 不仅仅可以用来作为存储单元, 还可以用来进行计算. 如图 22 所示, 左图显示了一个带有两个神经元的简单神经网络, 其中 a 代表输入, w 代表权重, b 代表输出. 将其映射到 ReRAM 存储单元上, 那么我们将输入 a 作为电压加载在 wordline 上, 然后将权重 w 作为数据, 存储在 ReRAM 单元中, 然后在 bitline 上得到的电流, 既为我们需要计算的神经网络输出. 这一点利用了 ReRAM 的电阻特性, 在加载在 wordline 上的电压通过电阻时, 相当于与其阻抗相乘然后再在 bitline 上进行累加, 得到我们需要的计算结果.

PRIME 的计算存储一体化设计遵循 ReRAM, 它既可以用作存储单元, 又可以进行神经网络计算的特点, 从而构成了一种可计算的存储器. 如图 23 所示, 它不同于传统的 von Neumann 架构, 也不同于 3D 集成存储器的可计算内存结构, PRIME 在存储芯片的内部进行计算. 这样的方式更强地融合了计算和存储, 并且使得可提供的数据带宽更高从而有更好的性能, 同时极大程度地减少了数据搬移的能耗. 不同于早期可计算存储设计在存储介质中添加昂贵的逻辑单元, PRIME 利用存储介质本身作为存储, 又作为计算单元的性质, 极大程度地减小了面积代价. 更重要的是, PRIME 提出了一种灵活的运作方式: 我们针对大量 ReRAM 存储单元的一小部分进行修改, 使得该区域支持计算和存储两种模式. 在需要进行神经网络加速的时候, 将该区域切换到加速器模式, 这样可以实现内存计算的高能效加速. 在其他时候, 该区域会作为存储区域, 从而使得系统得以利用更大的内存空间.

5 低比特神经网络计算

近年来, 基于低比特权重和特征图的神经网络模型开始成为热点, 研究者发现在引入批正则化以及适度扩大网络规模的基础上, 可以通过合理的低比特化手段, 可以将网络模型的数据位宽降低, 并且在部分数据集上保持可观的识别率. 我们对神经网络低位宽量化方法、计算架构, 以及电路实现进行了系统研究, 提出了支持低位宽网络的高能效计算方案. 该方案针对二值/三值网络的标准卷积形式进行优化, 大幅降低了算法复杂度. 同时在此基础上, 设计了神经网络通用计算芯片 Thinker-II [20].

5.1 低比特神经网络

由于神经网络层数的增加以及结构的复杂化, 这阻碍了神经网络计算性能的提高. 相关研究者们

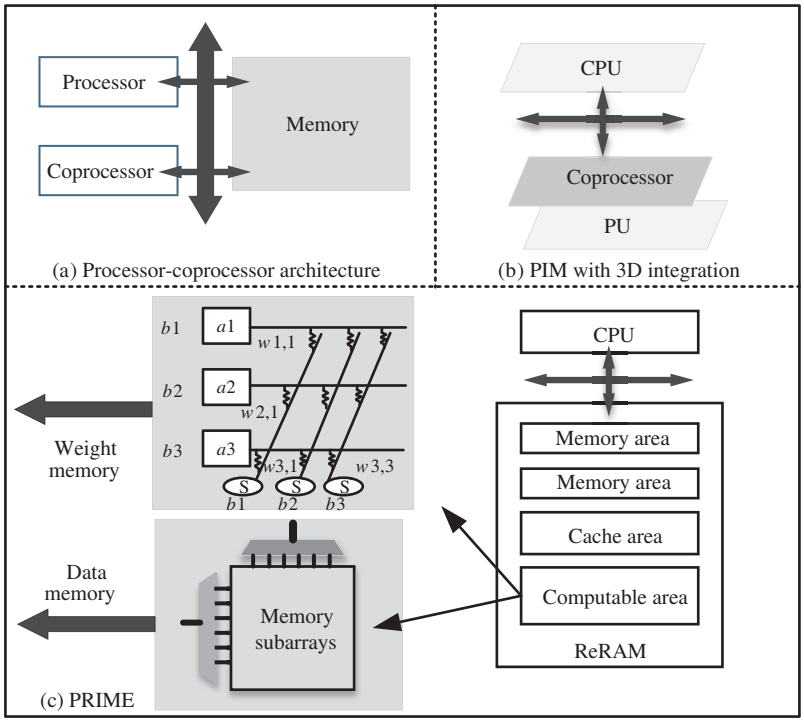

图 23 PRIME 计算存储一体化设计 [19]

Figure 23 PRIME design [19]

开始思考提高神经网络的计算性能. 这些方法中, 神经网络位宽优化是最为重要的方法之一. 有许多相关算法提出减小神经网络的数据位宽. 网络位宽定点化是指将神经网络在算法层面的浮点运算转换为针对硬件层面的定点运算, 并降低网络位宽需求, 并保持其识别精度不发生较大的变化, 其难点就在于如何达到低位宽网络并使识别精度不发生太大的改变. 而将低比特发挥到极致也就催生了二值网络以及三值网络, 以二值神经网络 (binarized neural network, BNN) 为例, 它的关键特征是将权重参数和特征图的数值限制在 $+1$ 和 -1 两个值上. 这些低比特网络在语音识别应用中发挥着至关重要的作用.

5.2 面向二值或三值权重神经网络的卷积优化技术

在面向二值或三值的权重神经网络中, 我们提出了两种卷积优化方法 [20], 它能够让硬件架构在执行过程中大幅降低了二值/三值权重神经网络 (BTNN) 的算法复杂度, 并去除了冗余计算.

5.2.1 基于积分的卷积计算

第 1 种卷积优化方法是基于积分的卷积计算. 如图 24 所示. 它通过用两种低复杂度的计算等效替代标准的卷积计算, 降低了计算复杂度. 对于一个标准的二值或三值卷积运算, 可以将权重数据进行处理分解, 原先的权重值 $-1, 1, 0$ 通过减 1 并除以 2 的操作 $W = 1 - 2 \times W_{\mathrm{FIBC}}$, 变成了 $1, 0, \frac{1}{2}$, 以增加稀疏性和移位运算的代价而减少了复杂的操作数. 这样一来, 原先的标准卷积就被分解成了一个整体卷积减去两个部分卷积. 对于第 1 部分特征求和而言, 通过图 24 部分知道, 它的计算被分成了两个阶段. 第 1 个阶段先对一个输入特征图 (input feature map) 进行积分求和, 得到第 2 个阶段的特征图, 计算规则是以输入中的部分区域的总和作为映射值, IM 中的部分区域的右下角的数的坐标值作为映

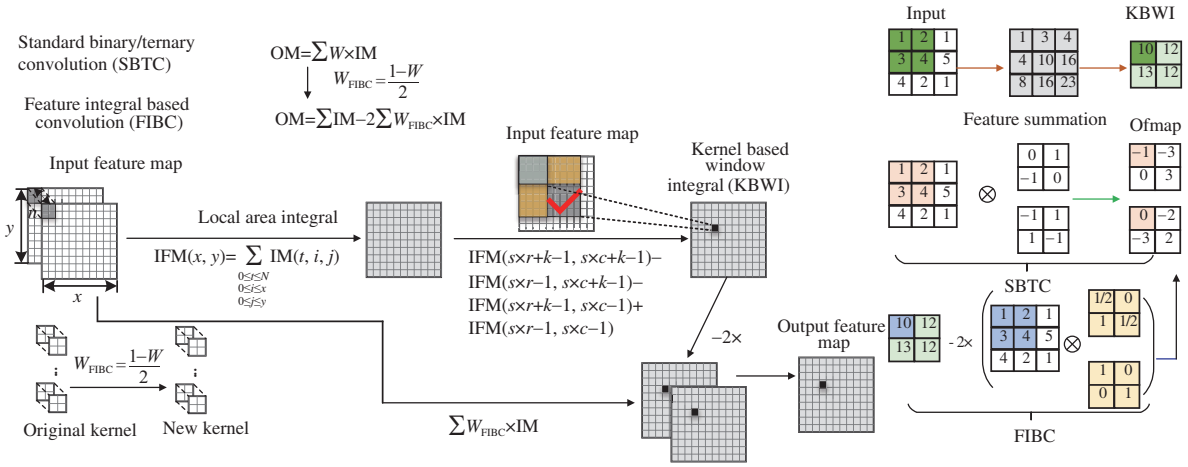

图 24 (网络版彩图) 基于积分的卷积计算

Figure 24 (Color online) The process of feature-integral-based convolution

射到 IFM 上的坐标位置. 于是 IFM 中每一个值所代表的就是 IM 中部分区域的总和数值. 第 2 个阶段对 IFM 进行数值选择, 因为 IFM 中的每一个值代表了 IM 中不同区域的数值和, 于是只要挑选出合适的数值就能通过加减法得到 IM 中的卷积区域的部分和数值. 一个标准的卷积运算就可以拆分为特征整数固定的卷积运算了. 而图 24 则真正说明了整个计算过程的细节, S 代表步长, k 代表卷积核大小, r 和 c 表示输出点的横、纵坐标值. 通过在 IFM 中取值做简单的加减法就能完成特征求和的过程, 这个值可以进行共享复用, 用来减去拆分之后的 $W_{\mathrm{FIBC}} \times \mathrm{IM}$ 部分的和, 而这一部分的计算就是正常的卷积化计算, 只是卷积权重的值被替换为了由 1, 0, $\frac{1}{2}$ 组成的 FIBC 卷积核.

5.2.2 基于滤波器拆分特征重建的卷积计算

第 2 种是基于滤波器拆分特征重建的卷积计算, 如图 25 所示, 通过将相似度高的卷积核, 进行相加减和除 2 运算, 可以两两拆分成更稀疏的不相关的卷积. 然后进行拆分后的卷积计算, 最后将卷积结果重建到标准卷积值, 这大大减少了低比特网络卷积的冗余计算.

我们将这两种方式应用到二值或三值化处理完后的网络中, 如 AlexNet, VGG16, GoogLeNet, 并进行运算量的分析比较. 实验过程采用 1 比特位宽权重 16 比特位宽特征值的模式. 我们统计了该模式下的精度损失情况, 并分别使用标准二值/三值卷积 (SBTC, standard binary/ternary convolution)、基于积分的卷积 (FIBC, feature integral based convolution) 和基于滤波器拆分重建卷积 (KTFR, kernel transformation feature reconstruction convolution) 3 种方式考察计算量的减少情况. 结果显示, 在二值或三值化处理之后, 精度有所下降, AlexNet 下降了 0.4%, VGG16 下降了 1.8%, GoogLeNet 下降了 6.5%. 而在计算量减少方面, 基于积分的卷积方式对于 1 比特模式下的 AlexNet, VGG16 和 GoogLeNet 3 个网络分别减少了 63.4%, 62.4% 和 70.7%. 基于特征整数固定卷积方式则分别减少了 49.5%, 49.9% 和 48.4%.

5.2.3 层次化均衡调度机制

我们也用了一种层次化均衡调度机制, 通过软硬件协同的两级任务调度对计算单元间算子和计算单元内计算的数据进行有效的调度. 因为每个计算单元 (PE) 负责一个卷积的运算, 当卷积的稀疏性

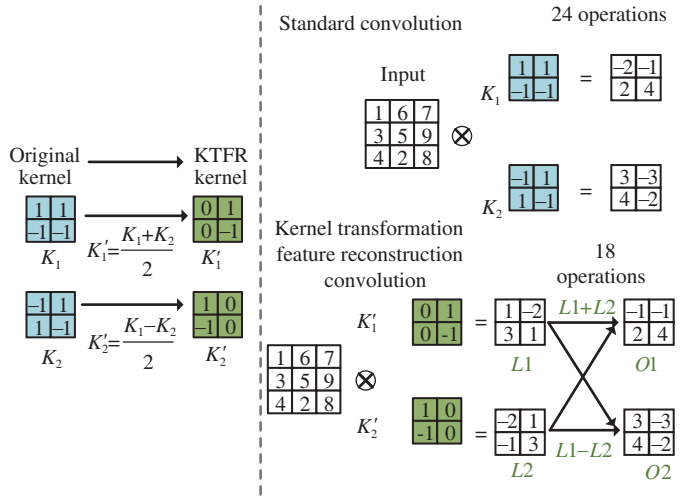

图 25 (网络版彩图) 基于滤波器拆分特征重建的卷积计算

Figure 25 (Color online) The process of kernel transformation feature reconstruction convolution

不同时, 每个 PE 的负荷也会不同, 我们会将稀疏性最大的卷积核和稀疏性最小的卷积核运算统一到一个 PE 计算任务上去; 次大的和次小的再统一分配到另外一个 PE 上去; 通过这样的方式对 PE 间的计算任务进行有效的分配调度; 而针对同一个 PE 内部来说, 在每个计算时钟下, 进行通道维度的累加由于稀疏性不同而产生不同数量的非零值加法次数. 那么针对这类情况, 我们采用 zero-skipping 的零值跳转机制来平衡稀疏化带来的问题, 整体来看这样的策略能够有效解决由稀疏化带来的负载不均衡问题, 提升了资源利用率. 此外, 一种联合优化方法离线地为每一个卷积层的每一个卷积核寻找最优的卷积计算方法, 最小化了功耗和计算时间, 使得能效最大化.

6 结语

人工智能芯片作为硬件基础, 将不断推动人工智能技术的进步. 目前尚没有真正意义上的通用 AI 芯片诞生, 而基于可重构计算架构的软件定义芯片 (software defined chip) 可以让芯片的计算能力按照软件的需求来调整适应, 进而满足神经网络计算多样性的需求. 本文介绍了可重构神经网络计算芯片 Thinker 系列, 探讨了具备可重构特性的硬件架构, 它可以解决不同类型的神经网络运算. 并在该架构基础上, 探究了相应的数据访存优化方案来降低运算功耗. 而后, 将优化方案逐步拓展到存储系统, 不同于传统 RAM 的存储器和可重构架构配套实现新的神经网络加速框架. 针对低比特网络, 我们优化了卷积运算部分, 来实现高性能计算. 如今, 充足的算力得益于 Moore 定律的不断演进发展, 未来高性能芯片将大幅降低深度学习算法所需的计算时间和成本.

参考文献

1 Wei S J, Liu L B, Yin S Y. Reconfigurable Computing. Beijing: Science Press, 2014 [魏少军, 刘雷波, 尹首一. 可重构计算. 北京: 科学出版社, 2014]

2 He K M, Zhang X Y, Ren S Q, et al. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2016. 770–778

3 Chen Y H, Krishna T, Emer J S, et al. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J Solid-State Circ, 2017, 52: 127–138

4 Gao C, Neil D, Ceolini E, et al. Deltarnn: a power-efficient recurrent neural network accelerator. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2018. 21–30
5 Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014. ArXiv:1409.1556
6 Szegedy C, Liu W, Jia Y Q, et al. Going deeper with convolutions. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015
7 He K M, Zhang X Y, Ren S Q, et al. Delving deep into rectifiers: surpassing human-level performance on imageNet classification. In: Proceedings of IEEE International Conference on Computer Vision, 2016. 1026–1034
8 Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit. 2017. ArXiv:1704.04760
9 Donahue J, Hendricks L A, Guadarrama S, et al. Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of Computer Vision and Pattern Recognition, 2015
10 Yin S Y, Ouyang P, Tang S B, et al. A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications. In: Proceedings of Symposium on VLSI Circuits, 2017
11 Moons B, Uytterhoeven R, Dehaene W, et al. 14.5 envision: a 0.26-to-10 TOPS/W subword-parallel dynamic-voltageaccuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI. In: Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), 2017. 246–257
12 Yan J, Yin S, Tu F, et al. GNA: reconfigurable and efficient architecture for generative network acceleration. IEEE Trans Comput-Aided Des Integr Circ Syst, 2018, 37: 2519–2529
13 Tu F B, Yin S Y, Ouyang P, et al. Deep convolutional neural network architecture with reconfigurable computation patterns. IEEE Trans VLSI Syst, 2017, 25: 2220–2233
14 Chen T S, Du Z D, Sun N H, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. ACM SIGPLAN Notice, 2014, 49: 269–284
15 Zhang C, Li P, Sun G Y, et al. Optimizing fpga-based accelerator design for deep convolutional neural networks. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2015. 161–170
16 Johnson J, Alahi A, Feifei L. Perceptual losses for real-time style transfer and super-resolution. In: Proceedings of European Conference on Computer Vision, 2016
17 Ledig C, Theis L, Huszar F, et al. Photo-realistic single image super-resolution using a generative adversarial network. 2016. ArXiv:1609.04802
18 Tu F B, Wu W W, Yin S Y, et al. RANA: towards efficient neural acceleration with refresh-optimized embedded DRAM. In: Proceedings of the 45th Annual International Symposium on Computer Architecture, 2018. 340–352
19 Chi P, Li S C, Xu C, et al. PRIME: a novel processing-in-memory architecture for neural network computation in reram-based main memory. In: Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA), 2016. 27–39
20 Yin S Y, Ouyang P, Yang J X, et al. An ultra-high energy-efficient reconfigurable processor for deep neural networks with binary/ternary weights in 28 nm CMOS. In: Proceedings of Symposia on VLSI Technology and Circuits, 2018

Research on low-power neural network computing accelerator

Jiale YAN[1], Ying ZHANG[1], Fengbin TU[1], Jianxun YANG[1], Shixuan ZHENG[1], Peng OUYANG[1], Leibo LIU[1], Yuan XIE[2], Shaojun WEI[1] & Shouyi YIN[1*]

1. *Institute of Microelectronics, Tsinghua University, Beijing* 100084*, China*;
2. *Department of Electrical and Computer Engineering, University of California, Santa Barbara* CA 93106*, USA*
* Corresponding author. E-mail: yinsy@tsinghua.edu.cn

Abstract Artificial intelligence has aroused a global upsurge, which covers image recognition, video retrieval, speech recognition, automatic driving, and several other significant applications. As for artificial intelligence algorithms, neural network algorithms play a crucial role and have attracted considerable attention from numerous researchers. Moreover, neural networks have the characteristics of high flexibility, complex computation, and a large amount of data; which also indicates the requirements of high performance, low-power consumption, and flexibility for hardware computing platforms. This study aims to propose a reconfigurable hardware architecture to meet the flexibility requirements of a neural network. Based on the proposed architecture, the corresponding data access optimization schemes are explored to reduce the power consumption. In the optimization of the storage system, an acceleration scheme of neural network based on eDRAM and ReRAM scheme, which is used for computing and storage integration, satisfy the requirement of neural network computing. Regarding high-performance computing, we have proposed convolution optimization schemes based on integral and filter splitting feature reconstruction to enable low bit neural network operations to meet high-performance requirements.

Keywords artificial intelligence, neural network algorithms, neural network accelerator, reconfigurable hardware architecture, low power

Jiale YAN received his B.S. degree in electronic science and technology from the Harbin Institute of Technology, Harbin, China, in 2016. He is currently pursuing his master's degree at the Institute of Microelectronics, Tsinghua University, Beijing, China. His current research interests include deep learning, computer architecture, and very large scale integration design.

Ying ZHANG received her B.S. degree in electronic science and technology from the Northwest University, Xi'an, China, in 2017. She is currently pursuing her master's degree in electronics science and technology at the Tsinghua University. Her current research interests include deep learning, signal processing, and speech enhancement.

Fengbin TU received his B.S. degree in electronic science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2013. He is currently pursuing his Ph.D. degree at the Institute of Microelectronics, Tsinghua University, Beijing. His current research interests include deep learning, computer architecture, very large scale integration design, approximate computing, and reconfigurable computing.

Shouyi YIN received his B.S., M.S., and Ph.D. degree in electronic engineering from the Tsinghua University, Beijing, China, in 2000, 2002, and 2005, respectively. He was a research associate with the Imperial College London, London, U.K. He is currently serving at Institute of Microelectronics, Tsinghua University, as an associate professor. His current research interests include reconfigurable computing, mobile computing, and SoC design.