doi:10.11887/j.cn.202301009 http://journal.nudt.edu.cn

多核数字信号处理器矩阵乘卷积算法性能评测*

王庆林[1,2],裴向东[1],廖林玉[1,2],王浩旭[1,2],李荣春[1,2],梅松竹[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 并行与分布处理国防科技重点实验室,湖南 长沙 410073)

摘 要:矩阵乘卷积算法能够为各种卷积配置提供高性能基础实现,是面向给定芯片进行卷积性能优化的首要选择。针对国防科技大学自主研制的飞腾异构多核数字信号处理器(digital signal processor, DSP)芯片的特征以及矩阵乘卷积算法自身的特点,提出了一种面向多核DSP架构的高性能并行矩阵乘卷积实现算法ftmEConv。该算法由输入特征图转换、卷积核转换、矩阵乘以及输出特征图转换这四个均运行在通用多核DSP上的并行化部分构成,通过有效挖掘通用DSP核中功能单元的潜力来提升各个部分的性能。实验结果表明,ftmEConv实现了高达42.90%的计算效率,与芯片上的其他矩阵乘卷积算法实现相比,获得了高达7.79倍的性能加速。

关键词:多核数字信号处理器;卷积神经网络;卷积算法;算法优化
中图分类号:TN95 **文献标志码**:A **开放科学(资源服务)标识码(OSID)**:
文章编号:1001-2486(2023)01-086-09

Evaluating matrix multiplication-based convolution algorithm on multi-core digital signal processors

WANG Qinglin[1,2], *PEI Xiangdong*[1], *LIAO Linyu*[1,2], *WANG Haoxu*[1,2], *LI Rongchun*[1,2], *MEI Songzhu*[1,2], *LI Dongsheng*[1,2]
(1. College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China;
2. Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China)

Abstract: The matrix multiplication-based convolutional algorithm, which can efficiently implement convolutions with different parameters, is the first choice of convolution performance optimization for a given chip. Based on the architecture of Phytium heterogeneous multi-core DSPs (digital signal processors) developed by National University of Defense Technology and the characteristic of the matrix multiplication-based convolutional algorithm, a parallel implementation of the matrix multiplication-based convolutional algorithm (called ftmEConv) for different convolutions on multi-core DSPs was proposed. The ftmEConv consists of four parallelized parts (input feature maps transformation, filter transformation, matrix multiplication, and output feature maps transformation), all of which were optimized for multi-core DSPs, and the performance of each part was improved by effectively exploiting the potential of all functional units in DSP cores. The experimental results demonstrate that ftmEConv achieves computational efficiency of up to 42.90%. Compared with other implementations of the matrix multiplication-based convolutional algorithm on heterogeneous chips, ftmEConv gets a speedup of up to 7.79 times.

Keywords: multi-core digital signal processors; convolutional neural networks; convolutional algorithms; algorithm optimization

随着人工智能+(artificial intelligence +, AI +)的快速发展,深度学习技术逐渐在各个领域实现了技术落地。作为一类代表性深度神经网络,卷积神经网络(convolutional neural networks, CNNs)被广泛应用在各种场景中,如自动驾驶[1]、视频处理[2]、科学计算[3]等。在CNNs中,卷积层占据了大部分的计算开销,从而使得卷积层的优化成为CNNs网络性能的关键,目前成为学术界和工业界研究的热点。

实现卷积层计算的方法主要有直接卷积[4-5]、快速傅里叶变换(fast Fourier transform, FFT)卷积[6-9]、Winograd卷积[10-11]、矩阵乘卷积[12-14]四种算法。直接卷积算法根据卷积层的定义直接进行实现,为获得较高的性能,通常需要针对卷积核大小、卷积步长等卷积参数进行优化。FFT和Winograd卷积算法分别通过FFT和

* **收稿日期**:2022-09-13
基金项目:国家自然科学基金资助项目(62002365)
作者简介:王庆林(1987—),男,贵州思南人,副研究员,博士,硕士生导师,E-mail:wangqinglin_thu@163.com;
裴向东(通信作者),男,山西长治人,博士研究生,E-mail:18903588277@163.com

Winograd 转换来降低卷积复杂度,因此这两种算法也常被称为快速卷积算法。尽管如此,快速卷积算法通常只适用于部分卷积配置,如 Winograd 算法通常只适合于卷积核大小为 3 × 3 的情况。矩阵乘卷积算法是将卷积计算转换为通用矩阵乘操作,是实现卷积计算的通用算法,是 PyTorch[15]、TensorFlow[16] 等深度学习框架以及 cuDNN[17]、oneDNN[18] 等神经网络库首要提供的算法。矩阵乘卷积算法可分为显式算法和隐式算法。如果矩阵转换和矩阵乘融合为一体,则无须存储完整的转换矩阵,称为隐式算法,否则称为显式算法。本文主要聚焦显式矩阵乘卷积算法,构建完整的转换矩阵,通过调用已有的矩阵乘函数库来实现卷积计算,为各种卷积参数配置提供高性能实现基础。

FT-M7032 是国防科技大学面向 E 级计算自主研发的一款异构通用多核数字信号处理器[19](digital signal processors, DSP),由 32 个通用 DSP 核和 1 个 16 核 ARMv8 CPU 构成。在主频为 1.8 GHz 时,全芯片的单精度浮点峰值性能高达 11.06 Tflops/s,在科学计算和人工智能等领域具有巨大的潜力。在 FT-M7032 中,计算能力主要由 32 个通用 DSP 核提供。为了降低芯片面积和功耗,通用 DSP 核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,然后基于直接存储器存取(direct memory access,DMA)部件进行不同存储层次之间数据的传输。然而,面向 CPU、GPU 等芯片的已有算法在 FT-M7032 上没法直接运行或者没法实现高的性能。因此,针对多核 DSP 的体系结构进行算法优化是 FT-M7032 发挥高性能计算的必要措施。

面向 FT-M7032 处理深度学习应用的需求,本文结合其体系结构特征,在详细分析了 FT-M7032 芯片上实现显式矩阵乘卷积算法的各种技术路径的基础上,提出了一种面向多核 DSP 架构的高性能并行显式矩阵乘卷积实现算法 ftmEConv,并采用了不同的卷积配置对算法进行了详细性能评估。测试结果显示,ftmEConv 性能均超过了 FT-M7032 芯片上的其他显式矩阵乘卷积实现方法,获得了高达 7.79 倍的加速,最高达到了 42.90% 的多核 DSP 峰值性能。同时,针对算法开销进行了详细分析,也为后续面向 FT-M7032 的算法优化指明了方向。本文工作对于推动 FT-M7032 在人工智能领域的应用,以及面向 FT-M7032 的算法与应用优化,均具有重要的意义。

1　相关定义

1.1　卷积定义

本文的研究范围仅限于二维经典卷积,故令卷积的输入特征图为 $\boldsymbol{I}[N][C_d][H_i][W_i][L]$,输入卷积核为 $\boldsymbol{F}[C][K_d][H_f][W_f][L]$,卷积的输出特征图为 $\boldsymbol{O}[N][K_d][H_o][W_o][L]$。其中,$N$ 表示输入特征图的数量,$H_{i/f/o}$ 和 $W_{i/f/o}$ 分别表示空间上的高度和宽度,L 表示硬件向量处理单元并行处理的数据宽度,C_d 和 K_d 分别表示输入通道和输出通道的分块数,输入通道数为 $C = C_d \times L$,输出通道数 $K = K_d \times L$。卷积计算中的步长大小标记为 S,填充大小标记为 P,则输出特征图的高度和宽度分别为:$H_o = (H_i + 2 \times P - H_f)/S + 1$,$W_o = (W_i + 2 \times P - W_f)/S + 1$。基于以上参数表示,深度学习领域中的卷积定义如下:

$$\boldsymbol{O}_{n,k_d,h_o,w_o,k_l} = \sum_{c_d=0}^{C_d-1}\sum_{c_l=0}^{L-1}\sum_{h_f=0}^{H_f-1}\sum_{w_f=0}^{W_f-1}(\boldsymbol{I}_{n,c_d,h_o\times S+h_f-P,w_o\times S+w_f-P,c_l} * \boldsymbol{F}_{c_d\times L+c_l,k_d,h_f,w_f,k_l}) \tag{1}$$

其中,$0 \leqslant n < N$,$0 \leqslant k_d < K_d$,$0 \leqslant h_o < H_o$,$0 \leqslant w_o < W_o$ 以及 $0 \leqslant k_l < L$。

1.2　矩阵乘卷积算法

矩阵乘卷积算法是将卷积操作直接转换为通用矩阵乘计算。根据第 1.1 节卷积的定义,矩阵乘卷积的算法大致分为四步,如算法 1 所示。第一步(Step 1)是输入特征图转换,根据卷积操作的计算过程将输入特征图 $\boldsymbol{I}$ 转换为 $\boldsymbol{A}$ 矩阵。该步骤将单个通道特征图上的一个卷积点计算以及多输入通道上的累加操作所需来自 $\boldsymbol{I}$ 的全部元素转换为 $\boldsymbol{A}$ 矩阵的一行,称为 im2row (image-to-row) 操作;如果转换为 $\boldsymbol{A}$ 矩阵的一列,则称为 im2col (image-to-column) 操作。本文后续讨论中主要涉及 im2row 操作,$\boldsymbol{A}$ 矩阵常采用 $\boldsymbol{A}[M'][K']$ 表示,其中 $M' = N \times H_o \times W_o$,$K' = C_d \times H_f \times W_f \times L$,同时也根据算法的设计讨论需求采用其准确的数据布局 $\boldsymbol{A}[N][H_o][W_o][C_d][H_f][W_f][L]$ 表示。第二步(Step 2)将卷积核 $\boldsymbol{F}$ 转换为 $\boldsymbol{B}[K'][N']$ 矩阵,其中 $N' = K$。第三步(Step 3)执行矩阵乘操作 $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$,其中 $\boldsymbol{C}$ 矩阵大小为 $M' \times N'$。第四步(Step 4)将 $\boldsymbol{C}$ 矩阵转换为卷积的输出特征图 $\boldsymbol{O}$。

算法 1 原始矩阵乘卷积算法

Alg. 1 Original matrix multiplication-based convolutional algorithm

输入:$\boldsymbol{I}[N][C_d][H_i][W_i][L]$,$\boldsymbol{F}[C][K_d][H_f][W_f][L]$

输出:$\boldsymbol{O}[N][K_d][H_o][W_o][L]$

1. Step 1:基于卷积定义从输入特征图 $\boldsymbol{I}$ 中构建矩阵 $\boldsymbol{A}[N\times H_o\times W_o][C_d\times H_f\times W_f\times L]$(im2row 操作)
2. Step 2:通过格式转换将输入卷积核 $\boldsymbol{F}$ 转为 $\boldsymbol{B}[C_d\times H_f\times W_f\times L][K]$(transformF 操作)
3. Step 3:调用 BLAS 库中的 GEMM 函数完成 $\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$,其中 $\boldsymbol{C}[N\times H_o\times W_o][K]$(GEMM 操作)
4. Step 4:通过格式转换将矩阵 $\boldsymbol{C}[N\times H_o\times W_o][K]$ 转换为卷积的输出特征图 $\boldsymbol{O}[N][K_d][H_o][W_o][L]$(transformO 操作)

2 FT-M7032 体系结构

FT-M7032 是国防科技大学面向 E 级计算自主研发的一款异构通用多核 DSP,由一个 16 核 ARMv8 CPU 和 4 个 GPDSP 簇构成,其整体架构如图 1 所示。ARMv8 CPU 上运行 Linux 操作系统,负责进程与外设管理以及多芯片之间的通信,单精度浮点峰值性能为 281.6 Gflops/s。4 个 GPDSP 簇提供主要的计算能力支持,其中每个 GPDSP 簇由 8 个通用 DSP 核和大小为 6 MB 的全局共享内存(global shared memory, GSM)通过交叉开关网络连接而成。片上 GSM 带宽约为 307.2 GB/s。当 DSP 内核主频为 1.8 GHz 时,单个 GPDSP 簇可以提供高达 2.76 Tflops/s 的单精度浮点计算能力。CPU 与 4 个 GPDSP 簇共享全局内存空间,但每个 GPDSP 簇只能访问对应的局部 DDR 内存空间。单个 GPDSP 簇对应的内存空间理论带宽为 42.62 GB/s。

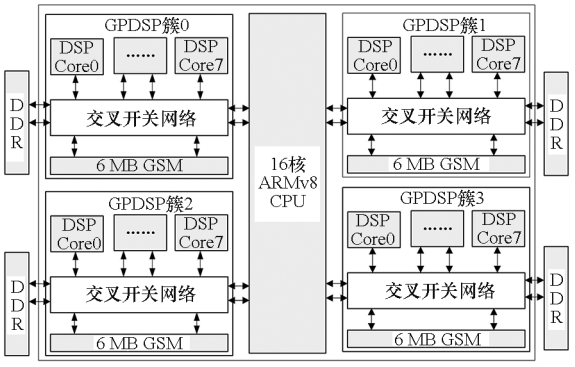

图 1 FT-M7032 芯片的整体架构

Fig. 1 Architecture of FT-M7032

FT-M7032 中单个 DSP 核的微架构如图 2 所示,主要由标量处理单元(scalar processing unit, SPU)、向量处理单元(vector processing unit, VPU)、指令调度单元(instruction fetch unit, IFU)以及 DMA 部件等构成。SPU 负责标量计算与流程控制,主要包括标量处理部件(scalar processing elements, SPE)和 64 KB 标量存储(scalar memory, SM)。SM 与寄存器之间的访问带宽为 28.8 GB/s。VPU 负责向量计算,主要由 16 个向量处理部件(vector processing elements, VPE)与 768 KB 向量存储(vector memory, AM)构成。每个 VPE 包含 3 个浮点乘累加(floating point multiply accumulator, FMAC)部件、1 个位操作(bit processing, BP)单元以及 2 个 Store/Load 部件,支持 6 条对应指令并行执行。16 个 VPE 以单指令多数据(single instruction multiple data, SIMD)的方式协作运行,一次可以处理 32 个单精度浮点数据(FP32),即对于 FP32 数据,$L=32$。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。

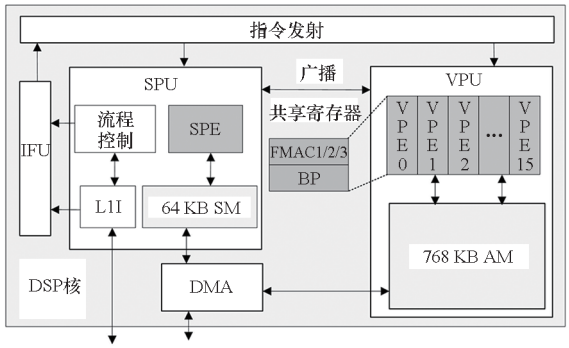

图 2 FT-M7032 中 DSP 单核微架构

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

3 面向 FT-M7032 的矩阵乘卷积实现分析

对照矩阵乘卷积算法的详细步骤以及 FT-M7032 的体系结构可知,在 FT-M7032 上实现高性能矩阵乘卷积算法主要有三种方法。

第一种方法是将矩阵乘卷积算法的四个步骤全部运行在 FT-M7032 的 16 核 ARMv8 CPU 上,其中第三步的矩阵乘通过调用 ARMv8 CPU 上的 BLAS 库来完成。该方法的优点是能够在 FT-M7032 上快速实现矩阵乘卷积算法,且工作量较小。但是该方法没有充分利用 FT-M7032 中 GPDSP 簇强大的计算能力,毕竟 GPDSP 簇的性能远高于 16 核 ARMv8 CPU 的性能。为便于后续的讨论与分析,统一将该方法标记为 Conv-CPU。

第二种方法是将 Conv-CPU 中的矩阵乘迁移

到其中一个 GPDSP 簇上执行,即第三步的矩阵乘通过调用 GPDSP 簇上的 BLAS 库来完成。为了保证 CPU 对四个簇的平衡控制,该方法实现其他三个步骤时,均只运行在单个 GPDSP 簇对应的四个 CPU 核上,即执行第一、二与四步骤时,并行的线程数均为 4。为便于后续的讨论与分析,统一将该方法标记为 Conv-CPU-DSP。

第三种方法是将矩阵乘卷积算法的四个步骤全部运行在 FT-M7032 的 GPDSP 簇上。与前面两种方法相比,该方法既能充分利用 GPDSP 簇的计算能力,也能基于 DSP 中的 DMA 部件提升 DDR 的访存效率。难点在于如何面向多核 DSP 的体系结构高效实现 im2row、transformF 以及 transformO 等访存密集型操作。

本文面向 FT-M7032 的矩阵乘卷积算法实现将采用第三种方法,拟基于 AM 与 GSM 相对 DDR 的高带宽属性,通过调用 DMA 操作、向量 Load/Store 操作的方法来高效实现 im2row、transformF 以及 transformO 等访存密集型操作,并通过集成已有的矩阵乘算法来高效完成矩阵乘操作。为便于后续的讨论与分析,将本文基于第三种方法的实现统一标记为 ftmEConv。

4　面向多核 DSP 的矩阵乘卷积算法优化

4.1　ftmEConv 算法整体设计

基于第 3 节的分析,并结合 FT-M7032 的体系结构特征,本文提出了面向多核 DSP 的矩阵乘卷积算法 ftmEConv,由六个步骤构成,如算法 2 所示。第一步(Step 1)将 CPU 缓存中的内容写回 DDR 中。第二步(Step 2)调用 DSP 端函数__im2row( )完成输入特征图的转换。第三步(Step 3)调用 DSP 端函数__transformF( )完成卷积核的转换。第四步(Step 4)调用 DSP 端函数__gemm( )完成矩阵乘的计算。第五步(Step 5)调用 DSP 端函数__transformO( )完成输出特征图的转换。第六步(Step 6)作废 CPU Cache 中的内容,完成卷积运算。

总的来说,ftmEConv 的主要操作过程(输入特征图转换、卷积核转换、矩阵乘以及输出特征图转换)均运行于通用 DSP 核上,通过有效挖掘通用 DSP 核的潜力来提升矩阵乘卷积的性能。同时,ftmEConv 的设计也使得数据在 CPU 端与 DSP 端之间的转换开销大幅降低,过程中仅需进行一次 DSP 读转换(Step 1)和 CPU 读转换(Step 6)操作即可。

算法 2　面向多核 DSP 的并行矩阵乘卷积算法 ftmEConv

Alg. 2　Parallel matrix multiplication-based convolutional algorithm on multi-core DSP (ftmEConv)

输入: $\boldsymbol{I}[N][C_d][H_i][W_i][L]$,$\boldsymbol{F}[C][K_d][H_f][W_f][L]$

输出: $\boldsymbol{O}[N][K_d][H_o][W_o][L]$

1. Step 1: 调用 cache_flush_all( )函数将 CPU Cache 中的内容写回 DDR
2. Step 2: 调用 DSP 端__ im2row($\boldsymbol{A}$,$\boldsymbol{I}$)函数构建 $\boldsymbol{A}$ 矩阵
3. Step 3: 调用 DSP 端__ transformF($\boldsymbol{B}$,$\boldsymbol{F}$)函数构建 $\boldsymbol{B}$ 矩阵
4. Step 4: 调用 DSP 端__ gemm($\boldsymbol{C}$,$\boldsymbol{A}$,$\boldsymbol{B}$)函数完成矩阵乘的计算
5. Step 5: 调用 DSP 端__ transformO($\boldsymbol{O}$,$\boldsymbol{C}$)函数获得卷积计算的输出特征图 $\boldsymbol{O}$
6. Step 6: 调用 cache_inv_all( )函数作废 CPU Cache 中的内容
7. Function __ gemm($\boldsymbol{C}$,$\boldsymbol{A}$,$\boldsymbol{B}$)
8. **if** $N' \geqslant 768$
9. Call TGEMM($\boldsymbol{C}$,$\boldsymbol{A}$,$\boldsymbol{B}$)
10. **else**
11. Call ftIMM($\boldsymbol{C}$,$\boldsymbol{A}$,$\boldsymbol{B}$)

4.2　输入特征图转换

结合 GPDSP 簇的体系结构特征,本文提出了面向多核 DSP 的 im2row 算法的并行实现方法,如算法 3 所示。该实现方法充分利用了片上 AM 和 GSM 相对 DDR 的高带宽特性,将输入特征图进行分块后传入 AM 空间,然后基于 DMA 函数先后完成两个维度的转换操作。具体为:在分块参数计算方面,鉴于深度学习中主流 CNNs 所处理的输入特征图空间维度相对较小等因素,同时根据片上 AM 和 GSM 空间大小的估计,本文决定将 $\boldsymbol{I}$ 传入片上空间最小粒度设置为 $W_i \times L$,即不在 W_o 维度上进行分块。根据 W_i、H_i、S 以及 P 等参数计算将 $\boldsymbol{I}$ 传入 AM 空间扩展后的特征图空间大小 $W_i' = (W_i + 2 \times P + S - 1)/S \times S$、$H_i' = (H_i + 2 \times P + S - 1)/S \times S$(第 1 行),从而使得填零后的特征图每行首个元素与每个通道首个元素均可以成为按步长 S 横向滑动后的卷积窗口中的首个元素。在算法 3 设计中,由于扩展后的分块输入特征图 $\boldsymbol{I}'_{am}$ 和转换后的矩阵 $\boldsymbol{A}'_{am/gsm}$ 分别存储在不同片上空间中,因此计算分块参数时,以 $\boldsymbol{I}'_{am}$ 和 $\boldsymbol{A}'_{am/gsm}$ 中的存储空间需求最大值来进行约束。同时,按以下原则来进行分块大小的调整:先设置 N、C_d、H_o 维度上的分块大小 N_b、C_{db}、H_{ob} 分别为 1;在片上空间限制下,尽可能先增大 H_{ob};如果 $H_{ob} = H_o$,再尽可能增大 C_{db};如果 $C_{db} = C_d$,然后尽

可能增大 N_b。

算法 3 im2row 操作的并行实现

Alg. 3 Parallel implementation of im2row algorithm

输入:$\boldsymbol{I}[N][C_d][H_i][W_i][L]$

输出:$\boldsymbol{A}[N][H_o][W_o][C_d][H_f][W_f][L]$

1. 根据 W_i、H_i、S 以及 P 等参数计算将 $\boldsymbol{I}$ 传入 AM 空间扩展后的特征图空间大小 H_i'与 W_i'等
2. 根据片上空间大小,计算 N、C_d、H_o 维度上的分块大小 N_b、C_{db}、H_{ob}
3. **for** $n=0{:}8{:}N_b{:}N$ **do** in parallel
4. $\quad n_b=\min(N-n,N_b)$
5. $\quad$ **for** $c_d=0{:}C_{db}{:}C_d$ **do** in parallel
6. $\quad\quad c_{db}=\min(C_d-c_d,C_{db})$
7. $\quad\quad$ **for** $h_o=0{:}H_{ob}{:}H_o$ **do** in parallel
8. $\quad\quad\quad h_{ob}=\min(H_o-h_o,H_{ob})$
9. $\quad\quad\quad$ 根据 h_{ob}推导输入上的 h_{ib}
10. $\quad\quad\quad$ Call im2row_kernel( )
11. Function im2row_kernel( )
12. $\quad$ Step 1:根据 P 的情况进行 AM 空间的初始化
13. $\quad$ Step 2:调用 $n_b\times c_{db}$次 DMA 函数将输入特征图 $\boldsymbol{I}$ 中的数据从 DDR 传输到 AM 空间,完成补零与扩展操作
14. $\quad$ Step 3:调用 1 次 DMA 函数将数据从 AM 空间传输到 GSM 空间,完成 W_f 维度的铺平转换操作
15. $\quad$ Step 4:调用 $n_b\times c_{db}\times H_f$ 次 DMA 函数将数据 GSM 空间再次传输到 AM 空间,完成 H_f 维度的铺平转换操作
16. $\quad$ Step 5:调用 1 次 DMA 函数将转换后的矩阵写回 $\boldsymbol{A}$ 中的对应位置

通过算法 3 中第 3 ~ 9 行的 for 循环,将输入特征图 $\boldsymbol{I}$ 划分成形状为$[n_b][c_{db}][h_{ib}][W_i][L]$的众多子块 $\boldsymbol{I}'_{ddr}$,然后采用多个 DSP 核分别调用 im2row_kernel( )函数来并行处理不同的子块。

im2row_ kernel ( ) 函数是本方案中实现 im2row 功能的核心函数,一共由五步构成,如算法 3 中第 11 ~ 16 行所示。下面以 $n_b=1$、$c_{db}=1$、$H_{ib}=H_i=3$、$W_i=3$、$H_f=W_f=2$、$S=2$ 以及 $P=1$ 为例进行每一步的详细介绍,如图 3 所示,其中每一个元素表示一个长为 L 的向量。

第一步(第 12 行的 Step 1):根据 P 的情况进行 AM 空间的初始化。如果 P 不等于 0,则意味着卷积计算过程中需要对特征图空间进行补零操作,本文通过对 AM 空间提前进行快速置零初始化来实现补零操作;如果 P 等于 0,则跳过此步的初始化。如图 3 中所示,由于 P 等于 1,则需将 AM 空间初始化为零。

第二步(第 13 行的 Step 2):调用 $n_b\times c_{db}$次 DMA 函数将输入特征图的子块 $\boldsymbol{I}_{ddr}[n_b][c_{db}][h_{ib}][W_i][L]$传入 AM 空间,并完成补零与扩展操作,从而将传入的输入特征图子块扩展为 AM 中的 $\boldsymbol{I}'_{am}[n_b][c_{db}][h'_{ib}][W'_i][L]$,其中 h'_{ib}表示传入的 h_{ib}经过补零与扩展后的大小。在图 3 中,$W'_i=6$、$h'_{ib}=6$;"0"表示补零后产生的向量,这是卷积计算的需求;"x"表示扩展后行和列中的无效向量元素,扩展后的子块在 Step 3 中即可通过一次 DMA 函数调用完成 W_f 维度的铺平转换。

第三步(第 14 行的 Step 3):调用一次 DMA 函数将 AM 空间中的 $\boldsymbol{I}'_{am}[n_b][c_{db}][h'_{ib}][W'_i][L]$传输到 GSM 中,转换成 $\boldsymbol{A}_{gsm}[n_b][c_{db}][h'_{ib}][W_o][W_f][L]$。在本步实现中,将 $\boldsymbol{I}'_{am}[n_b][c_{db}][h'_{ib}]$

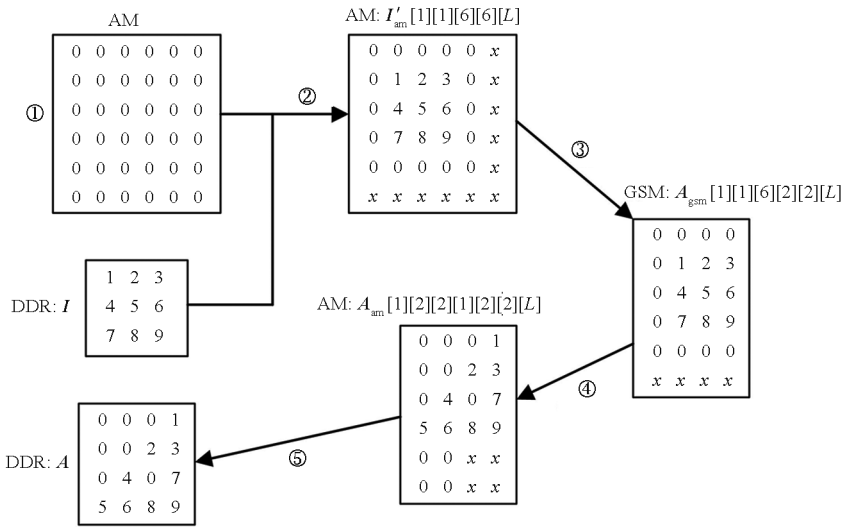

图 3 im2row_kernel( )的实现实例

Fig. 3 An implementation example for im2row_kernel( )

$[W'_i][L]$ 当作 $\boldsymbol{I}'_{am}[n_b \times c_{db} \times h'_{ib} \times W'_i][L]$ 来处理,通过合理设置 DMA 函数调用中的源块数、源块大小、源偏移量、目的块数、目的块大小以及目的偏移量来实现 W_f 维度的转换,其中通过将目的偏移量设置为负值来把矩阵中间扩展的"x"去掉。

第四步(第 15 行的 Step 4):调用 $n_b \times c_{db} \times H_f$ 次 DMA 函数将 GSM 空间 $\boldsymbol{A}_{gsm}[n_b][c_{db}][h'_{ib}][W_o][W_f][L]$ 传输到 *AM* 空间中,变成 $\boldsymbol{A}_{am}[n_b][h_{ob}][W_o][c_{db}][H_f][W_f][L]$,其中既涉及 H_f 维度的铺平转换,也涉及多个维度(如 $h_{ob} \times W_o$ 维度与 c_{db}维度)之间的转置。

第五步(第 16 行的 Step 5):调用一次 DMA 函数将 $\boldsymbol{A}_{am}[n_b][h_{ob}][W_o][c_{db}][H_f][W_f][L]$ 传输回 DDR 空间 $\boldsymbol{A}[N][H_o][W_o][C_d][H_f][W_f][L]$ 中对应位置。

4.3　卷积核转换

对于 1 ×1 卷积来说,$\boldsymbol{F}$ 与 $\boldsymbol{B}$ 完全相同,因而不需要进行任何转换;对于其他如 3 ×3 等的卷积来说,需要对 $\boldsymbol{F}$ 进行格式转换,本文的实现如算法 4 所示。在分块方面,由于 H_i 和 W_i 通常较小,在实现中仅需根据 AM 空间大小在 C 和 K_d 两个维度进行分块。单个 GPDSP 簇中 8 个通用 DSP 核并行将 $\boldsymbol{F}$ 分块后的子块传入 AM 空间(第 6 行),完成格式转换(第 7 行),然后传出到 DDR 中 $\boldsymbol{B}$ 的给定位置(第 8 行)。

算法 4　卷积核张量的并行转换

Alg. 4　Parallel transformation of filter tensors

输入:$\boldsymbol{F}[C][K_d][H_f][W_f][L]$
输出:$\boldsymbol{B}[C_d][H_f][W_f][L][K]$

1. 根据片上空间大小,计算 C、K_d 维度上的分块大小 C_b、K_{db}
2. **for** $c = 0 : C_b : C$ **do** in parallel
3. $\quad c_b = \min(C - c, C_b)$
4. $\quad$ **for** $k_d = 0 : K_{db} : K_d$ **do** in parallel
5. $\quad\quad k_{db} = \min(K_d - k_d, K_{db})$
6. $\quad\quad$ Step 1: 从 DDR 中将 $\boldsymbol{F}$ 子块传入 AM 的 $\boldsymbol{F}_{am}$
7. $\quad\quad$ Step 2: 按格式转换要求,将 $\boldsymbol{F}_{am}$ 转换成 $\boldsymbol{B}_{am}$
8. $\quad\quad$ Step 3: 从 AM 中的 $\boldsymbol{B}_{am}$ 传出到 DDR 的 $\boldsymbol{B}$ 中

4.4　矩阵乘

由于面向 FT-M7032 芯片的软件生态尚不完善,特别是如 BLAS 等数学库均没有成熟的版本,本文先根据文献[20]构建了基于 N' 维度并行的矩阵乘实现函数 TGEMM。然后,在 TGEMM 与不规则形状矩阵乘函数库 ftIMM[19] 之间根据 N' 的大小进行选择,如算法 2 中第 7 ~ 11 行所示。在 TGEMM 实现中,每个 DSP 核一次处理 N'维度上的分块大小为 96,只有 $N' \geqslant 8 \times 96 = 768$ 时,才能保证单个 GPDSP 簇中 8 个 DSP 核都能分配到相应的计算任务。为充分发挥单簇中所有 DSP 核的并行计算能力,只有当 $N' \geqslant 768$ 时,才调用 TGEMM 函数;否则,调用 ftIMM 中的优化实现。

4.5　输出特征图转换

在完成第 4.4 节矩阵乘函数调用后获得矩阵 $\boldsymbol{C}[M'][N']$,还需将其转换成卷积的输出特征图格式。本文提出了如算法 5 所示的输出特征图张量并行转换算法。在分块方面,由于主流 CNNs 中的 K 相对较小,因而根据片上 AM 空间大小,仅在 N 和 $H_o \times W_o$ 两个维度上进行分块。单个 GPDSP 簇中 8 个通用 DSP 核并行将分块后多个 $\boldsymbol{C}$ 子块传入 AM 空间(第 6 行),完成格式转换(第 7 行),最后传出到 DDR 中 $\boldsymbol{O}$ 的给定位置(第 8 行)。

算法 5　输出特征图张量的并行转换

Alg. 5　Parallel transformation of output feature maps tensors

输入:$\boldsymbol{C}[N][H_o][W_o][K]$
输出:$\boldsymbol{O}[N][K_d][H_o][W_o][L]$

1. 根据片上空间大小,计算 N、$H_o \times W_o$ 维度上的分块大小 N_b、HW_b
2. **for** $n = 0 : N_b : N$ **do** in parallel
3. $\quad n_b = \min(N - n, N_b)$
4. $\quad$ **for** $hw = 0 : HW_b : H_o \times W_o$ **do** in parallel
5. $\quad\quad hw_b = \min(H_o \times W_o - hw, HW_b)$
6. $\quad\quad$ Step 1: 从 DDR 中将 $\boldsymbol{C}$ 子块传入 AM 的 $\boldsymbol{C}_{am}$
7. $\quad\quad$ Step 2: 按格式转换要求,将 $\boldsymbol{C}_{am}$ 转换成 $\boldsymbol{O}_{am}$
8. $\quad\quad$ Step 3: 从 AM 中的 $\boldsymbol{O}_{am}$ 传出到 DDR 的 $\boldsymbol{O}$ 中

5　性能评估

5.1　实验设置

本节主要涉及 ftmEConv 与第 3 节所介绍的 Conv-CPU、Conv-CPU-DSP 之间的性能对比。其中,Conv-CPU 所有部分均运行在 FT-M7032 的 16 核 ARMv8 CPU 上,矩阵乘部分直接调用 OpenBLAS v0. 3. 1[21] 中的 cblas_sgemm 函数实现;Conv-CPU-DSP 中转换部分运行在 FT-M7032 中一个 GPDSP 簇对应的 4 个 ARMv8 CPU 核上,

矩阵乘部分则直接采用 ftmEConv 中矩阵乘的实现(第 4.4 节)。同时,三种算法实现均在相同的带宽下运行,即所有内存空间均分配在一个 GPDSP 簇匹配的局部内存空间。

在本节中涉及三个指标来表示卷积的性能,第一个是完成卷积计算的时间 T,第二个是卷积计算所到达的计算性能 P_{conv},第三个是卷积计算在单个 GPDSP 簇上所实现的计算效率 E_{conv}。三个指标之间的相互关系如式(2)与式(3)所示,其中 $Peak_{\text{gpdsp}}$ 表示单个 GPDSP 簇的峰值性能。

$$P_{\text{conv}} = \frac{2 \times N \times H_{\text{o}} \times W_{\text{o}} \times C \times H_{\text{f}} \times W_{\text{f}} \times K}{T} \tag{2}$$

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

5.2 ftmEConv 的开销剖析

本小节首先对运行在 FT-M7032 中一个 GPDSP 簇上 ftmEConv 实现的时间开销进行深度剖析,然后对 ftmEConv、Conv-CPU 以及 Conv-CPU-DSP 三种实现中占比相对较大的输入特征图转换进行性能对比分析。

当 $N=32$、$H_{\text{f}}=W_{\text{f}}=3$、$H_{\text{i}}=W_{\text{i}}=28$、$S=1$ 以及 $P=0$ 时,ftmEConv 中各部分开销的占比随 $C=K$ 取不同值的变化情况如图 4 所示。ftmEConv 的开销主要由输入特征图转换(输入转换)、卷积核转换、矩阵乘以及输出特征图转换(输出转换)四部分组成。当 $C=K$ 取不同值时,最耗时的始终是矩阵乘 GEMM 与输入转换 im2row 两部分。随着 $C=K$ 增大,矩阵乘部分的占比逐渐增大,最大达到了 92.41%,因而矩阵乘的性能是决定矩阵乘卷积性能的关键因素。尽管如此,输入转换 im2row 部分也始终占据一定比例的开销。在图 4 所示的测试中,输入转换 im2row 部分开销占比在 5.77%~33.83% 之间。

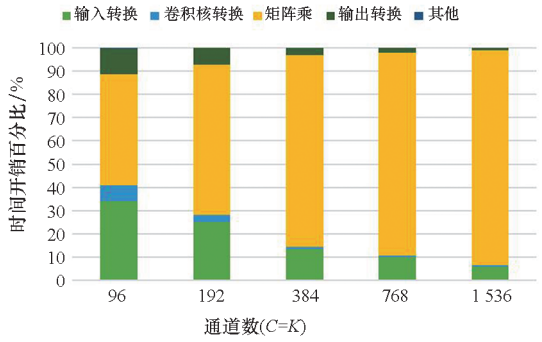

图 4 ftmEConv 在不同通道数设置下的开销分析

Fig. 4 Overhead analysis of ftmEconv with different channel sizes

当 $N=32$、$H_{\text{f}}=W_{\text{f}}=3$、$H_{\text{i}}=W_{\text{i}}=28$、$S=1$ 以及 $P=0$ 时,ftmEConv、Conv-CPU-DSP 以及 Conv-CPU 三种实现中输入转换 im2row 部分的性能随 $C=K$ 取不同值的变化情况如图 5 所示。

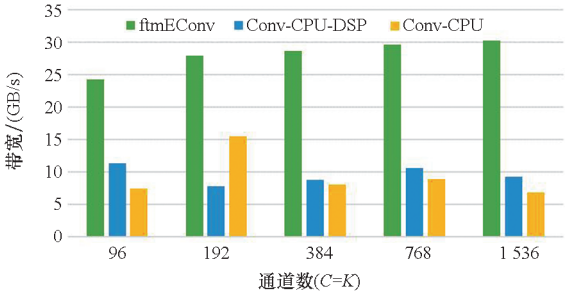

图 5 三种实现在不同通道数设置下的输入特征图转换性能

Fig. 5 Input transform performance of three implementations with different channel sizes

本文采用有效带宽来衡量输入转换部分的性能,即采用输入转换部分的理论数据访问量(等于 $\boldsymbol{I}$ 和 $\boldsymbol{A}$ 两个张量大小之和)除以其耗时。ftmEConv 中的输入转换部分实现了 24.9 ~ 29.97 GB/s的性能,对应的 DDR 带宽利用效率为 58.42%~70.32%。同时,ftmEConv 中输入转换部分显著优于 Conv-CPU-DSP 与 Conv-CPU 两者的实现,分别实现了高达 3.23 倍与 3.75 倍的性能加速。

5.3 不同实现的性能对比

本小节将采用不同的卷积参数配置对 ftmEConv、Conv-CPU-DSP 以及 Conv-CPU 三种实现进行全面的性能评测。

当 $C=K=384$、$H_{\text{f}}=W_{\text{f}}=3$、$H_{\text{i}}=W_{\text{i}}=28$、$S=1$ 以及 $P=0$ 时,三种实现性能随 N 不同取值的变化情况如图 6 所示。ftmEConv 的性能 P_{conv} 达到了 555.08 ~ 686.78 Gflops/s,对应的计算效率 E_{conv} 为 20.08% ~ 24.84%。相比 Conv-CPU-DSP,ftmEConv 实现了 1.43 ~ 1.61 倍的性能加速,主要

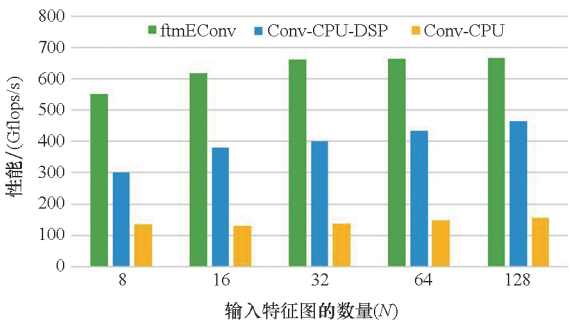

图 6 三种实现在不同输入特征图数量设置下的性能

Fig. 6 Performance of three implementations with different numbers of input feature maps

来源于面向多核 DSP 的三个转换过程的性能优化。相比 Conv-CPU,ftmEConv 实现了 4.73 ~ 5.13 倍的性能加速,除了三个转换过程的优化外,也得益于占比最大的矩阵乘部分的性能提升。

当 $N=32$、$H_f=W_f=3$、$H_i=W_i=28$、$S=1$ 以及 $P=0$ 时,三种实现性能随 $C=K$ 取不同值的变化情况如图 7 所示。ftmEConv 的性能 P_{conv} 随着 $C=K$ 增大而逐渐增大,最终达到了 1 186.10 Gflops/s,计算效率 E_{conv} 也达到了 42.90%。相比 Conv-CPU-DSP 与 Conv-CPU,ftmEConv 分别实现了 1.24 ~ 2.10 倍与 4.87 ~ 7.79 倍的性能加速。

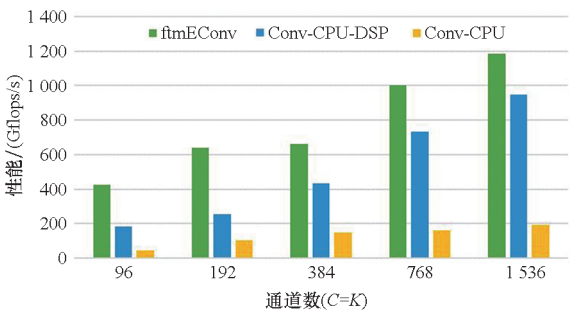

图 7　三种实现在不同通道数设置下的性能

Fig. 7　Performance of three implementations with different channel sizes

当 $N=32$、$C=K=384$、$H_f=W_f=3$、$S=1$ 以及 $P=0$ 时,三种实现性能随 $H_i=W_i$ 取不同值的变化情况如图 8 所示。ftmEConv 的性能 P_{conv} 随着 $H_i=W_i$ 减小而逐渐降低,当 $H_i=W_i=7$ 时,P_{conv} 为 263.60 Gflops/s。主要原因是随着 $H_i=W_i$ 减小,矩阵乘的维度 M' 快速变小,使得矩阵乘的性能逐渐降低,从而影响了 ftmEConv 整体的性能。尽管如此,ftmEConv 仍然在所有测试卷积层上获得了优于 Conv-CPU-DSP 与 Conv-CPU 两种实现的性能,相应的性能加速比分别为 1.19 ~ 1.58 与 2.41 ~ 5.07。

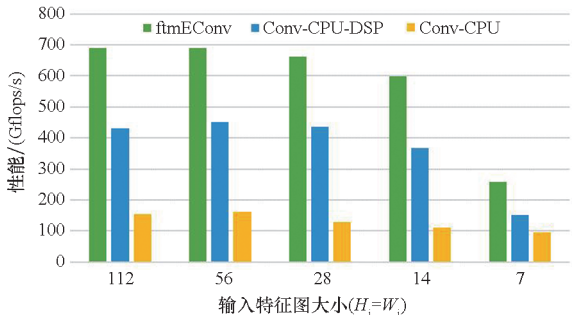

图 8　三种实现在不同输入特征图大小设置下的性能

Fig. 8　Performance of three implementations with different input feature maps sizes

当 $N=32$、$C=K=384$、$H_i=W_i=28$、$S=1$ 以及 $P=0$ 时,三种实现性能随 $H_f=W_f$ 取不同值的变化情况如图 9 所示。ftmEConv 的性能 P_{conv} 在 $H_f=W_f=1$ 时最低,在 $H_f=W_f=9$ 时最高,相应的计算效率 E_{conv} 分别为 16.96% 与 29.35%。相比 Conv-CPU-DSP,ftmEConv 在 $H_f=W_f=1$ 时获得了最大 3.87 倍的性能加速,主要是输入转换与输出转换部分的开销占比较大造成的。与 Conv-CPU 相比,ftmEConv 获得了 4.41 ~ 6.23 倍的性能加速。

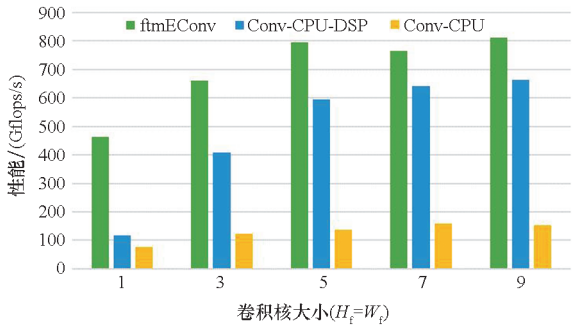

图 9　三种实现在不同卷积核大小设置下的性能

Fig. 9　Performance of three implementations with different kernel sizes

5.4　在典型网络上的性能测试

在本小节,采用典型网络 Resnet18[22] 中的卷积层来进行三种实现之间的性能对比测试,结果如图 10 所示。在图 10 中,横坐标表示来自 Resnet18 中的不同配置卷积层,其中 N 均设置为 128。与第 5.3 节的性能对比分析结果相似,在所有测试的卷积层上,ftmEConv 均优于 Conv-CPU-DSP 与 Conv-CPU 两种实现。具体而言,ftmEConv 获得了 348.42 ~ 512.87 Gflops/s 的性能,计算效率为 12.60% ~ 18.55%;相比 Conv-CPU-DSP 与 Conv-CPU,分别实现了 1.22 ~ 2.85 倍与 2.80 ~ 7.09 倍的性能加速。

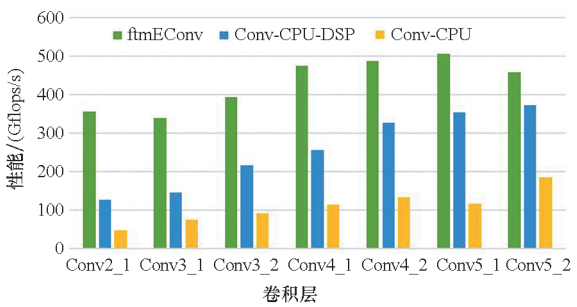

图 10　三种实现在 Resnet18 网络层上的性能

Fig. 10　Performance of three implementations for convolutional layers of Resnet18

6 结论

本文针对飞腾异构多核 DSP 的体系结构特征与矩阵乘转置操作的特点,提出了一种面向多核 DSP 架构的高性能并行显式矩阵乘卷积实现算法 ftmEConv。ftmEConv 由输入特征图转换、卷积核转换、矩阵乘以及输出特征图转换四个并行化部分构成,四个部分均运行在通用多核 DSP 上。ftmEConv 通过有效挖掘多核 DSP 的潜力来提升各个部分的性能,同时大幅降低了 CPU 端与 DSP 端之间的转换开销。实验结果显示,ftmEconv 能够显著加快 FT-M7032 芯片上的矩阵乘卷积操作,其计算效率最高达到了 42.90%;与 FT-M7032 芯片上的其他实现相比,获得了1.18 ~ 7.79 倍的性能加速。该项研究对于推动国产 DSP 在人工智能领域的广泛应用具有重要意义。

下一步将研究面向多核 DSP 的其他卷积算法实现,以期进一步提升 FT-M7032 芯片上的卷积实现性能。

参考文献(References)

[1] ZUO Z R, YU K, ZHOU Q, et al. Traffic signs detection based on faster R-CNN[C]//Proceedings of IEEE the 37th International Conference on Distributed Computing Systems Workshops, 2017.

[2] BONETTINI N, CANNAS E D, MANDELLI S, et al. Video face manipulation detection through ensemble of CNNs[C]//Proceedings of the 25th International Conference on Pattern Recognition (ICPR), 2021.

[3] CHEN X H, GONG C Y, LIU J, et al. A novel neural network approach for airfoil mesh quality evaluation[J]. Journal of Parallel and Distributed Computing, 2022, 164: 123 - 132.

[4] HAO R C, WANG Q L, YIN S F, et al. Towards effective depthwise convolutions on ARMv8 architecture[EB/OL]. (2022 - 01 - 24)[2022 - 08 - 21]. https://arxiv.org/abs/2206.12124.

[5] CHO M, BRAND D. MEC: memory-efficient convolution for deep neural network[EB/OL]. (2022 - 01 - 21)[2022 - 08 - 21]. https://arxiv.org/abs/1706.06873.

[6] WANG Q L, LI D S, HUANG X D, et al. Optimizing FFT-based convolution on ARMv8 multi-core CPUs[C]// Proceedings of European Conference on Parallel Processing, 2020.

[7] HUANG X D, WANG Q L, LU S Y, et al. Evaluating FFT-based algorithms for strided convolutions on ARMv8 architectures[J]. Performance Evaluation, 2021, 152: 102248.

[8] ZLATESKI A, JIA Z, LI K, et al. The anatomy of efficient FFT and Winograd convolutions on modern CPUs[C]// Proceedings of the ACM International Conference on Supercomputing, 2019: 414 - 424.

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

[10] LAVIN A, GRAY S. Fast algorithms for convolutional neural networks[C]//Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[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] CHELLAPILLA K, PURI S, SIMARD P. High performance convolutional neural networks for document processing[C]// Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition, 2006.

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

[14] 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.

[15] PASZKE A, GROSS S, MASSA F, et al. PyTorch: an imperative style, high-performance deep learning library[EB/OL]. (2019 - 12 - 03)[2022 - 08 - 22]. https://arxiv.org/abs/1912.01703.

[16] ABADI M, BARHAM P, CHEN J M, et al. TensorFlow: a system for large-scale machine learning[EB/OL]. (2016 - 05 - 31)[2022 - 08 - 22]. https://arxiv.org/abs/1605.08695.

[17] CHETLUR S, WOOLLEY C, VANDERMERSCH P, et al. cuDNN: efficient primitives for deep learning[EB/OL]. (2014 - 12 - 18)[2022 - 08 - 22]. https://arxiv.org/abs/1410.0759.

[18] INTEL. oneDNN: oneAPI Deep Neural Network Library[DB/OL]. (2022 - 05 - 12)[2022 - 08 - 22]. https://github.com/oneapi-src/oneDNN.

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

[20] 刘仲, 田希. 面向多核向量处理器的矩阵乘法向量化方法[J]. 计算机学报, 2018, 41(10): 2251 - 2264.
LIU Z, TIAN X. Vectorization of matrix multiplication for multi-core vector processors[J]. Chinese Journal of Computers, 2018, 41(10): 2251 - 2264. (in Chinese)

[21] ZHANG X Y, WANG Q, WERNER S. OpenBLAS: an optimized BLAS library[CP/OL]. (2022 - 05 - 12)[2022 - 08 - 22]. http://www.openblas.net/.

[22] HE K M, ZHANG X Y, REN S Q, et al. Deep residual learning for image recognition[C]//Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2016.