doi:10.11887/j.cn.202401011 http://journal.nudt.edu.cn

多核数字信号处理卷积算法并行优化

许金伟[1,2],王庆林[1,2],李娅琳[1,2],姜晶菲[1,2],高 蕾[1,2],李荣春[1,2*],李东升[1,2]
(1. 国防科技大学 计算机学院,湖南 长沙 410073;
2. 国防科技大学 并行与分布计算全国重点实验室,湖南 长沙 410073)

摘 要:针对国防科技大学自主研发的异构多核数字信号处理(digital signal processing, DSP)芯片的特征以及卷积算法自身特点,提出了一种面向多核DSP架构的高性能多核并行卷积实现方案。针对1×1卷积提出了特征图级多核并行方案;针对卷积核大于1的卷积提出了窗口级多核并行优化设计,同时提出了逐元素向量化计算的核内并行优化实现。实验结果表明,所提并行优化方法实现单核计算效率最高能达到64.95%,在带宽受限情况下,多核并行扩展效率可达到48.36%~88.52%,在典型网络ResNet50上的执行性能与E5-2640 CPU相比,获得了5.39倍性能加速。

关键词:多核DSP;卷积神经网络;卷积算法;并行优化
中图分类号:TP391 **文献标志码**:A **开放科学(资源服务)标识码(OSID)**:
文章编号:1001-2486(2024)01-103-10

听语音 聊科研 与作者互动

Parallel optimization of convolution algorithm on multi-core DSP

XU Jinwei[1,2], *WANG Qinglin*[1,2], *LI Yalin*[1,2], *JIANG Jingfei*[1,2], *GAO Lei*[1,2], *LI Rongchun*[1,2*], *LI Dongsheng*[1,2]
(1. College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China;
2. National Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha 410073, China)

Abstract: According to the characteristics of the heterogeneous multi-core DSP(digital signal processing) chip independently developed by National University of Defense Technology and the characteristics of the convolution algorithm, a high-performance multi-core parallel convolution implementation scheme for multi-core DSP architecture was proposed. A feature graph level multi-core parallel scheme is proposed for 1×1 convolution. For convolutions with kernels larger than 1, a window level multi-core parallel optimization design was proposed, and an element-wise vectorization based intra-core parallel optimization implementation was proposed. The experimental results show that the proposed parallel optimization method can reach a maximum single core computing efficiency of 64.95%. When the bandwidth is limited, the parallel expansion efficiency of multi-core can still reach 48.36% ~ 88.52%. Compared with E5-2640 CPU, the execution performance on the typical network ResNet50 achieves 5.39x performance acceleration.

Keywords: multi-core DSP; CNNs; convolutional algorithms; parallel optimization

卷积神经网络已经成为人工智能领域应用最广泛的模型之一,不仅在图像分类[1]、目标识别[2]、视频分析[3]等计算机视觉领域的应用中取得了显著效果,还在自然语言处理[4]、语音识别[5]等其他领域同样取得了突破性进展。随着精确感知和高精度识别任务需求的不断增长,大量智能应用在深度学习算法中均使用层数更深的结构,这需要高算力专用硬件的支持,因此卷积神经网络的加速技术一直是研究的热点[6-7]。典型的卷积神经网络主要由卷积层、全连接层、非线性激活函数层、池化层等构成,其中卷积层和全连接层集中了卷积神经网络的主要计算负载。因此研究卷积神经网络的加速技术变成了卷积层加速优化的难点和关键点。

目前常用的卷积优化方法主要有直接卷积[8]、通用矩阵乘法(general matrix multiplication, GEMM)[9-10]和快速算法[11-12]三种实现方式。通用矩阵乘法实现卷积是将卷积操作转换成矩阵操作,再优化矩阵乘法,从而达到优化卷积计算的目的。整个过程中需要增加img2col[10]操作,同时由于输入数据转换成矩阵时会膨胀,从而存储压力也会增加。快速算法是将卷积操作通过

收稿日期:2022-09-20
基金项目:国家自然科学基金资助项目(61732018)
第一作者:许金伟(1990—),男,河南淮阳人,助理研究员,博士,E-mail:xujinwei13@nudt.edu.cn
***通信作者**:李荣春(1985—),男,安徽无为人,研究员,博士,硕士生导师,E-mail:rongchunli@nudt.edu.cn

winograd[11]和快速傅里叶变换(fast Fourier transform, FFT)[12]等变换方式转换成其他快速实现方法的操作,其降低了计算复杂度,并在此基础上开发了并行性以加速卷积的计算。但快速算法也存在计算精度损失和卷积核规模限制等问题。

直接卷积是通过循环展开的方式开发卷积计算的并行性,优化过程中不会增加存储压力,也不会降低计算精度。最近,已经提出了一些直接卷积方法[13-15]来直接实现卷积层。Georganas等[14]通过动态编译方法在x86架构上引入了直接卷积实现,并达到了接近理论极限的性能。Zhang等[13]提出了一种直接卷积实现,在各种CPU架构上显示出比基于GEMM的卷积算法更好的性能。Wang等[15]在ARMv8多核CPU上提出了一种新的并行逐层直接卷积实现,该实现针对具有批次、通道、高度、宽度(batch, channel, height, width, BCHW)数据布局的逐层卷积。与基于GEMM的卷积相比,直接卷积不会产生额外的内存空间开销,在实现中,矢量化和线程级并行化分别用于矢量单元和多核并行性能的有效地利用,寄存器和缓存块用于提高片上多级存储器中的数据重用。这些实现在Phytium FT-1500A和FT-2000处理器上进行了测试,在性能和可扩展性方面,优于Caffe conv和Mxnet conv实现。

FT-M7032是国防科技大学面向E级计算自主研发的一款通用多核数字信号处理(digital signal processing, DSP)芯片[16],由32个DSP核和16个ARMv8 CPU核构成。该芯片的DSP核支持FP32和FP16操作,工作主频为1.6 GHz时,FP16峰值浮点性能可达19.7 TFLOPS,而半精度浮点足以支持人工智能领域中大多数应用的推理计算精度,由此可见,FT-M7032芯片在人工智能领域存在着巨大的应用潜力。在FT-M7032中,采用基于超长指令字的顺序执行架构,并采用软件控制的存储作为片上缓存,提供丰富的直接存储访问(direct memory access, DMA)模式,DMA模式实现各级存储空间的快速访问。现有的面向GPU、CPU等通用处理平台和张量处理单元(tensor processing unit, TPU)等专用处理平台的卷积优化算法在FT-M7032上无法直接运行或者执行效率很低。因此,面向多核向量处理器体系结构进行针对性算法优化有助于发挥FT-M7032芯片的计算性能。因此,本文基于FT-M7032芯片,聚焦直接卷积实现方式,通过不同维度的并行度开发,实现卷积计算在向量处理器上的高效执行。

面向FT-M7032芯片处理卷积神经网络的应用需求,本文结合FT-M7032芯片的体系结构设计,针对目前卷积神经网络中的卷积计算,仔细分析了不同卷积核大小的卷积计算的执行特点和并行粒度,从而针对1×1卷积提出了按行计算的优化方法,针对卷积核大于1的卷积提出了逐元素计算的向量化优化方法,同时针对深层卷积神经网络中,不同层输入数据和权值矩阵规模的变化,提出了不同的多核并行处理方案,从而提升输入数据和权值矩阵的复用度,降低带宽压力。之后,又采用DMA乒乓缓冲机制,实现数据搬移和计算的遮掩。为了验证本文所提方案的有效性,将所提方法在ResNet50卷积神经网络[17]上进行验证。

1 背景知识

1.1 卷积定义

卷积涉及3个张量:输入特征图($\boldsymbol{I}$)、滤波器($\boldsymbol{F}$)和输出特征图($\boldsymbol{O}$)。在C代码样式中,这些具有$BCHW$布局的张量可以表示为$\boldsymbol{I}[B][Ci][Hi][Wi]$,$\boldsymbol{F}[Co][Ci][Hf][Wf]$和$\boldsymbol{O}[B][Co][Ho][Wo]$。卷积的定义如下:

$$\boldsymbol{O}_{(b,co,ho,wo)} = \sum_{ci=0}^{Ci-1}\sum_{hf=0}^{Hf-1}\sum_{wf=0}^{Wf-1}(\boldsymbol{I}_{(b,ci,ho\times s+hf,wo\times s+wf)} \times \boldsymbol{F}_{(co,ci,hf,wf)}) \tag{1}$$

式中,$0\leqslant b<B$,$0\leqslant co<Co$,$0\leqslant ho<Ho$,$0\leqslant wo<Wo$。因此,卷积层的并行度可以分为3个级别,分别是特征图级并行、窗口级并行和操作级并行。其中:特征图级并行是指不同输入通道和输出通道的并行;窗口级并行是指卷积核在一幅特征图上滑动时,不同位置之间的并行;操作级并行是指一个卷积窗口内部,不同元素之间的并行。而在不同卷积层之间还存在层级并行,不同输入输出之间还存在任务级并行。本文在面向FT-M7032芯片实现卷积优化时,会设计各层级的并行度开发,从而提升卷积操作执行性能。

1.2 FT-M7032异构处理器

FT-M7032异构处理器由1个16核ARMv8 CPU和4个通用数字信号处理器(general purpose digital signal processor, GPDSP)集群组成,如图1所示。多核CPU是Phytium FT-2000plus处理器的简化版本,主要负责线程管理和通信。多核CPU的单精度浮点峰值性能为281.6 GFLOPS。每个GPDSP集群包括8个DSP核,它们共享6 MB片上全局共享内存(global shared memory, GSM)。每个集群中的所有8个DSP核和GSM都可以通过片上纵横网络进行通信,通信带宽达到

了 300 Gbit/s。软件开发人员需要维护它们之间的数据一致性。多核 CPU 和 4 个 GPDSP 集群共享相同的主内存空间,多核 CPU 可以访问整个主空间,但每个 GPDSP 集群只能使用 42.6 Gbit/s 的硬件带宽访问自己的相应部分。由于 CPU 内核之间的缓存一致性如 FT-2000plus 所提供的那样,CPU 的缓存数据必须在每个 GPDSP 集群上运行的函数启动之前写入主内存,并在其完成后退出。

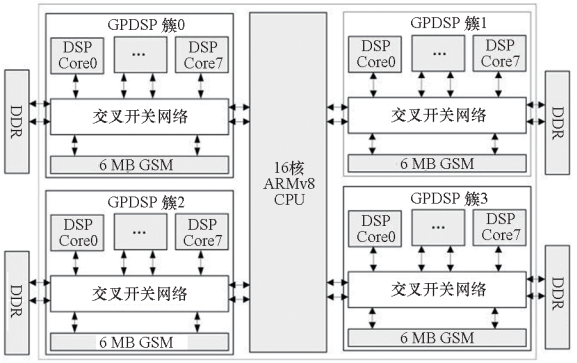

图 1　FT-M7032 的整体架构

Fig. 1　Architecture of FT-M7032

每个 GPDSP 簇中的每个 DSP 核基于超长指令字架构(very long instruction word, VLIW),包括指令调度单元(instruction fetch unit, IFU)、标量处理单元(scalar processing units, SPU)、向量处理单元(vector processing units, VPU)和 DMA 引擎,如图 2 所示。IFU 被设计为每个周期最多启动 11 条指令,其中包含 5 条标量指令和 6 条矢量指令。SPU 用于指令流控制和标量计算,主要由标量处理元件(scalar processing element, SPE)和 64 KB 标量内存(scalar memory, SM)组成,它们匹配 5 条标量指令。VPU 为每个 DSP 核提供主要计算性能,包括 768 KB 阵列存储器(array memory, AM)和以单指令多数据流(single instruction multiple data,SIMD)方式工作的 16 个矢量处理元件(vector processing element, VPE)。每个 VPE 有 64 个 64 位寄存器和 3 个浮点乘累加(float multiply accumulate, FMAC)单元,1 个 FMAC 单元每周期可以处理 2 个 FP32 或者 4 个 FP16 的乘加计算。其中,FP32 数据类型的 SIMD 宽度为 32,FP16 数据类型的 SIMD 宽度为 16,当工作在 1.8 GHz 时,每个 DSP 核可以提供的单精度浮点峰值性能为 345.6 GFLOPS,半精度浮点峰值性能为 691.2 GFLOPS。AM 可以通过两个加载存储向量单元,在每个周期向寄存器传送多达 512 B 数据。在 SPU 和 VPU 之间,可以通过广播指令和共享寄存器传输数据。DMA 引擎用于在不同级别的存储器(即主存储器、GSM 和 SM/AM)间传输数据。AM 每个周期可以向向量寄存器提供 512 B 数据,即 AM 与向量寄存器之间的带宽为 921.6 GB/s。

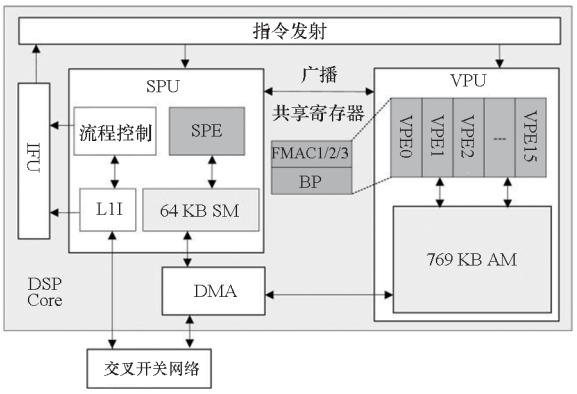

图 2　FT-M7032 中的 DSP 单核微架构

Fig. 2　Macro-architecture of each DSP core in FT-M7032

2　面向 FT-M7032 的卷积计算多核并行优化

在进行卷积计算时,为了降低带宽压力,提升计算效率,需要尽可能地将可复用的权值或者特征图放置在 GSM 或者 AM 中。为此,将根据深度卷积神经网络中特征图和权值矩阵数据规模随着层数而产生的变化,来规划多核间的并行方式。一般深度卷积神经网络中,在模型的前层,特征图规模大、通道数小,因而输入输出特征图所占用的存储空间大,权值占用的空间相应较少;后层则是特征图规模小、通道数多,因而输入输出特征图所占存储空间的比例会减少,权值矩阵随着通道数的增加而急剧变大,所占存储空间的比例会相应增加。因此,本文根据卷积层的存储分布特点在多核间分别采用层内多核并行和任务多核并行两种不同的优化方法。另外,由于 FT-M7032 芯片中不同簇之间并不共享 GSM,本文所提的多核并行方案仅是在单簇内部,不同簇之间均采用任务级并行方案。

2.1　特征图数据布局

为了后续行文方便,先介绍在面向 FT-M7032 芯片时所采用的数据布局。卷积的输入特征图的数据布局格式为 *NCHW*,其中 N 为 batchsize,C 为输入通道数,H 和 W 为特征图的高和宽,由于在 FT-M7032 的向量处理器中,单 FMAC 一次处理的位宽是 64 位,所以向量寄存器中单个寄存器的位宽也是 64 位,本文在数据布局时同时处理来自

4 幅图的数据,将 4 幅图对应的每个元素拼接成一个 64 位数据。所以输入特征图的数据布局方式为 N_4CHW4,其中 4(非下标)表示最内部是来自 4 幅图的 4 个元素,N_4 表示 N 除以 4。

2.2 卷积层内多核并行优化

卷积层内多核并行开发的难点在于如何将任务有效地分配到不同的处理核上,且在并行开发过程中,降低对带宽的需求,增加多核之间对于输入数据和权值的复用度。考虑到这一点,在设计时提出了特征图级并行和窗口级并行两种不同的实现方案。

卷积层内多核并行优化是指将卷积按照特征图区域或者输出通道进行划分,其中按照特征图区域划分是针对卷积核大于 1 的卷积层,将输入特征图划分成 M 等份,分给 M 个核同时处理,即多核间采用窗口级并行;输出通道划分是针对卷积核为 1 的卷积层,按照输出通道划分成 M 等份,分给不同核处理,即多核间采用特征图级并行。其中,M 是单簇中 DSP 的核数,在 FT-M7032 中 M 最大为 8。

2.2.1 特征图级多核并行

在卷积核为 1×1(Conv1 $\times$1)时,输入特征图可以看作列为 $Hi\times Wi\times4$、行为 Ci 的矩阵,记作输入数据:$[Ci][Hi\times Wi\times4]$,其中 Ci 表示输入通道,Hi 和 Wi 表示输入特征图的高和宽。权值数据可看作列为 Ci、行为 Co 的矩阵,记作权值数据:$[Co][Ci]$,其中 Co 表示输出通道。输出特征图可看作列为 $Ho\times Wo\times4$、行为 Co 的矩阵,记作输出数据:$[Co][Ho\times Wo\times4]$,其中 Ho 和 Wo 表示输出特征图的高和宽。因此可以将 1×1 的卷积操作看成一个矩阵乘操作。

Conv1 $\times$1 多核并行优化方案是在权值数据的输出通道 Co 上作切分,本文将 Co 切分为 M 等份,令 $m=\lceil Co/M\rceil$,将数据依次按核分配后传输到对应核的 SM 上,而各核中所有输入数据都需要参与计算,故本文将通过 DMA 通道传输完整的输入数据到每个核的 AM 上,各核计算后将得到 m 个对应输出通道的输出结果,图 3 为 Core0 的具体实现,实际运行中各核是同步进行的。

核内外的数据传输均需通过 DMA 通道,因为 GSM 到核内的带宽要远高于双倍数据速率(double data rate, DDR)同步动态随机存储器到核内的带宽,因此,可以将大部分输入输出数据暂存在 GSM 空间中,GSM 空间不足时,再存入 DDR 中,此举能最大限度减少输入输出的带宽压力。计算中,各核所需要的权值数据将以点对点的方式传输到对应的 SM 中;输入数据在 GSM/DDR 中连续存储,每次将其中的一个计算分块从 GSM/DDR 中以广播的方式传输到核内的 AM 上,各核用标量加载指令读取 SM 上的权值数据分块,再通过标量的广播指令传输到向量存储器中(这种方法能够显著降低对 DDR 的数据带宽需求),同时用向量加载指令读取 AM 上的输入数据分块,再对两者进行计算,所有核都计算结束后,将输出数据用向量存储指令存回 AM,再按分块逻辑存储至 GSM/DDR 中。依次加载并计算下一个分块,直到完成所有计算和数据存储。

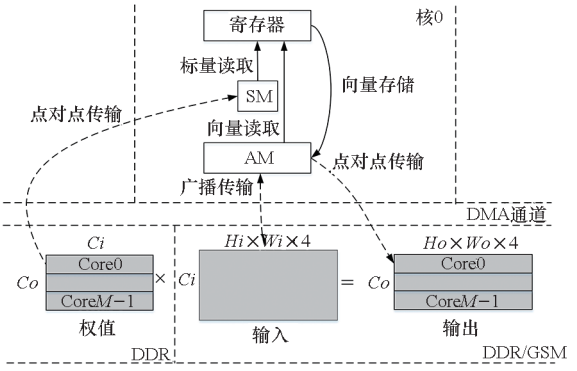

图 3 特征图级多核并行优化

Fig. 3 Feature map level multi-core parallel optimization

2.2.2 窗口级多核并行

在卷积核大于 1×1 时,采用窗口级多核并行优化设计,将输入特征图按区域划分为 M 份,分配到各个核上,每个核上都需要访问完整的权值数据,所以需要将权值数据存放在 AM 中实现向量化,将各个核分配好的输入数据存放在 SM 中,按照元素计算的方式进行向量化卷积。

同样地,核内外的数据传输需要通过 DMA 通道。首先将输入数据按多核机制分配,将其按基础分块大小从 GSM/DDR 中以点对点的方式传输到对应核的 SM 上,然后从 DDR 中将权值数据按分块大小广播到各核的 AM 中,分别用标量指令和向量指令将数据加载到标量和向量寄存器中,按逐元素计算的方法完成卷积,等到所有核的卷积计算完成,再将数据依次存储到 AM 中和 DDR/GSM 中,再依次加载并计算下一个分块,直到结束。

2.3 任务级多核并行方案

任务级多核并行优化设计中,核与核之间是独立运行的,每个核将独立处理 4 幅图,因此该方案下,单簇能同时处理 $4\times M$ 幅图的计算,因此需要等待前面的卷积层处理 $4\times M$ 幅图后才会启动该层的处理。

任务级多核并行优化设计中,因为单次处理

的数据量过大,无法暂存在 GSM 中,所有的输入输出均需要传输回 DDR 存储。同样地,本文将不同的图片数据从 DDR 以点对点的方式搬移至多核,而各核之间的权值数据是共享的,所以将权值数据从 DDR 以广播的方式传输至各核,同样是按照分块大小依次传输与计算。

2.4　两级 DMA 双缓冲机制

为减少数据搬移开销提高卷积计算效率,本文提出基于两级 DMA 双缓冲的数据搬移优化策略,分别在 SM 和 AM 中为输入数据或权值数据设计两个缓冲区,Conv1 ×1 计算中,在 SM 中为权值数据建立两个缓冲区,在 AM 中为输入数据建立两个缓冲区;卷积核大于 1 ×1 的卷积计算中,在 SM 中为输入数据建立两个缓冲区,在 AM 中为权值数据建立两个缓冲区。该机制将核心计算和数据搬移同步进行,两者的执行时间重叠,能覆盖耗时较短的那部分操作时长,提高计算性能,具体两级 DMA 双缓冲机制如图 4 所示。

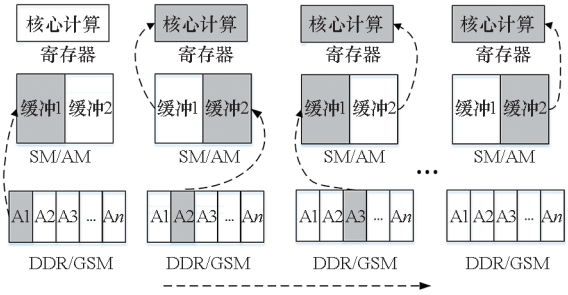

图 4　两级 DMA 双缓冲机制

Fig. 4　Two level DMA double buffer mechanism

将核心计算需要的第一组数据 A1 先搬移至 SM/AM 的缓冲区 buffer1 中,同时开启 A1 核心计算与下一组数据块 A2 到 buffer2 的搬移,等到 A1 的核心计算和 A2 的数据搬移都结束后,再开始进入下一轮 A2 的核心计算和 A3 的数据搬移,如此计算和数据通信同时进行,直至 DDR/GSM 的数据遍历结束。

3　面向单核 DSP 元素级并行优化

FT-M7032 芯片中 DSP 核上的存储空间是有限的,SM 的存储空间是 64 KB,AM 的存储空间是 768 KB。面对不同规格的卷积,需要对输入数据和权值数据进行分块处理。

由于 DSP 内部的 VPU 包含 16 个 VPE,每个 VPE 包含 3 个 FMAC,本文开发元素级并行就是将卷积计算中每个特征图元素的 48 个输出通道放到 48 个 FMAC 上并行执行,从而有效利用 DSP 内部的计算资源,提升卷积计算在每个 DSP 的执行性能。

令输入数据的子块为 *Ib*,权值数据子块为 *Wb*,输出数据的子块为 *Ob*。

3.1　Conv1 ×1 分块设计与核内计算

3.1.1　Conv1 ×1 分块设计

Conv1 ×1 计算中,权值将存放在 SM 中,输入和输出存放在 AM 中,由于程序设计了 DMA 和计算的乒乓缓冲机制,故 SM 和 AM 空间将划两个缓冲区分别存储子块。令输入的 $Hi \times Wi \times 4$ 维度的分块大小为 *Nb*,结合处理器结构特点,分块遵循以下原则:分块大小不得超出片上存储空间,*Nb* 为 192 的倍数,尽可能在单维度切分且尽可能减少数据的切割次数。

令子块 *Wb* 为 $m \times K$,*Ib* 为 $K \times Nb$,分块思路如下:①结合处理器结构特点,考虑到处理器指令的延迟槽循环和流水线排布,*Wb* 的 *m* 取 6 ~ 12 较为合适;②确立好 *m*,判断当 $K = Ci$ 时权值的子块是否满足 SM 的存储空间要求,若否,则将 *Ci* 按二分法继续切分直至满足 SM 存储空间要求;③输入数据分块中首先将 *Nb* 设为 192,结合 *Wb* 的取值 *K* 判断 $K \times 192$ 是否符合 AM 存储空间要求,若是,则继续以 192 为倍数扩大 *Nb*,直到取到范围内的最大值,若否,则在 *K* 上按二分法继续切割,直到满足存储要求为止。

3.1.2　Conv1 ×1 核内计算

结合上述分块设计可以得到子块 *Ib* 和 *Wb*,按行实现的 Conv1 ×1 如图 5 所示,具体算法见算法 1。

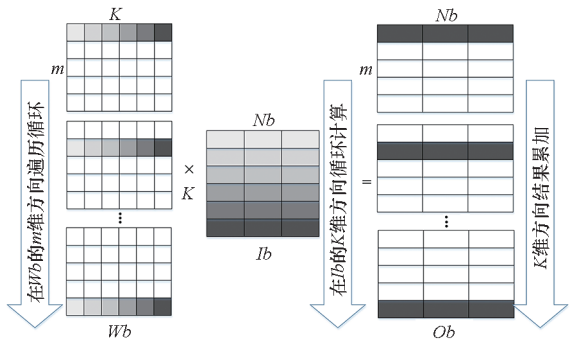

图 5　按行实现的 Conv1 ×1 示意

Fig. 5　Schematic diagram of row-wise Conv1 ×1

3.2　Conv3 ×3 分块设计与核内计算

3.2.1　Conv3 ×3 分块设计

Conv3 ×3 计算中,输入数据将存放在 SM 中,权值和输出数据存放在 AM 中,由于输入和权值的数据量较大,无法全部存入核内,也需对各数据进行分块。

算法1　按行实现的 Conv1×1 算法

Alg. 1　Row-wise Conv1×1 algorithm

输入：$Wb_{m\times K}$，$Ib_{K\times Nb\times 4}$

输出：$Ob_{m\times Nb\times 4}$

BEGIN

开启多核机制，各核同步运行以下程序；

根据 AM、SM 容量设置子块大小为 $Ib_{K\times Nb\times 4}$ 和 $Wb_{m\times K}$，令 K 维方向的索引为 i，m 维方向索引为 j；

将所有向量寄存器数据置零；

FOR $i=0$ TO K **DO**

读取 $Ib_{(i,Nb,4)}$ 上 $Nb\times 4$ 个数据至向量寄存器 VR0；

FOR $j=0$ TO m **DO**

读取 $Wb_{(j,i)}$ 的单个数据至标量寄存器 R0；

R0 数据扩展、广播至向量寄存器 VR1；

做乘加计算 VR2 = VR0 × VR1 + VR2；

在 m 维方向循环计算并将结果依次保存至 m 组不同的向量寄存器中；

END FOR

在 K 维方向循环计算并将结果累加至对应的 m 组向量寄存器中；

END FOR

将输出子块 $Ob_{(m,Nb,4)}$ 搬移至 AM 中；

将 AM 中的 $Ob_{(m,Nb,4)}$ 搬移至 GSM 或 DDR 中；

END

Wb 分块设计：考虑到处理器核内有 16 个 VPE，每个 VPE 有 3 个 FMAC，则处理器单周期可处理 48 个 64 位数据。Conv3×3 的分块设计思路如下：①向量化并行设计需要数据间具备计算的非相关性，*Ci* 通道需要做循环累加，不具备该条件，而特征图尺寸变化跨度大也不适合向量化，故只能在 *Co* 维做向量化设计；②向量的数据加载是连续的，故应将 *Co* 维的数据在格式上连续存储，按大小 48 划分，并将其提取到格式里面；③结合 ResNet50 网络结构特征和 AM 存储空间的大小，将 *Ci* 维按大小 64 划分。故权值子块为 *Wb*：[3][3][64][48][4]（舍入部分用零填充）。

Ib 分块设计：面对不同规格的输入数据，需要结合 SM 空间大小、DMA 双缓冲机制和 Conv3×3 卷积的计算原理综合考虑。已知 *Ib* 的 *Ci* 维分块为 64，而逐元素 Conv3×3 计算每次需要加载 3 行输入数据，则 *padding* = 1 的情况下，可计算出特征图 *Wi* 维度分块的最大值 *Wib*，即输入数据子块为 *Ib*：[64][3][*Wib*][4] 。

3.2.2　Conv3×3 核内计算

逐元素 Conv3×3 计算中，将 *Ib* 和 *Wb* 加载到核内空间后，读取同一 *Ci* 通道的 *Ib* 和 *Wb* 数据，将 *Ib* 感受野区域的 3×3 数据与 *Wb* 的卷积核数据对应相乘，遍历 *Ci* 维将累加计算结果，再将 3×3 的 9 个像素点数据相加，即得到单元素卷积结果，如图 6 所示，具体步骤见算法 2。

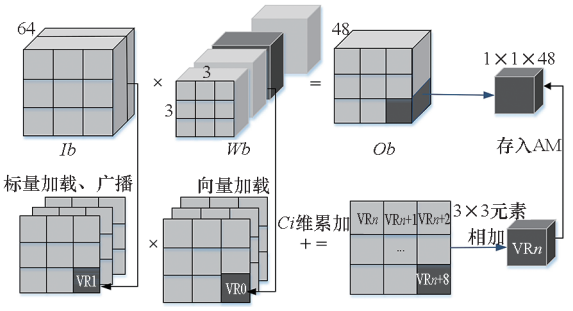

图 6　逐元素 Conv3×3 示意

Fig. 6　Schematic diagram of element-wise Conv3×3

算法2　逐元素 Conv3×3 算法

Alg. 2　Element-wise Conv3×3 algorithm

输入：$Wb_{3\times 3\times 64\times 48\times 4}$，$Ib_{64\times 3\times 3\times 4}$

输出：$Ob_{48\times 1\times 4}$

BEGIN

开启多核机制，各核同步运行以下程序；

根据上述分块设计和 Conv3×3 计算原理，设置子块大小为 $Wb_{3\times 3\times 64\times 48\times 4}$，取 *Ib* 大小为 $Ib_{64\times 3\times 3\times 4}$，令子块中输入通道 *Ci* 方向索引为 i，特征图 *Hi* 维方向索引为 j，特征图 *Wi* 维方向索引为 k；

将所有向量寄存器数据置零；

FOR $i=0$ TO 64 **DO**

FOR $j=0$ TO 3 **DO**

FOR $k=0$ TO 3 **DO**

依次在感受野区域和卷积核上偏移取值；

读取 $Wb_{(j,k,i,48,4)}$ 上 48×4 个数据至向量寄存器 VR0；

读取 $Ib_{(i,j,k,4)}$ 数据至标量寄存器 R0；

R0 数据扩展、广播至向量寄存器 VR1；

做乘加计算 VR2 = VR0 × VR1 + VR2；

在特征图 *Wi* 维、*Hi* 维方向循环计算并将结果依次保存至 3×3 组向量寄存器中；

END FOR

END FOR

在输入通道 *Ci* 维方向循环计算并与暂存的中间结果累加；

END FOR

Ci 遍历结束，3×3 的 9 个像素点结果相加，得到输出子块 $Ob_{48\times 1\times 4}$；

将输出子 $Ob_{48\times 1\times 4}$ 搬移至 AM 中，并按特征图输出格式存储；

END

单元素计算完成后,需要按序依次对输入子块 *Ib* 中的其他元素做卷积,根据上述设计已知子块为 $Wb_{3\times3\times64\times48\times4}$ 和 $Ib_{64\times3\times Wib\times4}$,得到的输出子块为 $Ob_{48\times Wib\times4}$,算法实现见算法 3。

算法 3　Conv3 ×3 子块卷积算法

Alg. 3　Conv3 ×3 sub-block convolution algorithm

输入: $Wb_{3\times3\times64\times48\times4}$, $Ib_{64\times3\times Wib\times4}$

输出: $Ob_{48\times Wib\times4}$

BEGIN

开启多核机制,各核同步运行以下程序;

令子块中 *Wib* 方向的索引为 *s*;

FOR *s* =0 TO *Wib* **DO**

按给定步长偏移,加载子块 $Ib_{(64,3,s,4)}$ 和 $Wb_{3\times3\times64\times48\times4}$ 的数据做算法 2 计算;

END FOR

在 *Wib* 维方向遍历结束后,得到输出子块 $Ob_{Wib\times48\times4}$;

将输出子块 $Ob_{Wib\times48\times4}$ 转置成 $Ob_{48\times Wib\times4}$;

将 AM 中的 $Ob_{48\times Wib\times4}$ 搬移至 GSM 或 DDR 中;

END

4　性能评估

本节将全面评估与分析本文所提出的面向 FT-M7032 异构处理器的卷积并行优化方法的性能。

4.1　实验设置

本节主要涉及所提卷积并行优化方案与 CPU 上卷积实现之间的性能对比,CPU 卷积实现是指基于 PyTorch[18] 框架实现的在 Intel(R) Xeon(R) CPU E5-2640 v3 上的卷积计算。

在本节中涉及四个指标来评判卷积的性能,第一个是完成卷积计算的时间 *T*,第二个是卷积计算所达到的计算性能 P_{conv},第三个是卷积计算在单个 GPDSP 簇上的计算效率 E_{conv}(计算效率是指实际执行时所达到的计算性能与峰值计算性能之间的比率),第四个是卷积计算的扩展效率 S_{conv}(扩展效率是指 *n* 核并行时所达到的计算性能与单核计算性能的 *n* 倍之间的比率)。四个指标之间的相互关系可表示为:

$$P_{\text{conv}}=\frac{2\times N\times Ho\times Wo\times K\times K\times Ci\times Co}{T} \tag{2}$$

$$E_{\text{conv}}=\frac{P_{\text{conv}}}{Peak_{\text{gpdsp}}} \tag{3}$$

$$S_{\text{conv}}=\frac{P_n}{n\times P_{\text{conv}}} \tag{4}$$

其中,$Peak_{\text{gpdsp}}$ 表示单个 GPDSP 簇的 FP16 峰值计算性能,P_n 表示 *n* 个 DSP 核时所能达到的计算性能。

4.2　在不同数据规模下单核性能及计算效率评估分析

为了评估本文所提出的核内计算和分块设计的有效性,本节选取典型的卷积层来评估其在单核上的计算性能和计算效率。

在实验时,首先评估了 Conv1 ×1 的单核性能,从 ResNet50 的 stage2、stage3、stage4 和 stage5 中分别选取了其第二个 block 的第一层卷积,其特征图大小分别为 56、28、14、7,输入通道分别为 256、512、1 024、2 048,输出通道分别为 64、128、256、512。图 7 展示了其各层的执行时间和计算效率。其中 *Tconv*1 和 *Econv*1 分别代表 Conv1 ×1 的执行时间和计算效率,*Tconv*3 和 *Econv*3 分别代表 Conv3 ×3 的执行时间和计算效率,后续图表中的 *Sconv*1 和 *Sconv*3 分别表示 Conv1 ×1 和 Conv3 ×3 的扩展效率。由图 7 可以看出,由于这四层的总计算量是一样的,在本文提出的分块设计和核内计算优化的情况下,前三层的执行时间基本一致,计算效率也保持在 60% 以上,可见本文所提的方法是有效的。而第四层的计算效率之所以偏低,是因为其特征图规模变得过小,在进行分块时需要添加冗余以保证程序的正确性。

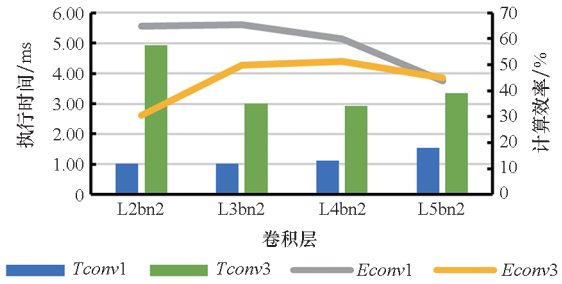

图 7　不同规模卷积时 DSP 单核性能和计算效率

Fig. 7　Performance and calculation efficiency of DSP single core for different scale convolution

其次评估了 Conv3 ×3 的单核性能和执行效率,从 ResNet50 的 stage2、stage3、stage4 和 stage5 中分别选取了其第二个 block 的第二层卷积,其特征图大小分别是 56、28、14、7,输入通道为 64、128、256、512。图 7 展示了其各层的执行时间和计算效率,可以看出,后三层的执行时间相当,计算效率分别为 50.01%、51.37% 和 45.07%,这是因为这四层的计算量是相同的,同时也说明本文的方法对于不同规模的卷积都是适应的。第一层的计算效率之所以会略微偏低,是因为输出通道

相比于分块方法中所提出的基本块输出通道分割基数 48 偏小,需要填充较多冗余。

4.3 在不同核数时计算效率及扩展效率评估分析

在深度学习应用中,由于特征图像和权值数据过大,无法在片上进行存储,当计算峰值和带宽不匹配时,即当带宽不足时,会影响其计算性能和计算效率。为此,特别设计不同的数据访存方式和多核方案,为了验证带宽受限情况下本文方法的有效性,本节选取了不同核数进行了计算效率评估和扩展效率评估,结果如图 8 所示。

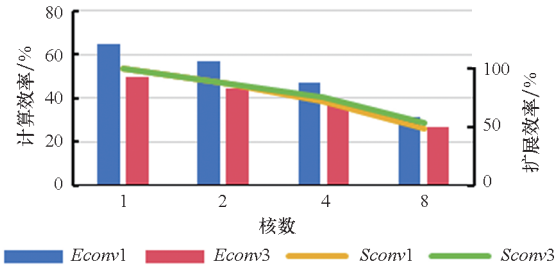

图 8 不同核数时卷积计算效率和扩展效率

Fig. 8 Convolution calculation efficiency and scalability efficiency with different cores

首先,选取和上节相同的典型卷积层进行 Conv1 ×1 的多核测试,分别测试 1、2、4、8 核的计算效率以及扩展效率。实验结果表明:随着核数的增加,本文的计算效率在降低,以第一个卷积为例,其 1 核、2 核、4 核和 8 核的计算效率分别为 64.95%、57.18%、46.78% 和 31.41%,其 2 核、4 核和 8 核的扩展效率为 88.04%、72.02% 和 48.36%。可见,2 核时带宽影响较小,其扩展效率较高,随着核数变多,计算峰值提升,而带宽压力变大,其扩展效率就逐步降低,但 8 核时仍能保持 31.41% 的计算效率。

其次,采用同样的实验方法对 Conv3 ×3 进行了测评。实验结果表明:随着核数的增加,本文的计算效率逐步降低,以第二个卷积为例,其 1 核、2 核、4 核和 8 核的计算效率分别为 50.01%、44.27%、38.21% 和 26.93%,扩展效率分别为 100%、88.52%、76.42% 和 53.86%。Conv3 ×3 的评估结果同样表明:随着核数提升,由于带宽的影响,计算效率会降低,但 8 核的计算峰值仍能保持 26.93% 的计算效率。

4.4 在不同卷积规模下不同平台性能对比分析

为了对比不同平台上的性能,选择E5-2640 CPU 作为对比平台。在实验时,为了保持结果的可靠性,本文设置 batchsize 为 32,在 FT-M7032 上,N_4 则为 8,即单簇执行 32 幅图的性能。图 9 展示了在不同卷积规模下 E5-2640 CPU 和 FT-M7032 平台上的执行时间,其中执行时间是指 32 幅的总计执行时间。从图中可以看出,在 FT-M7032 上的卷积实现时间能够达到 E5-2640 的 1.52 ~ 8.22 倍,这个结果验证了本文方法的有效性。

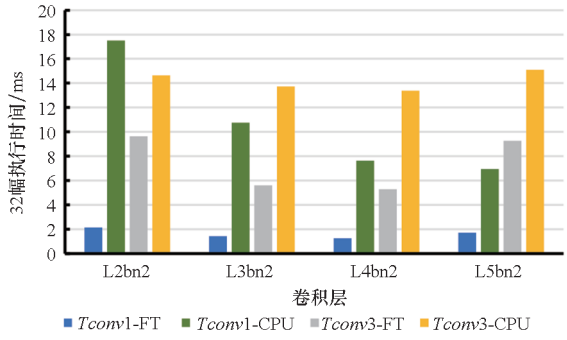

图 9 不同平台上不同卷积规模性能对比

Fig. 9 Performance comparison of different convolution scales on different platforms

4.5 在典型网络上的相关工作对比

4.5.1 典型网络在不同平台上的性能对比

在本小节,将采用典型网络 ResNet50 模型进行测评。在 ResNet50 中,核为 7 × 7 的卷积(Conv7 ×7)采用与 Conv3 ×3 相类似的方式实现,全连接层采用与 Conv1 ×1 相类似的方式实现,池化层同样根据类似的设计理念进行了优化实现,卷积计算后紧跟批量归一化(batch normalization, BN)和 ReLu 操作,包括 Conv1 ×1、Conv3 ×3 和 Conv7 ×7。为了提高网络的效率,将 BN、ReLu 和卷积计算融合在一起,卷积计算之后的输出数据不立即存回 AM 空间,在寄存器中继续做 BN 和 ReLu 计算,ReLu 计算之后将数据传输到 AM 和核外。该处理能减少输入输出数据的通信次数和时间,提高计算性能。面向典型卷积神经网络与通用平台性能对比结果见表 1,基于 FT-M7032 所实现的 ResNet50 模型在 batchsize 为 32 的情况下,相比于 E5-2640 CPU 有 5.39 倍的加速比,其中 E5-2640 CPU 上的执行时间是基于 PyTorch 架构测试获得的。

表 1 面向典型卷积神经网络与通用平台性能对比

Tab. 1 Performance comparison with general platform for typical convolutional neural network

平台	执行时间/ms	加速比	主频/GHz	核数
E5-2640	1 131.27	1.00	2.6	6
FT-M7032	209.73	5.39	1.6	8

4.5.2　扩展效率对比

为了进一步展现本文工作的有效性,本文方法与文献[15]的方法进行了扩展效率的比较,见表 2,文献[15]仅选用 ResNet50 中的 Conv1 ×1 作为测试对象,因此比较时本文同样选取Conv1 ×1作为对比对象。从表 2 数据可知,本文方法的扩展效率略有优势。关于计算性能,文献[15]中,面向 Conv1 ×1,其性能相比于同平台的 Caffe 架构获得 1.16 ~3.79 倍的性能加速,其文中无绝对时间。而本文所测完整 ResNet50 相比于 E5-2650 有 5.39 倍的加速比。因此,由于平台的差异性,无法进行绝对的对比。

表 2　面向典型卷积神经网络与相关工作扩展效率对比

Tab.2　Scalability efficiency comparison with related work for typical convolutional neural network

平台	扩展效率/%	主频/GHz	核数
FT-1500A[15]	48.36	1.5	16
FT-M7032	42.00	1.6	8

5　结论

本文针对 FT-M7032 异构多核 DSP 体系结构和卷积计算的特点,分析并行机理,针对不同的卷积规模设置了不同的多核并行优化方案和核内优化方法,针对 Conv1 ×1 提出了特征图级多核并行方案,针对核大于 1 ×1 的卷积提出了窗口级多核并行优化设计,同时提出了逐元素向量化计算的核内并行优化实现。实验结果表明:并行优化方法单核计算效率最高能达到 64.95%,在带宽受限的情况下,多核并行扩展效率可达到 48.36% ~88.52%,在典型网络 ResNet50 上的执行性能与 E5-2640 CPU 相比,获得了 5.39 倍加速。该项研究工作对于推动国产自主 DSP 芯片在智能领域的应用具有重要意义。

致谢

感谢 FT-M7032 设计团队对于本文工作的大力支持。

参考文献(References)

[1] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84 -90.

[2] GIRSHICK R, DONAHUE J, DARRELL T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014: 580 -587.

[3] XU Z W, YANG Y, HAUPTMANN A G. A discriminative CNN video representation for event detection [C]// Proceedings of 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015: 1798 -1807.

[4] COLLOBERT R, WESTON J. A unified architecture for natural language processing: deep neural networks with multitask learning[C]//Proceedings of the 25th International Conference on Machine Learning, 2008: 160 -167.

[5] HINTON G, DENG L, YU D, et al. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups [J]. IEEE Signal Processing Magazine, 2012, 29(6): 82 -97.

[6] ZHANG C, WU D, SUN J Y, et al. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster[C]// Proceedings of the 2016 International Symposium on Low Power Electronics and Design, 2016: 326 -331.

[7] ZHAO R Z, NIU X Y, WU Y J, et al. Optimizing CNN-based object detection algorithms on embedded FPGA platforms[C]// Proceedings of International Symposium on Applied Reconfigurable Computing, 2017: 255 -267.

[8] BETTONI M, URGESE G, KOBAYASHI Y, et al. A convolutional neural network fully implemented on FPGA for embedded platforms [C]//Proceedings of 2017 New Generation of CAS (NGCAS), 2017: 49 -52.

[9] WANG Q L, MEI S Z, LIU J E, et al. Parallel convolution algorithm using implicit matrix multiplication on multi-core CPUs [C]//Proceedings of 2019 International Joint Conference on Neural Networks (IJCNN), 2019: 1 -7.

[10] JIA Y Q, SHELHAMER E, DONAHUE J, et al. Caffe: convolutional architecture for fast feature embedding[C]// Proceedings of the 22nd ACM International Conference on Multimedia, 2014: 675 -678.

[11] 王庆林, 李东升, 梅松竹, 等. 面向飞腾多核处理器的 Winograd 快速卷积算法优化[J]. 计算机研究与发展, 2020, 57(6): 1140 -1151.
WANG Q L, LI D S, MEI S Z, et al. Optimizing Winograd-based fast convolution algorithm on Phytium multi-core CPUs[J]. Journal of Computer Research and Development, 2020, 57(6): 1140 -1151. (in Chinese)

[12] HUANG X D, WANG Q L, LU S Y, et al. NUMA-aware FFT-based convolution on ARMv8 many-core CPUs[C]// Proceedings of 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, 2021: 1019 -1026.

[13] ZHANG J Y, FRANCHETTI F, LOW T M. High performance zero-memory overhead direct convolutions[C]// Proceedings of the 35th International Conference on Machine Learning, 2018: 5776 -5785.

[14] GEORGANAS E, AVANCHA S, BANERJEE K, et al. Anatomy of high-performance deep learning convolutions on SIMD architectures[C]//Proceedings of SC18: International Conference for High Performance Computing, Networking,

Storage and Analysis, 2018: 830 – 841.

[15] WANG Q L, LI D S, MEI S Z, et al. Optimizing one by one direct convolution on ARMv8 multi-core CPUs [C]// Proceedings of 2020 IEEE International Conference on Joint Cloud Computing, 2020: 43 – 47.

[16] YIN S F, WANG Q L, HAO R C, et al. Optimizing irregular-shaped matrix-matrix multiplication on multi-core DSPs [C]//Proceedings of 2022 IEEE International Conference on Cluster Computing, 2022: 451 – 461.

[17] REDDI V J, CHENG C, KANTER D, et al. MLPerf inference benchmark [C]//Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020: 446 – 459.

[18] PASZKE A, GROSS S, MASSA F, et al. PyTorch: an imperative style, high-performance deep learning library[C]//Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019: 8026 – 8037.

(编辑: 梁慧, 杨琴)