

引用格式:刘偲旻,蒋剑飞,毛志刚.一种指令集控制的神经网络加速器设计[J].微电子学与计算机,2021,38(5):1-6. [LIU S Y, JIANG J F, MAO Z G. A programmable accelerator for convolution neural network[J]. Microelectronics & Computer, 2021,38(5):1-6.]

一种指令集控制的神经网络加速器设计

刘偲旻,蒋剑飞,毛志刚

(上海交通大学 电子信息与电气工程学院,上海,200240)

摘 要:随着深度学习和神经网络技术的发展,为了充分挖掘卷积神经网络(CNN)计算的并行性,硬件加速器以其高速、低成本、高容错能力等特点得到更加广泛的应用.本文提出了一种可以逐层优化 CNN 网络的新算法,设计了对应的指令集.所提出的算法可用于为具有特定计算资源和存储资源的不同网络找到最佳加速方案.在优化过程中,可以将不同类型的数据量化为半精度以减少内存访问.基于 40 nm CMOS 工艺和提出的算法,完成了一种指令集控制的神经网络加速器设计.该加速器在 200 MHz 的工作频率下,峰值性能可达到 416 GOP/s.在设计的加速器上实现了 VGG16 网络的推理过程,整个网络的延迟仅为 116 毫秒.

关键词:加速器;指令集;动态精度量化;卷积神经网络

中图分类号: TN722.1

文献标识码: A

文章编号: 1000—7180(2021)05—0001—06

A programmable accelerator for convolution neural network

LIU Si-yang, JIANG Jian-fei, MAO Zhi-gang

(School of Electronic Information and Electrical Engineering,
Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: In order to fully explore the parallelism of convolutional neural network (CNN) computing, hardware accelerators are more attractive for their characteristics of high speed, low cost and high fault tolerance. A novel algorithm that can optimize the CNN network layer by layer is proposed, and the corresponding instruction set is designed in this paper. The proposed algorithm can be used to find an optimal acceleration scheme for different networks with specific computing and storage resources. In the optimization process, different types of data can be quantized to half-precision to reduce memory access. Based on the 40 nm CMOS process and the proposed algorithm, a programmable accelerator for CNN is designed, which can achieve peak performance of 416 GOP/s under 200 MHz working frequency. VGG is implemented on our accelerator as a case study, and the latency of the total network is 116 ms.

Key words: accelerator; instruction set; dynamic precision quantization; CNN

1 引言

卷积神经网络(CNN)在图像分类和识别领域应用非常广泛.但是,由于它们需要大量的计算操作,通用处理器不能很好地挖掘它们计算的并行性.因此,基于 GPU,ASIC 或 FPGA 的 CNN 加速器层出不穷.各种加速器旨在实现高性能神经网络,其中

卷积运算是最关键的部分.

由于卷积可视为多层循环,因此具有巨大的设计空间.不同的加速方案会显著影响功率,延迟和性能.通常,Roofline 模型可用于评估在相应硬件上加速 CNN 可获得的理论最佳性能,但这仅是性能上限的粗略估计^[1];更常见的方案是分析整个 CNN 结构,并在整个 CNN 的所有层上执行相同的优化,

这样可以简化硬件结构,但是某些层的加速效果可能较差^[2-3];CNN 的不同层也可以分别分析和优化^[4].这种加速方案更具针对性,但是难以实现全局优化.本文提出了一种新颖的算法,可以针对具体的网络结构和硬件资源,计算出最优化的加速方案.本文还基于提出的优化方案,设计了一种支持上述多种加速方案的加速器.

2 设计优化算法

本文提出的设计优化算法主要由两个部分组成,一个是循环优化算法,主要应用于卷积层和全连接层;一个是动态精度量化算法,主要用于优化计算和访存开销不能很好隐藏的层,并对指定类型数据进行半精度处理以减小延迟.

2.1 循环优化算法

常见的循环优化算法有循环分块,循环展开和循环顺序调度.

大多数 CNN 加速器通常将 CNN 网络的每层卷积计算划分为固定的循环分块和固定的循环顺序.就像文献[3]中的 AlexNet 的加速器一样,每一层都使用相同的展开因子.然而对于不同的 CNN 层或不同的 CNN 网络,使用相同的展开因子并不是最佳的加速方案,反而会带来一定程度的性能损失.这是因为固定配置无法适应不同结构的网络层.因此,对于 CNN 网络的每一层,采用可调整的分块大小和可调整的循环顺序可以使加速器在不同的网络或者不同的 CNN 层中有不同程度的性能提升.

本算法旨在广大的设计空间中找到对于每层网络逐层定制的最优循环优化方案,并在包括所有层在内的完整网络结构中达到最优性能.

加速器进行加速的限制因素主要是构成脉动阵列的计算资源,SRAM 资源和对应的 CNN 网络结构.因此我们的算法可以分为两个步骤.第一步是在存储输入,权重,偏置和输出数据的总片上 SRAM 容量固定不变的情况下,为单个卷积层或全连接层计算最佳循环顺序和循环分块大小.第二步是通过遍历确定最优的 SRAM 资源分配方案,即在延迟最小时实现最佳的 SRAM 资源利用率.

首先,计算特定循环顺序和循环分块下的每层网络延迟.先确定一种 SRAM 资源分配方案,其他方案在后续计算中进行遍历.片上的 SRAM 资源采用分布式 SRAM 的结构来满足片上带宽的需求,同时增加利用率.这一步是为了完成片上 SRAM 资源

的划分,如图 1 所示.

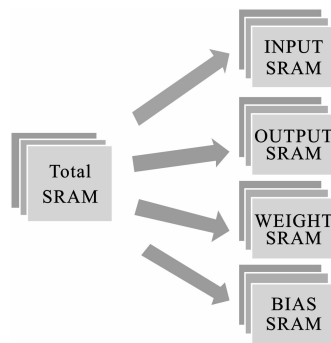


图 1 分布式 SRAM

SRAM 资源分配方案的约束如下,其中 $data_types$ 表示输入,权重,偏差和输出这四种数据类型.

$$\sum_i^{data_types} SRAM_i \leq Total_SRAM \quad (1)$$

$$total_cycles = \max(0, memory_cycles - compute_cycles + compute_cycles + latency) \quad (2)$$

$$resource_util = \sum_i^{data_types} \frac{tile_i}{sram_i} \quad (3)$$

单个卷积层的延迟是内存访问,计算和其他延迟的总和.如果无法隐藏计算和内存访问延迟,那么这两部分将带来大量的延迟代价,因此本加速器采用乒乓缓冲的结构来隐藏部分延迟.

于是可以通过等式(2)来计算延迟开销.可以从(2)中第一项判断隐藏的计算和内存访问延迟情况.此项值越小,隐藏效果越好.如果该值较大,则应分析 $memory_types$ 的具体组成部分以确定导致这种情况的数据类型,然后等待 2.2 节中的进一步优化.

通过以上计算,可以计算得到在某种循环顺序和循环分块方案下的卷积层或全连接层的整体延迟.我们通过遍历多种循环顺序和循环分块方案找寻延迟最小的情况.而另一项重要指标 SRAM 资源利用率计算如公式(3).

但是,这一步类似于贪婪算法,可能在不考虑整体的情况下获得局部最优解.因此,第二步是必要的.由于不同的 SRAM 资源分配方案会导致不同的循环优化方案,从而影响最终加速性能.因此只有再次遍历不同的 SRAM 资源分配条件,才能获得最优的整体加速方案.为了最小化完整 CNN 网络的整体延迟,通过遍历多个 SRAM 资源分配方案,得到的每一层的延迟总和即整体延迟. SRAM 总资源利用率可计算为循环优化算法中的 y_2 .

Algorithm 1 Optimization Algorithm

Inputs:

- 1: Total SRAM resource
- 2: CNN structure
- 3: Systolic array size

Algorithm:

- 1: **for** $k = 1$ to m **do**
- 2: Choose collection R_k
- 3: **for** $i = 1$ to n **do**
- 4: Find $x_i \in Z_1 \cap R_k$ to minimize y_{1i}
- 5: Find $x_i \in Z_2 \cap Z_1 \cap R_k$ to maximize y_{2i}
- 6: **end for**
- 7: $X_k = \{x_i\}$
- 8: Total cycles: $y_1 = \sum y_{1i}$
- 9: Total resource utilization: $y_2 = \sum y_{2i} \times \frac{y_{1i}}{y_1}$
- 10: **end for**
- 11: Find $X \subseteq \{X_k\}$ which can minimize y_1 and maximize y_2

Outputs:

- 1: X : Best acceleration scheme
- 2: y_1 : Minimum latency for total network
- 3: y_2 : Maximum SRAM resource utilization

本文提出的算法综合考虑了计算资源, SRAM 资源, 存储器访问开销, 计算开销, 乒乓缓存, 数据重用等因素, 可以有效地平衡带宽和性能, 最大程度地隐藏延迟, 并充分利用有限的硬件资源.

此算法的整体过程如循环优化算法所示. R_k 表示如图 1 所示的 SRAM 资源分配方案 k 下的满足等式 (1) 的加速方案的集合. x_i 表示其中对于第 i 层的一种加速方案. y_1 表示整个 CNN 网络的延迟, y_2 表示整体的 SRAM 资源利用率. 算法输出所有 y_1 最小的加速方案中 y_2 最大的那一种方案, 即算法得到的最优方案.

通过对加速器在 100 MHz 工作频率下进行功能仿真, 可以测出加速器在推理 VGG16 网络时每一层的延迟. 将仿真结果与根据我们的算法计算出的延迟进行比较, 结果如表 1 所示. 可以得到, 对于前 13 个卷积层, 最大误差不超过 1.3%, 平均误差仅为 0.7%.

2.2 动态精度量化

尽管通过循环优化算法计算出的延迟对于卷积层足够准确, 但对于全连接层, 该误差约为 10~20%.

不同的循环顺序会导致不同的数据复用模式, 如输入复用, 权重复用, 输出复用等, 这导致在下一个分块计算期间不需要从 DRAM 访问这些数据. 因此, 复用的数据类型不能从当前层的乒乓缓存结构中受益. 在我们的算法中, 每个周期都是理想的情

表 1 100 MHz 下加速器仿真与算法计算延迟误差

VGG16 层	仿真延迟 /ms	算法计算延迟 /ms	误差 /%
CONV1	10.3	10.2	0.97
CONV2	20.59	20.6	0.05
CONV3	10.3	10.3	0
CONV4	20.6	20.53	0.34
CONV5	11.84	11.9	0.25
CONV6	23.65	23.53	0.51
CONV7	23.65	23.53	0.51
CONV8	9.9	9.95	0.51
CONV9	19.76	19.95	0.96
CONV10	19.76	19.95	0.96
CONV11	5.34	5.41	1.31
CONV12	5.34	5.41	1.31
CONV13	5.34	5.41	1.31
FC14	64.3	72.65	12.99
FC15	10.49	12.82	22.21
FC16	2.656	2.93	10.32

况. 但是, 对于 DRAM 访问, 还有刷新, 行激活和其他操作引起的其他延迟, 这些延迟在 2.1 节提出的算法无法细致地计算到. 以 VGG16 网络为例, 卷积层频繁的的内存访问和大量的计算彼此隐藏, 可以将这些影响减小到很小的程度. 但是对于全连接层, 权重的访存时间占据了整层延迟的绝大部分, 此时乒乓缓存的结构对于隐藏延迟收效甚微, 因为访存延迟与计算延迟大小差异巨大, 而 DRAM 的多种延迟的影响将是显而易见的, 必须予以考虑.

对于 VGG16 的全连接层, 权重访存占据了式 (2) 中 `memory_types` 的绝大部分, 是计算和访存延迟无法充分隐藏的限制因素. 如果将权重访存配置为半精度模式, 则加速器可以在同一时间段内访问两倍数量的权重. 在 VGG16 的加速中, 发现权重的半精度模式可以将对应层 (FC14 和 FC15) 的整体延迟减少近一半. 而通过将全精度和半精度处理后的数据与 CPU 计算后的浮点精确结果进行对比, 两者有几乎同样的误差, 这表明这些层的权重半精度处理不会带来太多的精度损失.

由于用于输入/权重/输出/偏置的脉动阵列的控制逻辑几乎相同, 因此我们可以进一步扩展动态精度的量化策略. 对于卷积层或全连接层, 式 (2) 的第一部分可用于判断是否可以很好地隐藏计算延迟和内存访问延迟. 如果不能, 应该进一步分析式 (2) 中 `memory_types` 的组成部分, 得出导致这种现象

的数据类型. 对于这种类型的数据, 可以将其单独配置为半精度模式, 而不是像其他加速器设计对所有类型的数据采用相同的量化长度^[5]. 本设计比当前的量化方法更加灵活, 因为可以通过提出的算法, 根据每一层的访存与计算隐藏情况进行分析判断, 分别选择是否对每层的这四种数据进行半精度处理.

3 加速器硬件设计

3.1 整体结构

本文提出的加速器如图 2 所示, 它由脉动阵列, 指令存储器, 译码器, 控制器, SRAM, 数据路由器和 PU 计算单元组成.

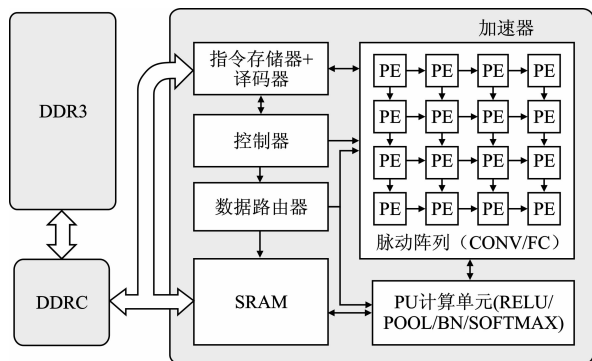


图 2 加速器整体结构图

卷积层和全连接层的计算是在脉动阵列中进行的, 而 RELU, POOL, BATCH NORMALIZATION 和 SOFTMAX 层的计算是在 PU 计算单元中进行的.

3.2 加速器各模块工作情况

3.2.1 指令存储器, 译码器与控制器

加速器收到主控单元传来的启动信号后, 指令存储器发出请求, 从 DRAM 指定地址加载相应的指令. 当最后一条指令从 DRAM 加载到片上后, 译码器开始逐层翻译相应的指令, 并根据这些指令配置数据路由器, 脉动阵列和 PU 计算单元的工作模式. 每层的指令译码和配置完成后, 控制器控制各模块开始工作, 并对各模块的工作情况进行总控.

3.2.2 数据路由器

卷积可以看作多层循环, 对于加速器而言, 可以把卷积拆分为外层的访存循环和内层的计算循环, 如图 3 所示. 首先将算法求得的最优循环顺序和循环分块方案预先以指令的形式存储, 而加速器执行推理过程的具体的循环控制由加速器的数据路由器完成. 数据路由器通过控制循环的次数和每层循环的步长对从 DRAM 中取得的数据, 按指令标示的

循环顺序和分块进行重整后再存入片上 SRAM, 同时也可以按相似的操作控制内层计算的顺序. 总的来说, 数据路由器作为加速器实现循环优化的核心模块, 同时控制着片外存储器访存顺序和片上计算顺序, 即卷积循环的片外和片上部分. 它还控制着脉动阵列和计算单元的工作模式.

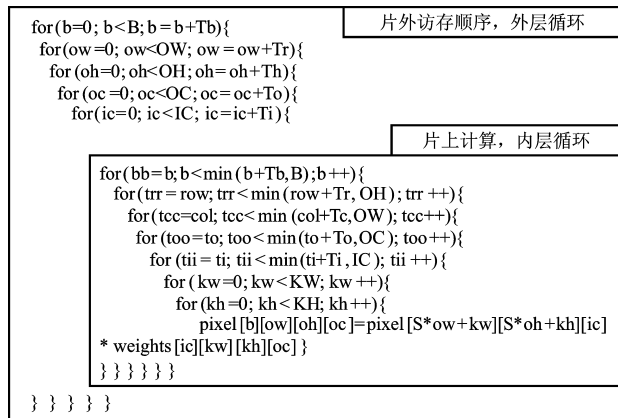


图 3 卷积循环的划分

3.2.3 脉动阵列

脉动阵列进行着卷积层和全连接层的计算. 计算方式如图 4 所示. 为了便于进行多种网络的计算, 我们对 input channel 和 output channel 两个维度进行循环展开. 图 4 中每列对应不同的 input channel, 每行对应不同的 output channel. input 在不同行之间进行数据转发, 即输入数据复用; 部分和在同一行之间从左到右传递进行累加, 即输出数据复用.

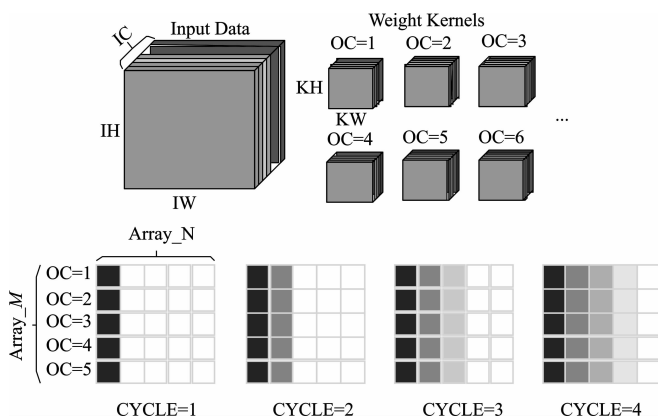


图 4 脉动阵列计算方式

每一个 PE 在 32 个 cycle 内进行 32 个 input channel 的计算, 并累加. 此时 PE 之间没有依赖性, 可以展开实现并行计算, 也更利于用指令集进行循环顺序调度的控制, 适应于更多网络加速的场景.

3.2.4 PU 计算单元

PU 计算单元内执行 POOL, RELU, SOFT-

MAX, BN (BATCH NORMALIZATION)等层的计算. 通过指令控制移位,比较大小,取数,乘法等操作,完成各层所需的计算.

传统的逐层计算会浪费不必要的访存开销. 为了减少不必要的访存,本文采用层间融合的形式,将 CONV/FC 层与下一个 CONV/FC 层之间的 POOL/RELU/SOFTMAX 层进行层间合并. 即将 CONV/FC 层与 POOL/RELU /SOFTMAX 层融合为一个高层,高层的完整计算完成才将结果存回 DRAM.

3.2.5 SRAM

SRAM 采用分布式 SRAM 的结构,分成对应不同数据类型的四大块,分别存储 input, weight, bias,output 四种类型的数据. 每大块的 SRAM 都采用乒乓缓存的结构:当一块 SRAM 通过总线向 DRAM 进行访问存取时,另一块和脉动阵列进行数据交互完成片上计算. 通过这种结构可以有效的隐藏延迟,因为片上的计算和片外的访存同时执行,在总延迟中只需要计算一次.

3.3 指令集

本文设计了指令集对加速器的各个加速场景进行控制,指令集主要如表 2 所示. 每个指令长度为 32 bit.

表 2 指令集	
指令类型	具体控制
BASE_ADDR	指定访存请求在 DRAM 中对应的基地址.
LOOP	循环控制. 指定该层循环次数.
STRIDE	循环控制. 指定该层循环的步长.
BLOCK_END	本层指令的结束. 译码器译码到这条可以控制本层网络推理开始.
DONE	整个 CNN 指令的结束. 指令寄存器搬到这一条则指令加载结束.
PU_START	标示接下来的指令控制 PU 单元内控制寄存器的配置.
PU_COMPUTE	控制 RELU, SOFTMAX, POOL 等层需要的取数,比大小等操作.

4 系统实现及仿真分析

本文的设计基于中芯国际(SMIC)40 nm 工艺实现的. 表 3 给出了 1.1 V 条件下,本设计的面积,时序和功耗. 表 4 中将相关参数与其他支持 VGG16 推理的 ASIC 加速器进行了比较. 可以看出,本文设计的加速器在平均算力,能效比,整体时延方面对比其它设计有明显提高.

表 3 性能参数	
Technology	SMIC 40 nm
Area/mm ²	24. 8
Power/mW	564 (1. 1V)
Frequency/MHz	200
Total SRAM/KBytes	1 589
Instruction SRAM/KBytes	8. 384
Bitwidth/bits	16
Peakthroughput/GOP/s	416
Energy-Efficiency(GOPS/W)	473. 82
Average performance/GOP/s	Conv:328. 69 Total:267. 24
Support layers	CONV, POOL,BN, RELU, SOFTMAX, FC
Computation time(ms)	116

Work		文献[6]	文献[7]	Ours
Technology/nm		65	65	40
Frequency/MHz		250	200	200
Area/mm ²		12. 25	3. 5	24. 8
Bitwidth/bits		16	16	16
Peak performance/(GOP/s)		84	76	416
Voltage/V		1	1	1. 1
Ave performance	Conv	21. 4	67. 7	328. 69
/(GOP/s)	Total	NA	NA	267. 24
Power/mW		236	260	564
Energy-Efficiency	Conv	90. 7	260. 4	582. 8
/(GOPS/W)	Total	NA	NA	473. 83
Total latency/ms	Conv	4 309. 5	453. 3	93. 4
	Total	NA	NA	116

5 结束语

本文提出了一种新颖的算法,可以对 CNN 网络进行逐层优化,为每层网络量身定制不同的循环分块,循环顺序调度策略,在网络整体延迟最小的情况下选择 SRAM 资源利用率最高的方案. 算法同时支持使用半精度访存策略来处理不同类型的数据,从而实现高度的灵活性. 本文设计了一种加速器以支持提出的算法得到的设计优化方案. 加速器的工作频率为 200 MHz,峰值性能可以达到 416 GOP / s. 对 VGG16 进行推理的总延迟为 116 ms.

参考文献:

- [1] WILLIAMS S, WATERMAN A, PATTERSON D. Roofline: an insightful visual performance model for floating-point programs and multicore architectures [R]. Berkeley: LBNL, 2009. DOI: 10.1145/1498765.1498785.
- [2] MA Y F, SUDA N, CAO Y, et al. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA[C]//Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications. Lausanne: IEEE, 2016: 1-8. DOI: 10.1109/FPL.2016.7577356.
- [3] ZHANG C, LI P, SUN G Y, et al. Optimizing fpga-based accelerator design for deep convolutional neural networks[C]//Proceedings of 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey: ACM, 2015: 161-170. DOI: 10.1145/2684746.2689060.
- [4] SHARMA H, PARK J, MAHAJAN D, et al. From high-level deep neural models to FPGAs [C]//Proceedings of the 2016 49th IEEE/ACM International Symposium on Microarchitecture. Taipei, China: IEEE, 2016: 1-12. DOI: 10.1109/MICRO.2016.7783720.
- [5] QIU J T, WANG J, YAO S, et al. Going deeper with embedded fpga platform for convolutional neural network[C]//Proceedings of 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey: ACM, 2016: 26-35. DOI: 10.1145/2847263.2847265.
- [6] CHEN Y H, KRISHNA T, EMER J, et al. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks [C]//Proceedings of 2016 IEEE International Solid-state Circuits Conference. San Francisco: IEEE, 2016: 262-263. DOI: 10.1109/ISSCC.2016.7418007.
- [7] ARDAKANI A, CONDO C, AHMADI M, et al. An architecture to accelerate convolution in deep neural networks[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2018, 65(4): 1349-1362. DOI: 10.1109/TCSI.2017.2757036.

作者简介:

刘偲旻 女,(1996-),硕士研究生.研究方向为数字集成电路设计. E-mail: lsyyalta@sjtu.edu.cn.

蒋剑飞 男,(1979-),助理研究员.研究方向为高速数字集成电路设计、SoC设计与系统集成.

毛志刚 男,(1962-),博士生导师.研究方向为先进计算架构及其芯片技术、高速光电互连及三维芯片技术、低功耗/超低功耗芯片技术.