高技术通讯 2022 年 第32 卷 第8 期:773-788

doi:10.3772/j.issn.1002-0470.2022.08.001

面向 CNN 加速器的一种建模与优化设计方法研究①

祁玉琼②*** 张明喆* 吴海彬* 叶笑春③*

(*中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 本文提出了一种卷积神经网络(CNN)加速器性能与能耗通用评估模型(CNNGModel)。CNNGModel 通过 CNN 加速器中不同结构的具体设计,可以估计该加速器处理不同任务时需要的时间与能耗。在硬件工程师使用硬件描述语言实现该加速器前,通过 CNNGModel 可以提前判断当前 CNN 加速器的设计是否符合应用需求,从而减少后续不必要的工作量。在实验部分,首先设计并实现了 3 个 CNN 加速器;其次分析对比 CNNGModel、模拟器 VTA 以及仿真综合 3 种方式得到的每个加速器在处理不同 CNN 时的多项结果,其中对于处理时间的估计,CNNGModel 与仿真综合的差距低至 3.0%,对于功率,差距则低至 6.5%;最后依据 CNNGModel,从能耗和性能两方面给出了多项 CNN 加速器优化策略。

关键词 卷积神经网络(CNN);CNN 加速器;性能与能耗通用评估模型(CNNGModel);CNN 加速器优化策略

0 引 言

目前已有多种用于解决各类识别与分类相关问题的卷积神经网络(convolutional neural network, CNN)被提出,例如 AlexNet[1]、VGG[2] 及 ResNet[3] 等。同时,针对一些特定领域也演化出了很多 CNN 相关的神经网络。例如用于生成图像数据集、实现文字到图像转换的生成式对抗网络(generative adversarial network,GAN),用于物体识别、人脸识别的深度迁移学习算法(transfer learning),用于自动驾驶和安防监控的目标检测算法(YOLO 等)以及用于机器人、游戏自动化的强化学习算法(DQN 等)。因此可以看到,由于 CNN 可以有效地处理各类复杂问题,其在现实生活中被各行各业所广泛使用。

为了能够有效地处理各种基于 CNN 的算法,越来越多的 CNN 加速器被提出,例如 Eyeriss[4]、DaDianNao[5]、ShiDianNao[6] 等。大部分 CNN 加速器的存储结构分为 3 层,即片外存储(DRAM)、片上存储(SRAM)以及计算单元(processing element, PE)中的寄存器(Register),且不同加速器中相同存储结构的大小、容量都各不相同。此外不同加速器的 PE 阵列(PE array)大小和对应的计算能力也均不同。由此可见 CNN 加速器的设计包括很多方面,是一个相对复杂的工作。

当 CNN 加速器设计完成后,硬件工程师需要使用硬件描述语言(Verilog 或 VHDL)对其进行基于寄存器传输级的抽象和实现。这一步的工程量巨大,需要耗费大量的时间以及人力。此后通过仿真工具和逻辑综合器对寄存器传输级的设计进行仿真和综合,可以得到该加速器的性能、能耗以及面积的

① 国家自然科学基金(61732018,61872335,61802367),中国科学院战略性先导科技专项(XDC05000000),中国科学院国际伙伴计划(171111KYSB20200002)资助项目。

② 女,1993 年生,博士生;研究方向:计算机系统结构;E-mail:qiyuqiong1993@qq.com。

③ 通信作者,E-mail:yexiaochun@ict.ac.cn。

(收稿日期:2021-04-15)

评估。

不同应用场景对性能以及能耗的要求都不同,例如数据中心主要用于处理大量后端应用,因此其更关心处理器的性能功耗比[7];边缘物端如 IoT,则对处理器功耗的要求更为严格[8];而在高实时性要求的智能驾驶场景下,处理器的性能则是重点关注指标。因此针对特定领域,若设计出的加速器其仿真综合后得到的性能和能耗指标不符合应用需要,硬件工程师则需要重新设计 CNN 加速器的结构,再用硬件描述语言修改相应的寄存器传输级设计。

加速器的执行时间或能耗可以通过多种策略进行优化。若选择增加静态随机存储器(static random access memory, SRAM)的带宽来减少能耗,则不仅要修改 SRAM 的结构,同时还需要增加 PE 阵列中的寄存器大小。若选择增加 PE 阵列大小来减少执行时间,则需要修改片上 NoC 以及 SRAM 的带宽。所以当 CNN 加速器某个结构设计发生改变后,加速器中其他部分的结构设计也会有相应的改变。因此,为了符合应用的性能与能耗需求,在探索 CNN 加速器结构设计方案时,每一次方案的调整都会造成硬件工程师大量修改多个部件、不同部件间连线以及相关信号的寄存器传输级实现。

为了能够尽可能地减少或避免上述调整与修改,本文提出了一种 CNN 加速器性能与能耗通用评估模型(general performance and energy consumption model for CNN accelerator, CNNGModel)。CNNGModel 通过 CNN 加速器中不同结构的具体设计,可以估计该加速器处理不同任务时所需的时间与能耗。从而在硬件工程师使用硬件描述语言实现该加速器前,提前判断当前 CNN 加速器的设计是否符合应用需求,减少后续不必要的时间与人力的浪费。最后依据 CNNGModel,本文还给出了 CNN 加速器优化策略,以此帮助硬件架构师对加速器架构的调整。

本文的主要贡献包括以下 5 个方面。

(1)基于 CNN 加速器的通用硬件结构,提出加速器中数据传播路径的概念。

(2)提出了 CNN 加速器性能与能耗通用评估模型(CNNGModel),并实现了其中的核心部分——通用模型库,同时规定了模型的输入与多层输出。

(3)定义了 CNN 加速器中的 3 种数据传播方式,用于 CNNGModel 的输入。

(4)实验部分首先设计实现了 3 个 CNN 加速器,其次分析对比了通过 CNNGModel、模拟器 VTA 以及仿真综合 3 种方式得到的每个加速器在处理不同 CNN 时的多项结果。

(5)依据 CNNGModel,给出了对于 CNN 加速器优化的多项策略。

1 相关工作

本节首先介绍目前被广泛关注的一些 CNN 加速器,其次给出目前已有的 CNN 加速器性能评估模型,并对这些模型进行分析与总结。

DaDianNao[5]主要用于处理大型的 CNN,具有伸缩性。当单层 CNN 的参数量超过 DaDianNao 单片存储极限时,DaDianNao 通过其伸缩性,可以将单层 CNN 划分到多个芯片上执行。ShiDianNao[6]则更倾向于物端场景,它直接与视频图像传感器相连接,直接以传感器采集到的数据作为加速器的输入。因此 ShiDianNao 中输入访存的数据量更少,片上配置的 SRAM 更加精巧,常被用来处理较小规模的 CNN 传输。Eyeriss[4]则是一个高能效、可重配的神经网络加速器。其核心设计是 RowStationary 数据流,可以有效减少数据搬运带来的大量时间和能耗。Origami[9]利用其提出的一种新型卷积网络架构,使加速器的性能达到 TOP/s 量级的同时,保持较高的面积和能耗效率。NeuFlow[10]是一种运行时可重配置的数据流结构,一般用于嵌入式系统,可以处理人脸识别、场景分割等。

当使用分块技术和量化技术对 CNN 加速器进行优化时,可以对加速器的性能进行模拟,同时只针对低位宽的 CNN 加速器[11]。SCALE-Sim[12]是对基于 Systolic 架构的 CNN 加速器进行建模。其中规定,在 PE 阵列中数据只能在同一行或同一列中传播。因此对于一些基于 RowStationary 等数据流的加速器(这些加速器中的数据需要在不同行(列)间传播),SCALE-Sim 无法对其性能进行建模。文献[4]

主要估计了 CNN 加速器内的传输能耗。首先该工作面向的是不同数据流,而不是不同的加速器。其次在进行能耗估计前,需要提前假设 CNN 中 3 种类型数据在加速器中的传输路径。文献[13]提出了一种针对 CNN 加速器中数据划分和调度的分析模型,因此并不能用来估计所有 CNN 加速器的性能、能耗等。本文提出的 CNNGModel 是依据不同加速器处理 CNN 的真实过程设计的,主要考虑了以下 3 点:(1)CNN 3 种类型数据在加速器中的传输路径;(2)路径中上下层存储的大小;(3)加速器对于某一路径的数据处理模式。因为 CNNGModel 可以给出不同加速器在处理不同 CNN 时消耗的时间与能耗,所以可用于每个真实的 CNN 加速器。

2 CNN 加速器的数据传播路径

本节首先对 CNN 卷积神经网络进行介绍,其次给出 CNN 加速器的通用硬件结构,最后介绍 CNN 加速器中的数据传播路径。

2.1 CNN 网络参数

CNN 网络主要由卷积层(CONV)、池化层(POOL)以及全连接层(FC)构成。这些层均包含 2 种数据类型,即输入特征图(input feature map, ifmaps)及输出特征图(output feature map, ofmaps)。其中 ofmaps 是通过累加中间结果(partial sums, psums)得到的。此外卷积层和全连接层还包含第 3 种数据类型滤波器(filters)。图 1 给出卷积层示例,其 ifmaps 的大小是 $I \times I \times C$, 个数为 N;ofmaps 的大小是 $O \times O \times M$, 个数为 N;filters 的大小是 $F \times F \times C$, 个数为 M。表 1 给出了图 1 中各个参数的定义。

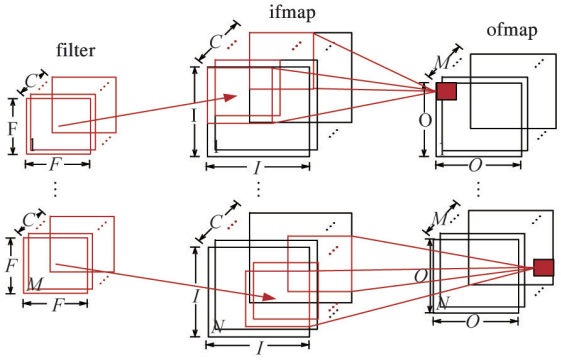


图 1 CNN 卷积层

表 1 CNN 的参数

参数	定义
C	ifmaps 和 filters 的通道数
M	filters 的个数/ ofmaps 的通道数
I	ifmaps 的边长
O	ofmaps 的边长
F	filters 的边长

2.2 CNN 加速器的通用结构

对于大多数 CNN,卷积层和全连接层的计算量在整个网络中占比很大(超过 90%),所以本文只对 CNN 中的卷积层进行分析(全连接层与卷积层的计算过程相同)。本节首先介绍 CNN 加速器的 2 种常见数据处理方式,其次给出每种数据处理方式对应的 CNN 加速器通用结构。

图 2 给出了目前比较常见的 2 种 CNN 加速器数据处理方式。其中单卷积(single convolution, Sconv)指加速器执行过程中,每次循环包含的数据为一个二维卷积;多卷积(multiple convolution, Mconv)指每次循环包含的数据为多个二维卷积(T_m 表示每次处理的 filters 的个数, T_c 表示每次处理 ifmaps 或 filters 中的通道数)。本文将处理器每次循环包含的数据称为一个基本处理单元(BasicUnit)。图 2 中,对于 Sconv 的一个 BasicUnit,filters 大小为 $F \times F$, ifmaps 的大小为 $I \times I$, ofmaps 大小为 $O \times O$;对于 Mconv 的一个 BasicUnit,filters 的大小为 $F \times F \times T_m \times T_c$, ifmaps 的大小为 $I \times I \times T_c$, ofmaps 大小为 $O \times O \times T_m$(对于 Sconv,其 BasicUnit 中包含的数据量可能会更小,但只会涉及 ifmaps 中一个通道的相关数据)。

基于图 2 的 2 种数据处理方式,图 3 给出了相应的加速器通用结构。图 3(a)为 Sconv 类型加速器的通用结构,其中包含加速器片上结构(accelerator chip)和外部存储(external memory chip, EXMC)。在 accelerator chip 中,处理 PE array 中的每个 PE 只有一个乘加计算部件(multiply-accumulate unit, MAC)。对于 PE 中的存储部件寄存器(Register),一共有 2 种结构,一种是分散式存储(dispersive register, DR),即每个 PE 中都有寄存器;另一种是集中式存储(concentrated register, CR),即将每个 PE 中的寄

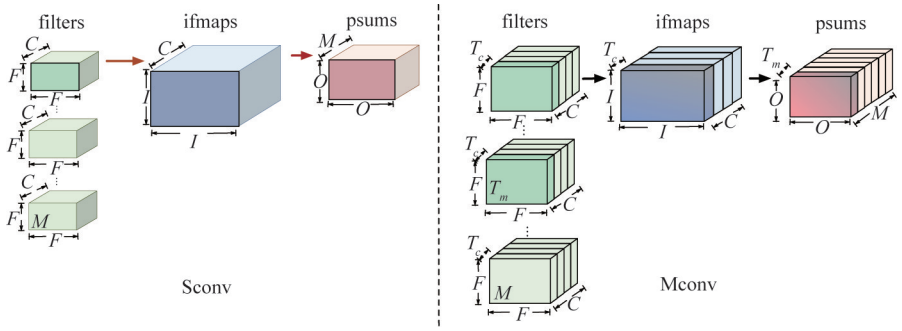


图 2　2 种数据处理方式

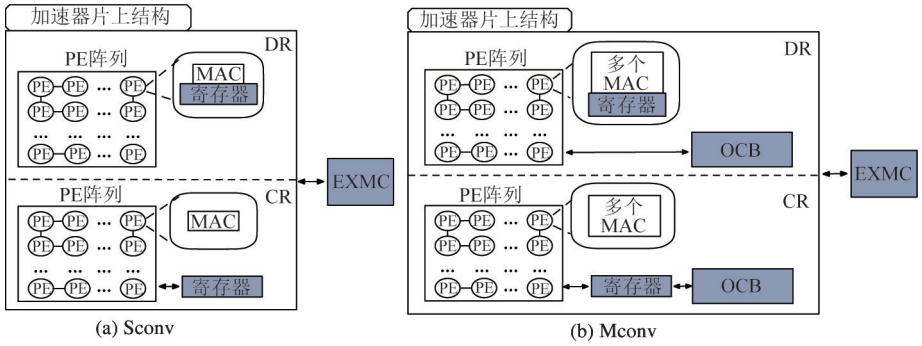


图 3　CNN 加速器通用结构

存器集中起来,统一放到 PE array 外部。需要说明的是,不同加速器的寄存器大小以及片上网络(network on chip,NoC)结构都不同。

图 3(b)为 Mconv 类型加速器的通用结构图。其与 Sconv 的不同处为:(1)每个 PE 不只有一个 MAC;(2)片上存储结构一共有 2 层,一层为片上缓存(on-chip buffer, OCB),用来缓存 ifmaps、ofmaps 以及 filters;另一层为寄存器。表 2 给出了一些典型的 CNN 加速器的数据处理方式和寄存器类型。

表 2　典型 CNN 加速器分类

	DR	CR
Sconv	ShiDianNao[6], CNP[14], NewFlow[10]	CE[15]
Mconv	Eyeriss[4], FPGA[16]	Origami[9]

2.3　数据传播路径

根据图 3 中加速器的存储结构,表 3 给出了数据在这些存储结构中的传播路径。这里需要说明的是:(1)当加速器片上结构中有寄存器时,路径中的"→PE"表示数据传输到寄存器中(CR 或 DR),否则表示数据直接传输到 PE 中的计算单元(MAC);(2)"←PE"表示计算结果从 DR 或者 MAC 中送出;(3)对于传播路径④"Among PEs",当加速器中的寄存器类型为 DR 时,其表示数据在 PE 间的寄存器中传播。当加速器中的寄存器类型为 CR 时,其表示 PE 阵列对 CR 的访问;(4)本文忽略数据在 DR 和 MAC 之间传播所需要的时间和能耗;(5)对于 EXMC→PE 和 EXMC←PE,上层存储均是 EXMC,下层存储均是 PE。

表 3　数据传播路径

序号	路径	序号	路径
①	EXMC→PE	②	EXMC→OCB
③	OCB→PE	④	Among PEs
⑤	EXMC←PE	⑥	EXMC←OCB
⑦	OCB←PE		

传播路径④"Among PEs"共有 3 种类型,ofmaps 传播(ofmaps propagation, OP)、ifmaps 传播(ifmaps propagation, IP)和多类型数据传播(multiple propagation, MP)。每个加速器中只能存在其中一个类型。在 OP 类型加速器中,对于一个 BasicUnit 中的数据,常见的情况为 ifmaps 被直接送到对应 PE;fil-

ters 被提前固定在 PE 中;ofmaps 的中间结果在 PE 间传播累加,最终结果在其遍历完所有 PE 后生成。NeuFlow[10]、CNP[14]均属于 OP。

在 IP 类型加速器中,对于一个 BasicUnit 中的数据,常见的情况为 filter 被直接送到对应 PE,ofmaps 固定在 PE 中直到得到最终结果,ifmaps 在 PE 间传播。ShiDianNao[6]即属于 IP。MP 类型加速器指同时会有多种类型的数据在 PE 间传播。CE[15]、Origami[9]、Eyeriss[4]均属于 MP。表 4 给出了一些 CNN 加速器的结构以及其中的数据传播路径。

3 CNN 加速器通用评估模型

图 4 为本文所提出的 CNN 加速器评估模型(CNNGModel)。当加速器在处理每层 CNN 网络时,评估模型需要做以下工作:(1)加速器相关参数和数据传播路径会作为模型的输入,其中每条数据传播路径为模型的处理单元。(2)通过调用模型库中的估算模型,得到每条数据传播路径的中间输出。(3)合并统计数据传播路径的中间输出,得到最终输出。

表 4 一些典型加速器的结构与数据路径

			加速器	ifmaps	filter	psum
Sconv	CR	MP	CE	①④	①	⑤④
	DR	IP	ShiDianNao	①④	①	⑤
		OP	NeuFlow	①	①	⑤④
Mconv	CR	MP	Origami	②③④	①④	⑤④
	DR	MP	Eyeriss	②③④	①	⑥⑦④

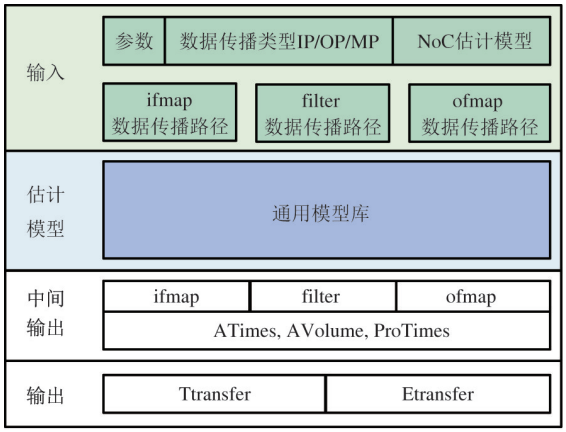


图 4 CNN 加速器通用评估模型

3.1 评估模型输入

本文评估模型的输入包括 4 部分:(1)CNN 加速器参数(表5和表6);(2)CNN加速器内部数据

表 5 CNN 加速器中的一些硬件参数(I)

	参数	变量	定义	
加速器结构	存储结构	*I(F/O)EXMC*	*size*	EXMC 存储 ifmaps、filters 或 ofmaps 的大小
			bandwidth	EXMC 对于 ifmaps、filters 或 ofmaps 的带宽
		I(F/O)OCB	*size*	OCB 存储 ifmaps、filters 或 ofmaps 的大小
			bandwidth	OCB 对于 ifmaps、filters 或 ofmaps 的带宽
			double _ buffer	存储 ifmaps、filters 或 ofmaps 的 OCB 是否有双缓存
		I(F/O)REG	*size*	寄存器存储 ifmaps, filters 或 ofmaps 的大小
			double _ buffer	存储 ifmaps、filters 或 ofmaps 的寄存器是否有双缓存
	计算部件	*ARRAY*	*size*	PE 阵列大小
		PECAP	*macs*	每个 PE 中 MAC 的数
		GROUP	*num*	PE 阵列的组数
			cycle	每组 PE 计算一个 BasicUnit 的时钟周期数
	片上网络	*NOC*	*i(f/o)num*	ifmaps、filters 或 ofmaps 的 NoC 吞吐量
数据处理方式	BasicUnit	*BASICUNIT*	*i(f/o)size*	一个 BasicUnit 中 ifmaps、filters 以及 ofmaps 的大小
			macs	计算一个 BasicUnit 所需的 MAC 量
			cycles	计算一个 BasicUnit 所需的时钟周期数
			num	一个 CNN 层包含的 BasicUnit 总数
	psums 处理方式	*PSUMS*	*macs*	一个 psum 从 PE 阵列到上层存储所需的 MAC 量
			pe	一个 PE 能够同时处理的 psums 的量

表 6 CNN 加速器中的一些硬件参数(II)

参数	变量	定义
I(F/O)EXMC	*accessE*	外部存储单次访问能耗
I(F/O)OCB	*accessE*	片上缓存单次访问能耗
I(F/O)REG	*accessE*	寄存器单次访问能耗
Frequency	*Hz*	加速器的频率

传播类型,分为 IP、OP 和 MP3 类(详细介绍见 2.3 节);(3)CNN 加速器中 3 种数据类型的传播路径(详细介绍见 2.3 节);(4)NoC 估计模型[17]用于估计“Among PEs”路径带来的时间与能耗开销。

表 5 中的参数为加速器设计时需要考虑的参数,主要用于在评估模型中得到中间输出。其可以分为两类:加速器结构参数以及数据处理方式参数。其中第 1 类包括 3 部分:加速器存储结构、计算部件以及片上网络的相关参数。第 2 类包括加速器中 BasicUnit 以及 psums 处理方式的相关参数。表 6 中的参数为加速器中不同部件的实际参数,例如频率、不同存储结构的单次访问能耗等。这些参数主要用于在评估模型中得到最终输出,使用 *Parameters. Variable* 来表示这些参数(例如 *IOCB. size*)。

3.2 评估模型输出

基于 3.1 节的输入,如图 5 所示,本文的评估模型将首先给出中间输出 ATimes、AVolume 以及 ProTimes;其次依据中间输出最终得到目标 CNN 加速器的性能以及能耗。

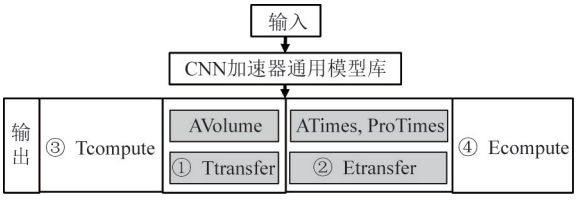


图 5 CNN 加速器评估模型输出

首先对于中间输出给出以下定义:当加速器在处理一个 BasicUnit 时,数据在每条传播路径中对上层存储的访问次数被定义为 ATimes;在每条传播路径中,下层存储在初始化时获得的数据量被定义为 AVolume;当路径为“Among PEs”时,PE 间传播的数据量被定义为 ProTimes。其次对于最终输出给出以下定义:Ttransfer 为由数据传输带来的时间;Etransfer 为数据传输所带来的能耗;Tcompute 为由计算带来的时间;Ecompute 为由计算带来的能耗。

因为大多数 CNN 的输入输出数据量都很大,如 AlexNet 过滤器的数据量为 61 MB、Overfeatfast 为 146 MB、VGG16 为 138 MB,所以当加速器在处理这些 CNN 时,无论是从片外到片上,还是只在片上,都会存在大量的数据搬移。从表 7 可以看到,大部分 CNN 加速器的存储结构面积均超过总面积的一半,其中 Eyeriss 甚至高达 81.1%。所以 CNN 加速器工作时,存储间的数据传输将消耗大量的时间与能耗。因此本文将重点分析 CNNGModel 中的 Ttransfer 与 Etransfer。

表 7 CNN 加速器存储结构面积

	Origami[9]	ShiDianNao[6]	SCNN[18]	Eyeriss[4]
Register	38.5%	\	32.5%	62.3%
SRAM	23.6%	80.18%	25.2%	18.8%
总量	62.1%	>80.18%	57.7%	81.1%

Ttransfer 通过 AVolume 计算得到。因为只需要考虑数据在不同存储结构之间传输所需的时间中不能被计算时间所覆盖的部分,而造成这部分不能被覆盖的时间的数据量为 AVolume。对于 Etransfer,通过 CACTI-6.5 可以得到单次访问 DRAM 的能耗,同时可以获取不同工艺下,单次访问不同存储结构(DRAM、SRAM、Register 等)产生的能耗之间的相对比值。因此进一步分别统计 3 种数据类型(ifmaps、filters、ofmaps)对于不同存储结构的访存次数(ATimes),就可以计算出 Etransfer。Tcompute 可以根据硬件的峰值性能得到,Ecompute 将通过任务的计算量以及硬件工艺参数得到。

3.3 CNN 加速器通用模型库

CNNGModel 的基本处理单元为目标加速器中的每条数据传播路径。因此当加速器在处理每层 CNN 网络时,CNNGModel 将会依据如下步骤工作:首先针对每一条数据传播路径选择通用模型库中对应的估算模型;其次依据输入的 CNN 加速器各项参数(表 5),通过估算模型得到每条路径的 ATimes、ProTimes 以及 AVolume;最后整合所有路径的计算结果,依据表 6 中的参数得到 Ttransfer 与 Etransfer。

3.3.1 ATimes、ProTimes 以及 AVolume

根据 2.3 节,目标加速器中一共会存在 12 种路径。将其分为 3 类:(1) IFTransfer(6 条):ifmaps 和 filters 的路径 EXMC→PE、EXMC→OCB 和 OCB→PE;(2)OTransfer(3 条):ofmaps 中间结果(psums)的路径 EXMC←PE、EXMC←OCB 和 OCB←PE;(3)PETransfer(3 条):ifmaps、ofmaps 以及 filters 的路径 AmongPEs。接下来将分别介绍上述 3 类路径的 ATimes、AVolume 和 ProTimes 估算模型。图 6 给出了通用模型库中所有路径的子路径以及对应的估算模型。

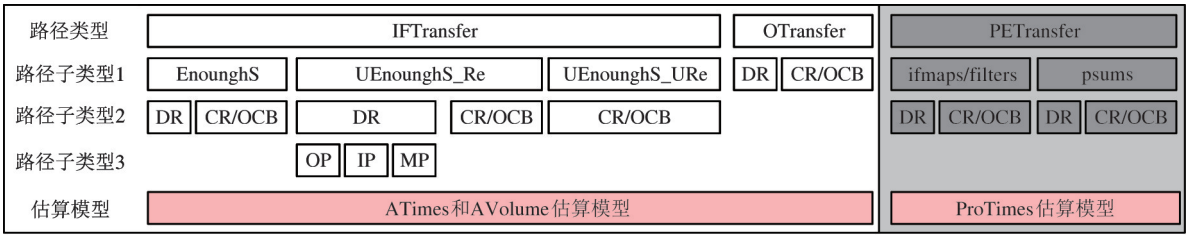


图 6 CNNGModel 通用模型库的内部结构

3.3.1.1 IFTransfer

在这类路径中,当 EXMC→PE 和 OCB→PE 的下层存储结构为 DR 时,本文规定所有 PE 中寄存器的总大小为下层存储大小。IF _ Transfer 的 ATimes 以及 AVolume 估算模型可以依据以下 3 类子路径讨论。

(1) 下层存储充足(EnoughS)

该类子路径下层存储的大小大于或等于一个 BasicUnit 中 ifmaps 或 filters 的数据量(依据路径中的数据类型来判断是 ifmaps 还是 filters)。对于该路径,只有当一个 BasicUnit 中的数据都存储到其下层存储后,加速器才会开始工作。所以该路径的 AVolume 等于 BasicUnit 中对应数据的数据量。对于 ATimes,若路径下层存储为 DR,则与 NoC 的吞吐量有关;若为 CR 或 OCB,ATimes 与上层存储带宽相关,因此:

$$AVolume = BASICUNIT.isize \text{ 或 } BASICUNIT.fsize \tag{1}$$

对于 DR 有

$$ATimes = \frac{AVolume}{NOC.inum \text{ 或 } NOC.fnum} \tag{2}$$

对于 CR/OCB 有

$$ATimes = \frac{AVolume}{upper\ storage\ bandwidth} \tag{3}$$

(2)下层存储不足但数据可重复使用(UEnoughS _ Re)

该类子路径下层存储的大小小于一个 BasicUnit 中的数据量,但每个数据只需要从上层存储中读取一次。这里将分为两大类对该子路径的估算模型进行讨论,一类为路径的下层存储是 DR,另一类为下层存储是 CR 或 OCB。

首先对下层存储为 DR 的路径进行讨论。这里对下层存储为 DR 的路径归纳了 3 类数据存取模式,并给出了对应的估算模型(未来若有新的模式,也可以依据本文的评估模型得到对应的估算模型)。

不同 PE 每次从上层存储得到完全相同的数据,且每次存取的数据量也相同,一般存在于 OP 类型加速器。对于属于这种数据存取模式的路径,对应加速器中的 PE 一般没有存储相应数据的寄存器,因此数据只能送到 PE 中的计算单元。所以 AVolume 与每个 PE 中计算单元的个数相关。ATimes 则与一个 BasicUnit 中对应数据的数据量以及 PE 每次得到的数据量 AVolume 相关。综上,对于该类数据存取模式的路径:

$$AVolume = PECAP.macs \tag{4}$$

$$ATimes = \frac{BASICUNIT.isize \text{ 或 } BASICUNIT.fsize}{AVolume} \tag{5}$$

不同 PE 每次从上层存储得到完全不同的数据,且每个 PE 每次得到的数据量不一定相同,一般存在于 IP 类型加速器。对于拥有这种数据存取模

式路径的加速器,其中 PE 间会互相传输数据,以此实现数据的重复利用。因为这类加速器在工作时,每个周期都有新数据读入,所以这里的 ATimes 等于完成一个 BasicUnit 的周期数。而 AVolume 与每个 PE 初始化时需要的数据量以及加速器中 PE 的个数有关。综上,对于该类路径,有:

$$ATimes = BASICUNIT.cycles \tag{6}$$

$$AVolume = PECAP.macs \times ARRAY.size \tag{7}$$

同组 PE 每次从上层存储得到相同的数据,且每次得到的数据量不同,一般存在于 MP 类型加速器。这种数据存取模式的 ATimes 一般与 PE 的组数 *GROUP. num* 以及每组得到数据的次数 *GROUP. cycle* 相关。其中 *GROUP. cycle* 与每个 PE 中相关数据的寄存器大小以及该数据本身的大小相关(根据 NoC 结构的设计,这里假设不同组的 PE 不会同时取数据)。所以

$$ATimes = GROUP.num \times GROUP.cycle \tag{8}$$

而 AVolume 与 PE 中相关数据的寄存储大小相关,则

$$AVolume = IREG \text{ 或 } FREG.size \tag{9}$$

接下来本文讨论 UEnoughS _ Re 类型路径中,下层存储为 CR 或 OCB 的情况。因为属于 UEnoughS _ Re 的路径的下层存储大小小于一个 BasicUnit 中的数据量,所以 CR 或 OCB 中的数据会被多次替换,因此 ATimes 等于替换次数。这里令替换次数为 *TimeRelpace*1,该参数与加速器的设计相关。例如对于 Origami 中 ifmaps 的路径 EXMC→OCB,每次滑动窗口在 ifmaps 上滑动时,OCB 中部分 ifmaps 的值将被替换,因此该路径的 $TimeRelpace1 = O \times O$, AVolume 为初始化时送入下层存储的数据量即下层存储的大小。综上,对于该类数据存取模式的路径:

$$ATimes = TimeRelpace1 \tag{10}$$

$$AVolume = I/FOCB.size \text{ 或 } I/FREG.size \tag{11}$$

(3)下层存储不足且数据不重复使用(UEnoughS _ URe)

该类子路径下层存储的大小小于一个 BasicUnit 中的数据量,且每个数据需要从上层存储中读取多次。该类型路径的下层存储一般为 CR 或 OCB,因此其中的数据会被多次替换,这里设替换次数为 *TimeRelpace*2, 与 *TimeRelpace*1 相同,其与加速器内部设计相关。因此:

$$ATimes = TimeRelpace2 \tag{12}$$

$$AVolume = I/FOCB.size \text{ 或 } I/FREG.size \tag{13}$$

3.3.1.2 OTransfer

本节将讨论 OTransfer 类型路径中 ATimes 以及 AVolume 的估算模型。因为 OTransfer 路径中的数据为 ofmaps 的中间结果 psums,而在初始化时 OTransfer 类型的路径不会有 psums 产生,所以:

$$AVolume = 0 \tag{14}$$

其次,对于 ATimes,若该类型路径的下层存储是 PE,则

$$ATimes = \frac{BASICUNIT.osize}{NOC.onum} \tag{15}$$

因为此时 psums 是从 DR 或 MAC 送出,所以 ATimes 与 BasicUnit 中 psums 的大小以及 NoC 有关。

对于 ATimes,若该类型路径的下层存储是 OCB,则

$$ATimes = 0 \tag{16}$$

这是因为 OCB 是用来存储中间结果的,即只有最终结果才会被从 OCB 送出到 EXCM,而一个 BasicUnit 产生的结果为 psums。

3.3.1.3 PETransfer

在本节中,将讨论 PETransfer 类型路径中 ProTimes 的估算模型。

首先对于 ifmaps 和 filters,依据加速器中寄存器的类型,ProTimes 的计算方式一共可以分为 2 类。若加速器中的寄存器为 DR,根据 3.2.1 节的分析,因为在下层存储为 DR 的路径中每个数据只需要从上层存储读取一次,所以 PE 间被重复利用的数据都是通过 PE 间传播得到的。同时由于计算 1 个 MAC 需要 1 个 ifmap 和 1 个 filter,因此通过将计算 1 个 BasicUnit 所需的 MAC 数量与 1 个 BasicUnit 中 ifmaps 或 filters 的数据量相减,可以得到此处的 ProTimes。综上,有:

$$ProTimes = BASICUNIT.macs - BASICUNIT.isize(\text{或 } BASICUNIT.fsize) \tag{17}$$

若加速器中寄存器的类型为 CR,则意味着 PE 里没有存储,所以 PE 在每次计算时都需要访问

CR。因此 ProTimes 可以通过 PE 阵列单次计算能力以及计算 1 个 BasicUnit 需要的能力得到(PE 阵列的非充分利用将在后续的工作中讨论)。综上,有:

$$ProTimes = \lceil \frac{BASICUNIT.\ macs}{ARRAY.\ size \times PECAP.\ macs} \rceil \times \lceil \frac{ARRAY.\ size \times PECAP.\ macs}{NOC.\ inum \text{ 或 } NOC.\ fnum} \rceil \tag{18}$$

对于 psums,ProTimes 主要与 2 个参数相关。一个是单个 PE 计算 1 个 psum 时提供的 MAC 数量,可以通过 $\frac{PECAP.\ macs}{PSUMS.\ pe}$ 得到。另一个为 1 个 psum 从在 PE 阵列中的初始状态转变为送出到上层存储时的状态需要的总 MAC 量,用 *PSUMS. macs* 表示。因此:

$$ProTimes = PSUMS.\ macs/(\frac{PECAP.\ macs}{PSUMS.\ pe}) \times BASICUNIT.\ osize \tag{19}$$

3.3.2 Ttransfer 与 Etransfer

CNNGModel 将通过中间输出 ATimes、ProTimes 以及 AVolume 得到 Ttransfer 以及 Etransfer。

ProTimes 为“Among PEs”路径在一个 BasicUnit 中产生的在 PE 间传播的数据总量,其中每一个数据均需要通过 NoC 从某个 PE 传播到相应的目的 PE。因为每个数据在 PE 间传播的路线都不同,所以传播每个数据所带来的能耗与时间都不同。依据大多数 NoCs 使用的健忘路由算法[19],可以得到每个 PE 阵列中存在的所有数据传播路线。本文将考虑最差的情况,因此假设所有数据在 PE 间的传播路线均是 PE 间最长的数据传播路线,即一共经过 K 个 PE,则每个数据在 PE 间传播所消耗的能耗均为 $K \times I(F/O)REG.\ accessE$, 所消耗的时间均为 K 个 cycle。此外考虑到 NoC 中会出现拥塞情况,根据文献[17]中给出的拥塞最差情况模型,本文为每条传播路线增加了因拥塞造成的额外时间与能耗。

ATimes 为每条传播路径在 1 个 BasicUnit 中对上层存储的访问次数,因此通过将 ATimes 与对应路径上层存储结构的单次访存能耗相乘,可以得到每条路径因访问上层存储而消耗的能量。

AVolume 为每条传播路径在 1 个 BasicUnit 中,下层存储在初始化时获得的数据量。如果路径中的下层存储有双缓存,则加速器初始化时,由传输 ifmaps 和 filters 所造成的时间只与第一个 BasicUnit 中的 AVolume 相关,否则与所有 BasicUnit 相关。

算法 1 给出了计算 Etransfer 的详细步骤。(1)需要获得通用评估模型输入中所有的数据传播路径;(2)通过上一节的估算模型,可以得到对于一个 BasicUnit,每条路径的 ATimes 和 ProTimes;(3)根据 1 个卷积层中 BasicUnit 的总量,可以获得每条路径 ATimes 和 ProTimes 的总量;(4)根据对应路径 ATimes 或 ProTimes 的总量,获得每条路径消耗的能量;(5)通过将每条路径因访存消耗的能量相加,获得加速器在处理当前卷积层的 Etransfer。

算法 1 计算 Etransfer

Require: CNN 层(*ConvLayer*)
Ensure:当前 CNN 层的 *Etransfer*

```
for ConvLayer do
    得到所有路径的 path[N];
end for
for all path_i do
    if path_i ∈ (IFTransfer ∨ OTransfer) then
        得到 ATimes_i;
        TotalATimes_i = BasicUunit.num × ATimes_i;
        etransfer_i = TotalATimes_i × EXMC 或
        OCB.accessE;
    end if
    if path_i ∈ PETransfer then
        得到 ProTimes_i;
        TotalProTimes_i = BasicUunit.num × ProTimes_i;
        etransfer_i = TotalProTimes_i × (REG.accessE ×
        K + NoC);
    end if
end for
for i = 1; i < N; i ++ do
    Etransfer = ∑_{i=1}^{n} etransfer_i;
end for
```

算法 2 给出了计算 Ttransfer 的详细步骤。与计算 Etransfer 相同,这里也需要获得每个卷积层所有的传播路径以及累加每条路径所造成的传输时间。此外由于加速器中大多数传输时间可以被计算时间所覆盖,因此 Ttransfer 只考虑不能被计算时间所覆

盖的部分,其主要与 AVolume 相关。

算法 2　计算 Transfer

Require:CNN 层(*ConvLayer*)
Ensure:当前 CNN 层的 *Ttransfer*
for *ConvLayer* **do**
　得到所有路径的 *path*[*N*];
end for
for all $path_i$ **do**
　if $path_i \in IFTransfer$ **then**
　　得到 $AVolume_i$;
　　if (*IOCB. double _ buffer* = =1) ∨ (*FOCB. double _ buffer* = =1) ∨ (*IREG. double _ buffer* = =1) ∨ (*FREG. double_ buffer* = =1) **then**
　　$TotalAVolume_i = AVolume_i$;
　　else
　　$TotalAVolume_i = AVolume_i \times Basicunit.num$;
　　end if
　　$ttransfer_i = \frac{TotalAVolume_i}{EXMC\ 或\ OCB.bandwidth} / Frequency.HZ$;
　end if
　if $path_i \in PETransfer$ **then**
　　得到 $ProTimes_i$;
　　$TotalProTimes_i = ProTimes_i \times Basicunit.num$;
　　$ttransfer_i = \frac{TotalProTimes_i \times (K + NoCtime)}{Frenquency.HZ}$;
　end if
end for
for $i = 1; i < N; i++$ **do**
　$Ttransfer = \sum_{i=1}^{n} ttransfer_i$;
end for

4　实　验

本节设计了 3 个 CNN 加速器——Sconv-DR-OP、Sconv-CR-IP 和 Mconv-CR-MP 用于验证 CNNGModel 的有效性。3 个加速器基于的原型分别为 NeuFlow[10]、ShiDianNao[6] 以及 Origami[9]。实验中,首先使用开源模拟器 VTA[20] 实现了上述 3 个加速器;其次基于 TSMC 16 nm 技术,使用硬件描述语言 Verilog 和 Synopsys 设计编译器(DC)对上述 3 个加速器进行仿真与综合;最后使用 CNNGModel 对 3 个加速器的性能与功率进行估计。

4.1　Sconv-DR-OP、Sconv-CR-IP 及 Mconv-CR-MP

如图 7(a)所示,Sconv-DR-OP 的数据传播方式为 OP,寄存器类型为 DR 且数据处理方式为 Sconv。其中的数据传播路径有:(1)ifmaps 的 EXMC→PE;(2)filters 的 EXMC→PE;(3)ofmaps 的 PEAmong;(4)ofmaps 的 EXMC←PE。对于每个 ifmaps 神经元,Sconv-DR-OP 只需要从 EXMC 读取一次,且每个时钟周期只读取一个并送到所有 PE。对于 filters 权重,其会被提前固定在不同的 PE 中。而对于 ofmaps,当其中间结果遍历完所有的 PE 与 FIFO 后,会从最后一个 PE 输出。

如图 7(b)所示,Sconv-CR-IP 的数据传播方式为 IP,寄存器类型为 CR 且数据处理方式为 Sconv。其中的数据传播路径与 Sconv-DR-OP 相比,只有传播路径(3)不同,为 ifmaps 的 PEAmong。在 Sconv-CR-IP 中,首先相同的 1 个 filters 权重会送到所有的 PE。其次对于 ifmaps 神经元,其在每个时钟周期会送到不同的 PE。而对于 ofmaps,每个 PE 一次只负责计算 1 个 ofmaps。

如图 7(c)所示,Mconv-CR-MP 的数据传播方式为 MP,寄存器类型为 CR 且数据处理方式为 Mconv。

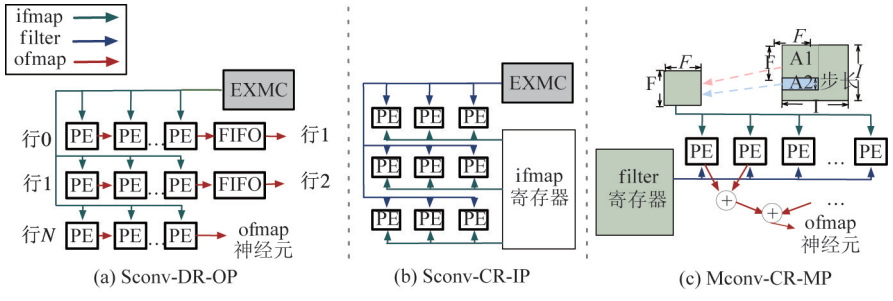


图 7　3 个 CNN 加速器

其中的数据传播路径有:(1) ifmaps 的 EXMC→OCB;(2) ifmaps 的 OCB→PE;(3) ifmaps 的 PEAmong;(4) filters 的 EXMC→PE;(5) filters 的 PEAmong;(6) ofmaps 的 EXMC←PE。对于该加速器,多个 filters 权重和 ifmaps 神经元会被同时送到每个 PE,每个 PE 一次负责计算 1 个 ofmaps。为了保证流水线,其中每个 PE 只负责乘法,剩余的加法操作会在 PE 外执行。

表 8 给出了 3 个加速器的参数。

表 8　3 个加速器的参数

	Sconv-DR-OP	Sconv-CR-IP	Mconv-CR-MP
IOCB. size	\	\	525 kB
IREG. size	\	2.178 kB	0.4 kB
FREG. size	0.2 kB	\	2.178 kB
ARRAY. size	11×11	3×3	3×(11×11)
PECAP. macs	1	1	11×11
NOC. inum	1	3×3	11×11×3
NOC. fnum	11×11	1	11×11×3
NOC. onum	8	8	8
BASICUNIT. isize	$I \times I$	$I \times I$	$I \times I \times 3$
BASICUNIT. fsize	$F \times F$	$F \times F$	$F \times F \times 1 \times 3$
BASICUNIT. osize	$O \times O$	$O \times O$	$O \times O$
BASICUNIT. macs	$F \times F \times O \times O$	$F \times F \times O \times O$	$F \times F \times O \times O \times 3$
BASICUNIT. cycles	$I \times F + O \times O - 2$	$F \times F \times \lceil \frac{O \times O}{9} \rceil$	$O \times O$
BASICUNIT. num	$C \times M$	$C \times M$	$\lceil \frac{C}{3} \rceil \times M$
PSUMS. macs	$F \times F$	$F \times F$	$F \times F \times 3$
PSUMS. pe	1	1	1
EXMC. accessE	\	\	0.006 84 nJ
OCB. accessE	0.006 84 nJ	0.006 84 nJ	0.006 84 nJ
REG. accessE	0.000 0612 nJ	0.000 0612 nJ	0.000 0612 nJ
Frequencu. Hz	1.6 GHz	1 GHz	1 GHz

4.2　实验结果分析

如表 9 所示,选择 AlexNet、VGG 以及 ResNet 中的部分卷积层作为实验数据集。

表 9　用于实验的卷积层参数

	I	*C*	*F*	*M*
AlexNet Conv2(A2)	27	96	5	256
AlexNet Conv4(A4)	13	384	3	384
VGG Conv3(V3)	112	64	3	128
VGG Conv11(V11)	14	512	3	512
ResNet Conv3-2(R3-2)	28	128	3	128
ResNet Conv5-2(R5-2)	7	512	3	512

4.2.1　性能结果对比

对于加速器的性能,本节将重点分析 PE 阵列的工作时间(CAL-cycle)以及加速器工作总时间(Time)。

图 8(a)给出了 Sconv-DR-OP 在处理 6 个卷积层时 PE 阵列的工作时间(CAL-cycle),其中 CNNGModel 的结果与仿真结果的差距分别为 5.2%、5.4%、4.6%、4.0%、4.9% 和 5.4%。可以看到这里的 CNNGModel 结果始终比仿真结果小。这是因为虽然 CNNGModel 考虑了由 NoC 传播以及拥塞所造成的时间,但其使用了加速器的峰值性能。另一方面,CNNGModel 与模拟器 VTA 的差距分别为

0.5%、1.4%、0.7%、0.3%、1.5%、1.2%，说明在不浪费大量人力和时间的情况下，CNNGModel 可以提供与模拟器 VTA 相近的性能。

图 8(b)给出了 Sconv-DR-OP 在处理 6 个卷积层时的总时间(Time)，其中 CNNGModel 的结果与仿真结果的差距分别为 6.7%、7.0%、6.1%、5.4%、5.2% 和 6.4%。造成上述现象的原因有：(1) CNNGModel 对于 CAL-cycle 的估计值小于仿真结果；(2) CNNGModel 没有考虑加速器的配置时间、BasicUnit 间有效信号、状态信号等的传输时间以及数据在 FIFO 中的传播时间。与 CAL-cycle 类似，对于工作总时间 Time，CNNGModel 结果与模拟器 VTA 结果十分接近，差距分别为 1.8%、2.3%、0.9%、1.3%、0.7%、1.7%.

图 9 和图 10 给出了其他 2 个加速器在处理 6 个卷积层时的 CAL-Cycle 以及 Time。与 Sconv-DR-OP 中的结果相似，CNNGModel 的估计值始终比仿真结果小，但与模拟器 VTA 的结果十分接近。

4.2.2　功率结果对比

加速器的能耗主要由传输能耗和计算能耗组成。由于模拟器 VTA 只能给出加速器的工作时间，所以本节将只比较 CNNGModel 与仿真综合的结果。首先将分析加速器工作时对不同存储的访问次数，这些次数可用于得到传输能耗为模型的中间输出。

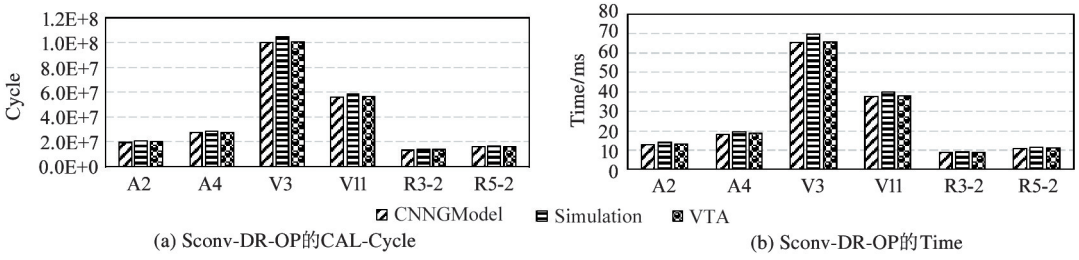


图 8　Sconv-DR-OP 的 CAL-Cycle 以及 Time

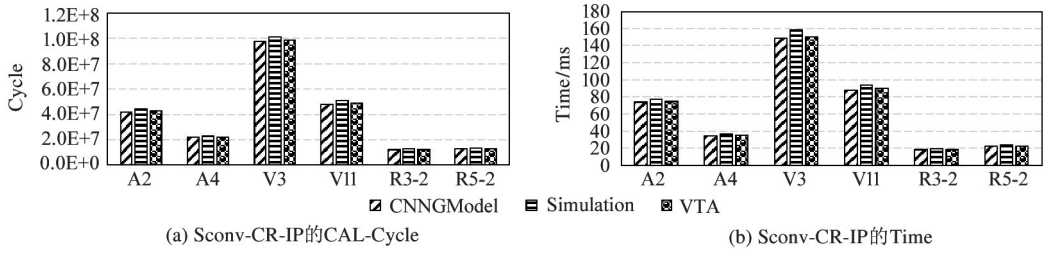


图 9　Sconv-CR-IP 的 CAL-Cycle 以及 Time

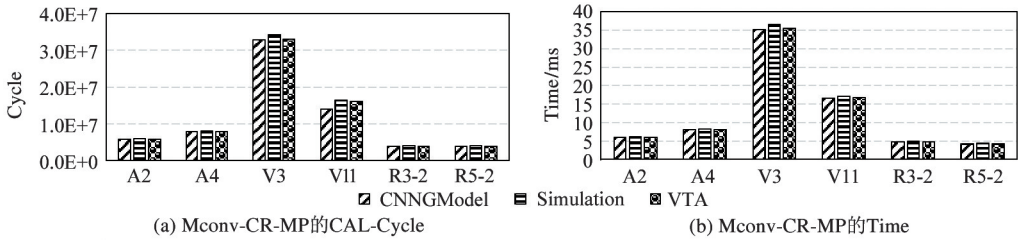
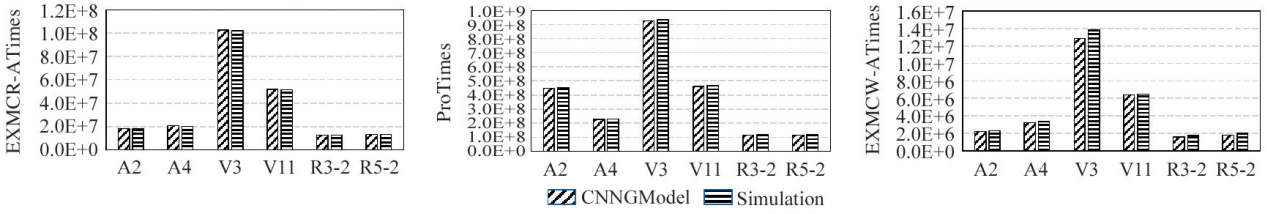


图 10　Mconv-CR-MP 的 CAL-Cycle 以及 Time

图 11(a)给出了 Sconv-DR-OP 在处理 6 个卷积层时对 EXMC 的读次数(EXMCR-ATimes)，其中 CNNGModel 的误差率分别为 0.13%、0.32%、0.56%、0.51%，0.13% 和 0.47%，与仿真结果十分接近。图 11(b)给出了 Sconv-DR-OP 在处理 6 个卷积层时的 PE 间传播次数(ProTimes)，其中 CNNGModel 的误差率分别为 0.57%、1.31%、1.03%、0.37%、2.98% 和 0.54%。可以看到 CNNGModel 比仿真结果稍小，而造成该现象的原因是 CNNGModel 没有考虑在 PE 间传播的一些额外控制信号

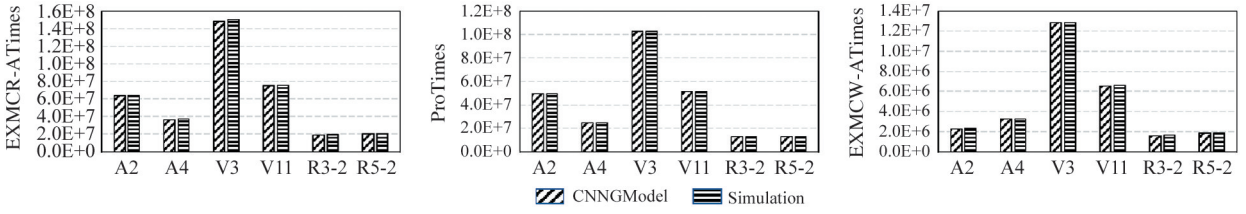
(如当 stride 不为 1 时,会有一些额外的控制信号送到 PE 中)。图 11(c)中给出了 Sconv-DR-OP 在处理 6 个卷积层时对 EXMC 的写次数(EXMCW-ATimes),其中 CNNGModel 的误差率分别为2.84%、4.20%、6.98%、1.25%、7.44%和8.04%。可以看到这里的建模结果比仿真结果稍小,而造成该现象的原因是由于 Sconv-DR-OP 在仿真时写回的位宽是固定的,所以当有效结果的总位宽低于写回位宽时,也会按写回的固定位宽来统计从而增加了额外的写回次数。图 12、图 13 给出了 Sconv-CR-IP、Mconv-CR-MP 在处理 6 个卷积层时各个参数的 CNNGModel 值和仿真值。可以看到这些参数的 CNNGModel 结果和仿真结果基本相同。

图 14 给出了 3 个加速器在处理 6 个卷积层时的功率。可以看到 CNNGModel 的结果与仿真综合结果相近,但始终比其小。造成该现象的原因是 CNNGModel 只考虑了传输、计算以及 NoC 消耗的动态功率,而 RTL 级的综合还考虑了静态功率等。



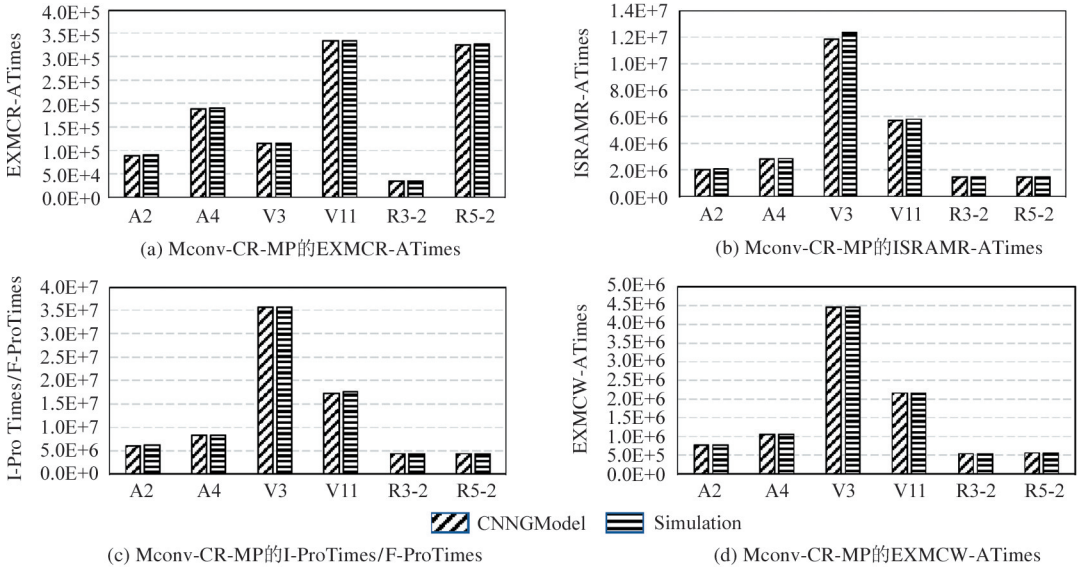
(a) Sconv-DR-OP的EXMCR-ATimes (b) Sconv-DR-OP的ProTimes (c) Sconv-DR-OP的EXMCW-ATimes

图 11 Sconv-DR-OP 的中间输出



(a) Sconv-CR-IP的EXMCR-ATimes (b) Sconv-CR-IP的ProTimes (c) Sconv-CR-IP的EXMCW-ATimes

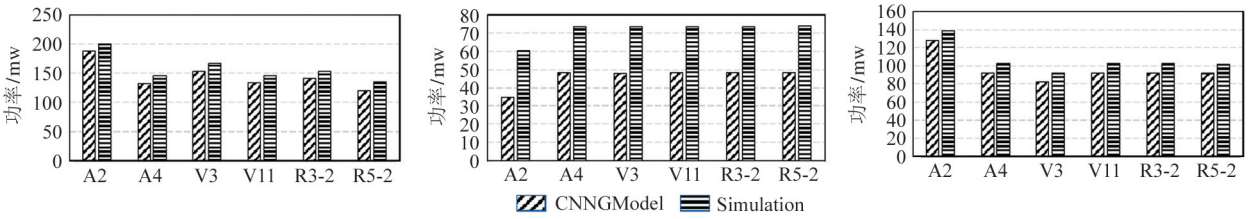
图 12 Sconv-CR-IP 的中间输出



(a) Mconv-CR-MP的EXMCR-ATimes (b) Mconv-CR-MP的ISRAMR-ATimes

(c) Mconv-CR-MP的I-ProTimes/F-ProTimes (d) Mconv-CR-MP的EXMCW-ATimes

图 13 Mconv-CR-MP 的中间输出



(a) Sconv-DR-OP的功率　(b) Sconv-CR-IP的功率　(c) Mconv-CR-MP的功率

图 14　3 个加速器处理每个任务的功率

5　CNN 加速器优化策略

依据 CNNGModel,本节从能耗和性能两方面给出了多项 CNN 加速器优化策略。

5.1　能耗优化策略

在 CNNGModel 中,CNN 加速器传输能耗的计算方式如式(20)所示。其中 $DRAM\_E$、$SRAM\_E$ 以及 $Register\_E$ 代表单次访问每种存储结构时需要的能耗,$ATimes_{\mathrm{DRAM}}$、$ATimes_{\mathrm{SRAM}}$ 以及 $ProTimes_{\mathrm{Register}}$ 代表每种存储结被访问的总次数。

$$E_{\mathrm{Transfer}} = DRAM\_E \times ATimes_{\mathrm{DRAM}} + SRAM\_E \times ATimes_{\mathrm{SRAM}} + Register\_E \times ProTimes_{\mathrm{Register}} \tag{20}$$

文献[4]给出了单次访问 DRAM、SRAM 和 Register 的能耗比为 200∶6∶1。因此为了减少传输能耗,根据式(20),常用的方法为增加 SRAM 以及 Register 大小从而减少对 DRAM 的访问次数。除此之外,根据式(20),通过减少 $ATimes_{\mathrm{DRAM}}$、$ATimes_{\mathrm{SRAM}}$ 以及 $ProTimes_{\mathrm{Register}}$,CNNGModel 还可以从更多的角度对减少传输能耗提出相应的策略。

当 $ATimes_{\mathrm{DRAM}}$ 通过式(3) $ATimes = \dfrac{AVolume}{upperstoragebandwidth}$ 计算时,可以选择增加相应存储的带宽从而减少 ATimes,最终达到减少传输能耗的目的。例如选择使用多块 DRAM 而不是一整块大面积 DRAM,因为多块 DRAM 可以被同时访问,通过增加 channel 提高并行度的方式来增加带宽。

当 $ATimes_{\mathrm{SRAM}}$ 通过式(2) $ATimes = \dfrac{AVolume}{NOC.inum}$ 计算时,此时若想使 ATimes 减半,可以将 PE 阵列每次从 SRAM 读取 ifmaps 的位宽增加 1 倍。例如可以将 PE 阵列中 Register 的个数增加 1 倍,或用 SRAM 代替 Register 并将 SRAM 写位宽按要求设置。当 $ATimes_{\mathrm{SRAM}}$ 通过式(5) $ATimes = \dfrac{BASICUNIT.isize}{PECAP.macs}$ 计算时,此时若想使 ATimes 降低,可以通过增加每个 PE 中计算单元 MAC 的个数来达到需求。

当 $ProTimes_{\mathrm{Register}}$ 通过式(18) $ProTimes = \left\lceil \dfrac{BASICUNIT.macs}{ARRAY.size \times PECAP.macs} \right\rceil \times \left\lceil \dfrac{ARRAY.size \times PECAP.macs}{NOC.inumorNOC.fnum} \right\rceil$ 计算时,此时若想使 ProTimes 降低,可通过增加 $ARRAY.size \times PECAP.macs$ 的值,即增加 PE 阵列大小或每个 PE 中计算单元 MAC 的个数来达到需求。

在 CNNGModel 中,CNN 加速器计算能耗的计算方式如式(21)所示。其中 $ComputeA$ 为当前应用的计算量,$Technology$ 为加速器的工艺。因此根据式(21),可以通过提升加速器的工艺来减少因为计算产生的能耗。

$$E_{\mathrm{compute}} = ComputeA \times Technology \tag{21}$$

5.2　性能优化策略

CNN 加速器的执行时间是由传输时间和计算时间两部分组成的。因为流水线的存在,加速器内部大部分由于数据传输所带来的时间可以被计算时间所覆盖。为了减少 CNN 加速器的总执行时间,需要尽量减少不能被计算时间覆盖的数据传输时间。因此 CNN 加速器在设计时需要尽量将片上的存储结构做双缓存处理,添加 ifmaps 和 filters 数据的预取机制。这样能减少流水线阻断情况的发生,保证每个卷积核的计算过程能够无缝迭代执行。

6 结 论

本文首先基于 CNN 加速器的通用硬件结构,提出了加速器中数据传播路径的概念。其次提出了 CNN 加速器性能与能耗通用评估模型(CNNGModel)。CNNGModel 的输入包括 4 部分:CNN 加速器参数、CNN 加速器数据传播方式、CNN 加速器的数据传播路径以及 NoC 评估模型。输出包括中间输出和最终输出两部分。CNNGModel 的核心部分为通用模型库。

实验结果证明,CNNGModel 的结果与真实的仿真值是十分接近的。最后本文从能耗和性能两方面给出了多项 CNN 加速器优化策略。

参考文献

[1] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[C]// Advances in Neural Information Processing Systems, Lake Tahoe, USA, 2012: 1097-1105

[2] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[EB/OL]. https://arxiv.org/pdf/1409.1556V1.pdf: arXiv, (2014-09-04),[2021-04-15]

[3] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]// IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, USA, 2016: 770-778

[4] CHEN Y H, KRISHNA T, EMER J S, et al. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks[J]. *IEEE Journal of Solid-state Circuits*, 2016, 52(1): 127-138

[5] CHEN Y, LUO T, LIU S, et al. Dadiannao: a machine-learning supercomputer[C]//2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014: 609-622

[6] DU Z, FASTHUBER R, CHEN T, et al. ShiDianNao: shifting vision processing closer to the sensor[C]// Proceedings of the 42nd Annual International Symposium on Computer Architecture, New York, USA, 2015: 92-104

[7] 徐玉. 全球数据中心发展趋势和特点[J]. 电信科学, 2011, 27(12): 62-66

[8] 李其高. 面向 IOT 高能效专用处理器的设计与实现[D]. 重庆:重庆邮电大学通信与信息工程学院, 2018

[9] CAVIGELLI L, BENINI L. Origami: an 803-gop/s/w convolutional network accelerator[J]. *IEEE Transactions on Circuits and Systems for Video Technology*, 2016, 27(11): 2461-2475

[10] FARABET C, MARTINI B, CORDA B, et al. Neuflow: a runtime reconfigurable dataflow processor for vision[C]// IEEE Conference on Computer Vision and Pattern Recognition 2011 Workshops, Colorado Springs, USA, 2011: 109-116

[11] CHOI S B, LEE S S, JANG S J. CNN inference simulator for accurate and efficient accelerator design[C]// 2019 International SoC Design Conference, Jeju, Korea, 2019: 283-284

[12] SAMAJDAR A, ZHU Y, WHATMOUGH P, et al. Scale-sim: Systolic cnn accelerator simulator[EB/OL]. https://arxiv.org/pdf/1811.02883.pdf:arXiv,(2019-02-02),[2021-04-15]

[13] GAO M, PU J, YANG X, et al. Tetris: scalable and efficient neural network acceleration with 3d memory[C]// Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems, Xi'an, China, 2017: 751-764

[14] FARABET C, POULET C, HAN J Y, et al. CNP: an FPGA-based processor for convolutional networks[C]// International Conference on Field Programmable Logic and Applications,Prague, Czech Republic, 2009: 32-37

[15] QADEER W, HAMEED R, SHACHAM O, et al. Convolution engine: balancing efficiency and flexibility in specialized computing[J]. *Communications of the ACM*, 2015, 58(4): 85-93

[16] ZHANG C, LI P, SUN G, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks[C]//Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, New York, USA, 2015: 161-170

[17] ZHANG M, SHI Y, ZHANG F, et al. Comrance: a rapid method for network-on-chip design space exploration[C]//2016 7th International Green and Sustainable Computing Conference, Hangzhou, China, 2016: 1-8

[18] PARASHAR A, RHU M, MUKKARA A, et al. SCNN: an accelerator for compressed-sparse convolutional neural

networks [J]. *ACM SIGARCH Computer Architecture News*, 2017, 45(2): 27-40

[19] 朱晓静. 一种递归定义的可扩展片上网络拓扑结构[J]. 计算机学报, 2011, 34(5):924-930

[20] MOREAU T, CHEN T, JIANG Z, et al. VTA: an open hardware-software stack for deep learning[EB/OL]. https://deepai.org/publication/vta-an-open-hardware-software-stack-for-deep-learning:DeepAI,[2021-04-05]

The research of modeling and optimization for CNN accelerators

QI Yuqiong[*,**], ZHANG Mingzhe[*], WU Haibin[*], YE Xiaochun[*]

([*]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

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

Abstract

A general performance and energy consumption model for convolutional neural networks (CNN) accelerators (CNNGModel) is proposed. According to the design of different structures in a CNN accelerator, CNNGModel can estimate the time and energy consumption of an accelerator when it processes different tasks. CNNGModel can determine whether the current CNN accelerator design meets application requirements before this accelerator is implemented by hardware engineers, thus reducing unnecessary time and manpower. In the experiment, first, three typical CNN accelerators are designed and implemented. Secondly, multiple results of each accelerator dealing with different CNNs are analyzed and compared, and these results are obtained through three methods: CNNGModel, simulator VTA, and simulation and synthesis. For processing time, the difference between CNNGModel and simulation is as low as 3.0%, and for power, the gap is only 6.5%. Finally, based on CNNGModel, some CNN accelerator optimization strategies are given in terms of energy consumption and performance.

Key words: convolutional neural network (CNN), CNN accelerator, general performance and energy consumption model for CNN accelerator (CNNGModel), CNN accelerator optimization strategy