计算机学报
CHINESE JOURNAL OF COMPUTERS

面向深度学习的批处理矩阵乘法设计与实现

黄春[1)]姜浩[1)]全哲[2)]左克[1)]何楠[2)]刘文超[1)]

[1)](国防科技大学计算机学院，长沙 410073)

[2)](湖南大学信息科学与工程学院，长沙 410082)

摘 要 本文设计并实现了面向深度学习的统一框架批处理矩阵乘法。我们细致的分析了利用矩阵乘法实现卷积的过程中卷积核、输入特征图和输出特征图在 NCHW 和 NHWC 两类存储格式下的矩阵数据排列特点，指出了其和矩阵行列主序的关系。在此基础上，为了更好复用共享的卷积核数据，我们提出将批量输入特征图转化为一个矩阵整体进行计算的方法。我们设计了统一框架的批处理分块矩阵乘法，该框架计算同一矩阵和多个不同矩阵的乘法，可以处理并输出任意存储格式的矩阵数据。我们优化了分块矩阵乘法实现，根据输入参数特征规划计算顺序，利用矩阵转置技巧复用核心计算模块，没有增加额外的数据组织操作。数值试验表明：本文设计实现的批处理单精度矩阵乘法的计算速度比循环调用原始单精度矩阵乘法的计算速度在处理中小尺度矩阵时在四款不同处理器平台上性能最高分别提高 4.80%，26.57%，29.27%和 25.55%，平均分别提升 2.37%，14.37%，9.89%和 15.72%。

关键词 批处理矩阵乘法；卷积；分块算法；深度学习；数据排列；
中图法分类号 TP391

Design and Implementation of Batched GEMM for Deep Learning

HUANG Chun[1)]JIANG Hao[1)]QUAN Zhe[2)]ZUO Ke[1)]HENan[2)] LIU Wen-Chao[1)]

[1)](College of ComputerScience and Technology, National University of Defense Technology, Changsha 410073)

[2)](College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082)

Abstract In this paper, wepresent the designandimplementationof batched general matrix-matrix multiplication (GEMM) for deep learning. By analyzing the GEMM implementation of 2D convolution with the feature map layouts NCHW and NHWC, we show the relationship between the row-major (column-major) layouts of convolution matrix elements and the feature map layouts. Wedeem that the corresponding matrices of the input feature maps can be grouped together in a single batched routine to reuse the convolution kernels. We describe a unified and new API for our batched GEMM, which returns the multiplication of some matrix and several different matrices and can deal with any feature map in either data layout together. We optimize the block GEMM algorithm with the trick of matrix transposition and reorder the computing schedule without additional data rearrangement. Experimental results show that our batched GEMM enhance performance by up to4.80%, 26.57%，29.27% and25.55% and by 2.37%, 14.37%，9.89%和 15.72% in average than the convention GEMM recalled several times for four kinds of processors, respectively.

Key words batchGEMM；convolution；blockalgorithm；deeplearning；datalayout

———————————————

本课题得到国家重点研究发展项目(2020YFA0709803)、国防科技173计划项目(2020-JCJQ-ZD-029)、科学挑战专题资助项目(TZ2016002)、湖南省自然科学基金项目(2018JJ3616)、国家自然科学基金项目(61907034)资助. **黄春**，博士，研究员，主要研究领域为编译器和系统软件. E-mail:chunhuang@nudt.edu.cn. **姜浩(通讯作者)**，博士，助理研究员，主要研究领域为高性能计算、舍入误差分析和数值计算. E-mail: haojiang@nudt.edu.cn.**全哲**，博士，副教授，主要研究领域为人工智能、算法和编译器. E-mail:quanzhe@gmail.com. **左克**，博士，副研究员，主要研究领域为网络和编译器.E-mail：kezuo@nudt.edu.cn.何楠，硕士研究生，主要研究领域为深度学习.E-mail:henan@hnu.edu.cn. 刘文超，硕士，助理工程师，主要研究领域为高性能计算，计算物理.E-mail:work_liuwenchao@163.com.

1 引言

深度学习基于神经网络发展而来，训练深层神经网络需要大量数据和计算力。在深度学习神经网络中，卷积层是最耗时的部分，贾扬清在其博士论文中曾指出 CNN 模型中卷积层甚至最多占用总计算时间的 90%以上[1]。计算力的提升从软硬件两个方面同时助力卷积计算的加速优化，达到减少深度学习训练和推理的耗时。当前业界的主流深度学习框架针对卷积层的计算展开优化，在算法层面主要有四种实现方式：即 Im2Col+GEMM 矩阵计算[2]，FFT 快速傅里叶变换计算[3]，Winograd 算法计算[4]，直接卷积计算，分别针对通用处理器和 AI 加速器实现高效的卷积计算实现。

从软件角度来讲，主流深度学习框架有 AWS 亚马逊的 Mxnet，Google 的 TensorFlow，Facebook 的 PyTorch（PyTorch 和 Caffe2 已经合并），微软的 CNTK，百度的 PaddlePaddle，以及偏向研究性质的 Keras 和 Theano。上述框架在模型训练阶段都提供稠密矩阵乘法 GEMM 实现卷积计算。而在移动端的深度学习框架，腾讯的 NCNN 和百度的 Paddlelite 等采用了 Winograd 算法，Facebook 的 NNPACK 采用了 FFT 算法。

从硬件角度来讲，对于通用处理器如 X86CPU 和 ARMCPU，深度学习框架一般依赖于第三方库，其采用稠密矩阵乘法实现卷积。如 TensorFlow 和 Mxnet 在 X86CPU 上都调用了 OneDNN（前身为 DNNL 和 MKL-DNN）或 MKL；在 ARMCPU 上调用 OpenBLAS 或 Eigen 等开源高性能库，或者调用 ARM 公司开发的主要用于推理的 ACL 库。其中，只有 ACL 包含 Winograd 算法实现，其它都是基于稠密矩阵乘法。NVIDIA 公司的 Turing 架构芯片在处理卷积神经网络时的核心思想就是将卷积转化为矩阵计算，再利用张量处理器单元进行并行加速。但同时，NVIDIA 公司提供的 CuDNN 库中卷积部分使用了 Winograd 算法。Google 的张量处理器 TPU 则采用脉动阵列的计算方式来实现卷积，其本质是直接卷积计算模式，适合较大规模的卷积计算。华为的昇腾 AI 处理器采用了达芬奇架构[5]，属于特定域架构 DSA 芯片，设计了专门的存储转换单元完成 Im2Col 的操作，并利用 AICore 快速计算矩阵乘法来实现卷积运算。

目前对于深度神经网络的优化主要集中在两个方面，一个是运算层优化，另一个是图层优化。运算层优化当前主要依赖于第三方库，也就是高性能的数值代数算法库，如 MKL 或 OpenBLAS[6,7,8] 等，其主要通过矩阵乘法来快速实现卷积运算。但是第三方库仅提供标准的 BLAS 函数接口，严格限制了矩阵数据格式、存储方式和输入输出标准等，不能完全适配深度学习对矩阵乘法的使用方式方法，一定程度上导致简单使用第三方 BLAS 库实现的神经网络卷积的性能很难达到第三方库的实际最高性能。比如在多核处理器上，想要矩阵乘法 GEMM 的性能接近机器峰值性能，往往要求矩阵尺度达到一定程度，而深度学习中卷积所用到的大量小尺度的矩阵乘法往往很难达到处理器峰值性能。运算层的优化还涉及一些其它如批量标准化，池化，激活等内核函数的实现。运算层的优化和硬件关系密切，如数据的存储方式 NHWC 和 NCHW 等。具体的运算层算法库实现有 OneDNN(DDNL，MKL-DNN)，ACL，CuDNN，TinyDNN 等。而图层的优化更多是由框架来完成，包括内存分配，内核融合和计算调度等和硬件无关的优化。

本文关注的是软件运算层的优化。我们针对深度学习的特点特征，面向硬件 CPU 端，改进第三方高性能库，增加第三方 BLAS 库的函数接口，提出了批处理的矩阵乘法实现，更好的完成批量的深度学习卷积运算，开发卷积层的多级并行性，使其充分发挥硬件的计算性能，

新一代线性代数库 BLAS 的接口 API 设计计划包含批处理线性代数运算[9]，如批处理矩阵乘法，批处理 LU 分解等。该提案由超级计算机 TOP500 排名的发起者美国田纳西大学 JackDongarra 教授等人于 2016 年提出，其指出当前和未来科学计算中需要一次完成大量的小尺度矩阵运算[10]。批处理的概念天然的适合在加速器或协处理器如 GPU 上执行，文献[11]是批处理线性代数在 GPU 上的较早的应用。Dongarra 团队开发的 MAGMA 软件集成了批处理矩阵运算[12,13]，在 GPU 上取得了很好的加速效果。硬件厂商对于批处理线性代数也显示出了极大的研究热情。NVIDIA 在 CUBLAS 中提供了优化的批处理的 BLAS[14].INTEL 也在 MKL 数学核心算法库中嵌入了批处理矩阵乘法[15]。ARM 公司在 APL 数学库提供类似函数接口[16]。需要注意这里还有一些批处理矩阵运算的变种，如 CompactBLAS 关注于矩阵压缩存储来开发向量化并行[17];Intel 提供的 Packed BLAS 接口关注共享矩阵的中间存储

利用[15]等。其中 PackedBLAS 中的 PackedGEMM 和本文提出的面向深度学习的批处理矩阵乘法功能类似，但是本文算法更加通用，可以说 PackedGEMM 是本文算法的一个子集。

在深度学习领域，通过 Im2Col+Gemm 计算卷积的时候，不可避免的遇到内存大量冗余的问题。文献[18,19]通过采用 path-matrix 的方法，将一个输入特征矩阵由 CHW 格式转化为批量小矩阵的存储格式 CKKHW 格式，在多通道多卷积核 MCMK 情况下，减少了内存的使用。但是其仅仅处理一张特征图，本文方法则可以处理批量特征图。此外，MEC 方法[20]通过修改 Im2Col 后的数据存储格式，调用多个小尺度 GEMM 实现卷积，降低了整体的内存消耗，提高了卷积效率。但是其输出格式默认为是 HNWC，与原始输入格式 NHWC 略有区别，有些时候需要额外的转化操作。文献[21]则是通过存储数据指针的方式来降低内存的消耗。上述方法中调用矩阵乘法 GEMM 的方式、方法和数据存储格式都与原始 GEMM 有所不同，由于篇幅所限，本文仅仅关注 NHWC 和 NCHW 存储格式下，通用 Im2Col 方法得到的输入特征矩阵。

本文基于开源的第三方数值代数算法库 OpenBLAS，设计了面向深度学习的统一框架的批处理矩阵乘法实现。我们首先细致的分析了卷积过程中卷积核、输入特征图和输出特征图在 NCHW 和 NHWC 两类存储格式下的矩阵乘法特点。考虑到卷积核生成的矩阵对于所有输入特征图是共享的，我们提出将批量输入特征图转化为一个矩阵进行矩阵乘法计算，将很好的复用这一共享数据。接着，我们基于 OpenBLAS 的多级分块算法框架，结合输入输出特征图的具体尺度，设计了自动选择匹配的分块参数的方法。最后，我们对于混合行列主序的一般情况，提出了统一框架的批处矩阵乘法。

本文结构如下：第二节分析了深度学习中 NCHW 和 NHWC 两类存储格式下的卷积所对应矩阵乘法特点；第三节给出了两类存储格式下批处理矩阵乘法的设计实现，并引申出统一框架批处理矩阵乘法的概念思想和实现途径；第四节的数值实验显示本文提出算法的优越性；第五节对本文做了全面总结，并对未来工作给予了展望。

2 深度学习中矩阵乘法的特点分析

2.1 卷积运算的数据存储和矩阵乘法实现

深度学习的卷积层通过二维卷积计算来提取输入特征图（FeatureMap）的特征，构成输出特征图。卷积核又称为滤波器（Filter），不同卷积核可以提取不同特征。卷积过程就是对输入特征图的像素进行邻域加权求和并加上偏置的过程，这个加权求和的过程就是与卷积核做卷积的过程。

卷积核和输入输出特征图一般采用四维张量存储，目前应用较多的是 NCHW 和 NHWC 这两类。其中 N 表示批次中图像的数量，C 表示通道数（深度，如 RGB 图像有 3 通道），H 表示高度（垂直维度像素个数），W 表示宽度（水平维度像素个数）。存储格式直接影响后续卷积实现方法的性能。绝大部分深度学习框架比如 Pytorch，Mxnet 和 Caffe 采用 NCHW 格式，而 TensorFlow 默认采用 NHWC 格式，此外 OpenCV 默认使用 NHWC 格式。硬件上考虑的话，GPU 端主要负责训练，其希望访问同一个通道的像素连续，一般是采用 NCHW 格式，也是 CuDNN 的默认格式。

但是比较而言，NHWC 访存局部性更好，与 NCHW 相比占用较少的临时空间，缓存的利用率会更高，因而 CPU 端做预测时则更喜欢 NHWC 格式。NCHW 和 NHWC 可以相互转化，它们和矩阵的行主序和列主序密切相关。后续文中为了便于分析 NCHW 和 NHWC 存储格式下卷积实现格式，我们采用表 1 中的参数表示方式。

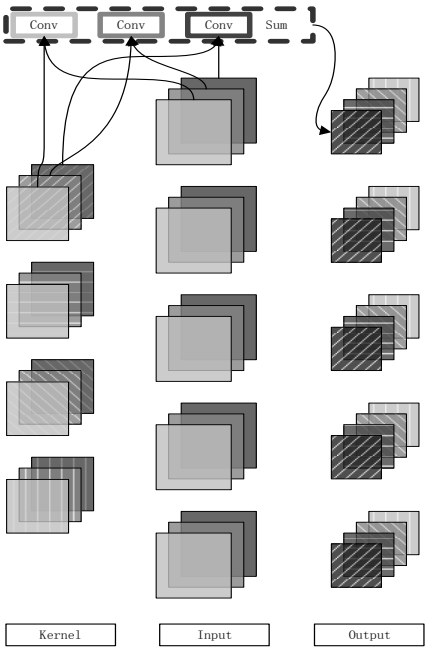

图 1 卷积示意图

表 1 NCHW 和 NHWC 存储格式对应的卷积表示

	N 个数	C 通道数	H 高度	W 宽度
卷积核	N_kernel	C_kernel	H_kernel	W_kernel
输入特征图	N_input	C_input	H_input	W_input
输出特征图	N_output	C_output	H_output	W_output

卷积核的权重矩阵一般采用 1*1、3*3、5*5 和 7*7 等大小的矩阵，对应于宽度 W_kernel 和高度 H_kernel。输入特征图的单个通道的图片的横向和纵向像素个数分别对应于宽度 W_input 和高度 H_input。输入特征图的通道数和卷积核的通道数相同，假设通道数为 C_kernel=C_input，如彩色图像的有 RGB 共 3 个颜色通道，对应通道的二维卷积核与二维输入特征图做二维卷积后。C_kernel(C_input)个通道的二维输出特征图累加到一起成为这个卷积核的最终输出特征图，其中 H_out 和 W_out 由 H(W)_kernel(input)决定，还和卷积过程的填充 padding 和步幅 stride 设置有关。卷积层通常需要提取多个特征，因此往往具有多个不同的卷积核，对于同一个输入特征图，N_kernel 个卷积核会生成 C_out=N_kernel 通道数的二维输出特征图。为了充分利用带宽和硬件计算性能，卷积层通常会一次处理 N_input 个输入特征图并生成 N_output 个输出特征图，满足 N_input=N_output. 具体细节可参见图 1，其为 5 批次 4 卷积核 3 通道的卷积示意图。

应用矩阵乘法实现卷积运算广泛应用于通用处理器上，预处理过程需要将卷积核和输入特征图分别展开成矩阵 A 和 B，Im2Col 算法通常用来生成 B 矩阵，然后调用已经优化好的 GEMM 计算 A*B 快速得到卷积计算结果。在 Mxnet 中，Im2Col 算子仅仅将 3 维 CHW 格式的输入特征图转化为行主序的矩阵格式，并没有提供针对一批图像的 Im2Col 算子。关于 Im2Col 的具体实现以及偏置值的处理方法可以参见[5].矩阵乘法实现卷积可以减少内存的访问次数，保证数据的空间局部性优异，但是其在 Im2Col 过程中原始输入特征图的数据被冗余存储，消耗了更多的内存空间。从并行角度考虑来讲，卷积层的不同并行部分可以用分块矩阵乘法来解释，即矩阵乘法自身蕴含的并行性可以完美适配卷积的并行，但是多个输入特征图的整体矩阵乘法实现目前还没有相关研究。

为了论述清晰，本文后续关注矩阵乘法 C=A*B，其中，C 是输出特征图转化的矩阵，A 是卷积核转化的矩阵，B 是输入特征图转化的矩阵。A 矩阵的尺度为 m*k，B 矩阵尺度为 k*n，C 矩阵尺度为 m*n。由于矩阵乘法的转置交换性，这种简单设定对于算法分析和设计是没有问题的。同时，在 Mxnet 中，默认设定卷积核转化的矩阵 A 作为左乘矩阵。

2.2 NCHW存储格式下矩阵乘法实现卷积

NCHW 又称为 channel_first，数据排序如图 2 所示。在内存中数据是按照地址连续存储，多维张量数组本质上可以看作是一维数组。

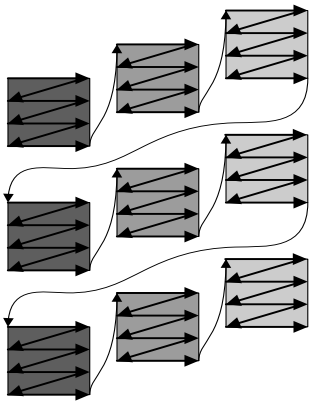

图 2 NCHW 存储示意图

首先，给出卷积核转化为矩阵的示意图，见图 3.卷积核的大小为 3*3，卷积核的个数为 8，卷积核的通道数为 3，按照行主序格式存储，和原始数据的 NCHW 一致，则 A 矩阵满足：

m=N_kernel=8,

k=C_kernel*Kernel_size=3*9=27.

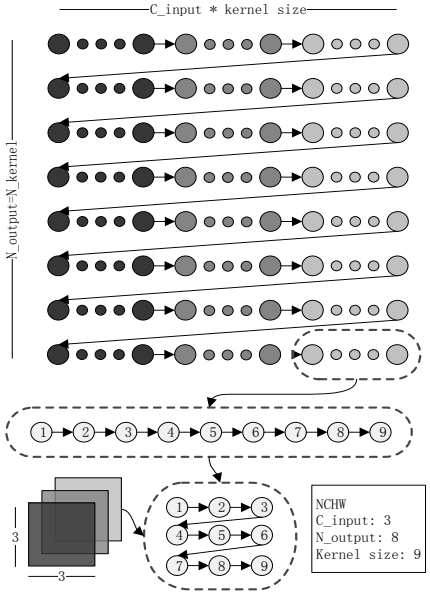

图 3NCHW 存储下多通道多卷积核转化矩阵示意图

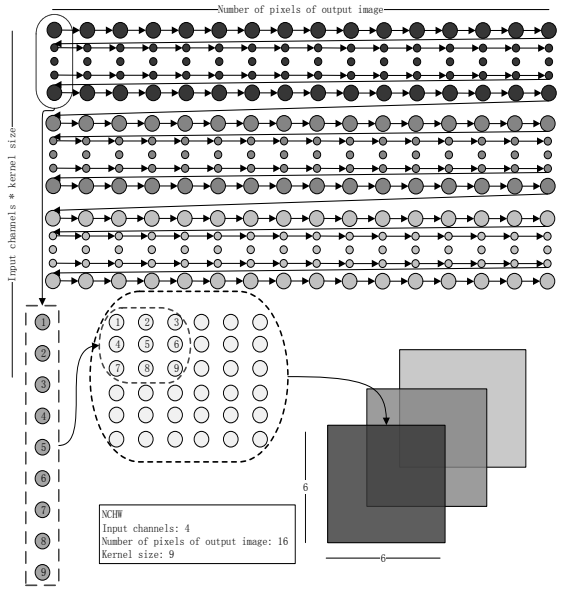

图 4NCHW 存储下多通道输入特征图转化矩阵示意图

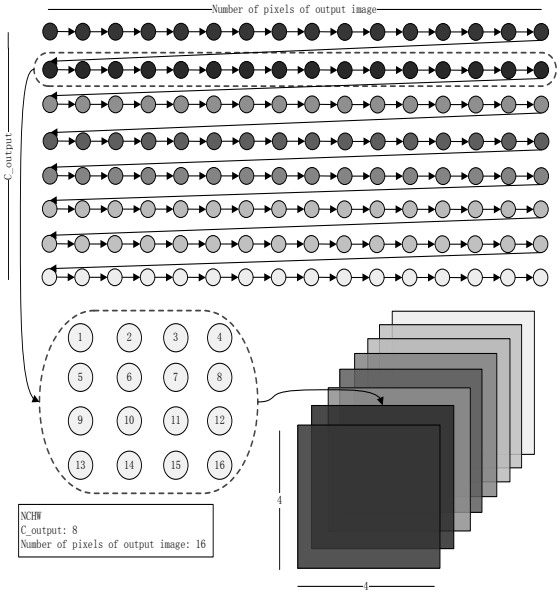

图 5NCHW 存储下多通道输出特征图矩阵示意图

接着，给出输入特征图转化为矩阵的示意图，见图 4.输入特征图的通道数和卷积核的通道数相同，输入特征图纵向和横向的像素 H=W=6，无填充 padding 和步幅 stride=1，C_input=C_kernel,则 B 矩阵满足：

k=C_input*Kernel_size=3*9=27

n=H_output*W*output=4*4=16.

为了满足 B 和 A 相同的行主序格式，Im2Col 函数需要对原始输入数据进行循环拷贝处理。Col 指的就是将卷积核对应的输入特征子矩阵转化为一列，列数对应了输入特征子矩阵的个数，也就是输出特征图的像素个数。从数据的空间连续性考虑，Im2Col 拷贝的数据绝大部分情况是局部连续的，即原始 NCHW 格式下的输入特征矩阵的连续两个元素绝大部分情况下也是B矩阵相邻的两个元素。

最后，给出输出特征图对应的矩阵的示意图，见图 5.同样 C 矩阵满足行主序格式，每一行为一个通道，每一行中按照行主序来排列，行数即为输出特征图的通道数，可以发现目前的格式即为 NCHW 模式，则 C 矩阵满足：

m=C_output=N_kernel=8

n=H_output*W*output =4*4=16.

需要注意的是：NCHW 格式下，这里的 B 和 C 矩阵对应的数据限定 N=1，也就是说仅仅针对批次内输入特征图为 1 的情况，每次调用 GEMM 来计算一个多通道输入特征图和多卷积核的卷积操作。

2.3 NHWC存储格式下矩阵乘法实现卷积

NHWC 又称为 channel_last，数据排序如图 6 所示。NHWC 应用最多的是 TensorFlow 框架，本文探讨 NHWC 格式下，卷积所需要的最合适的 GEMM 模式。

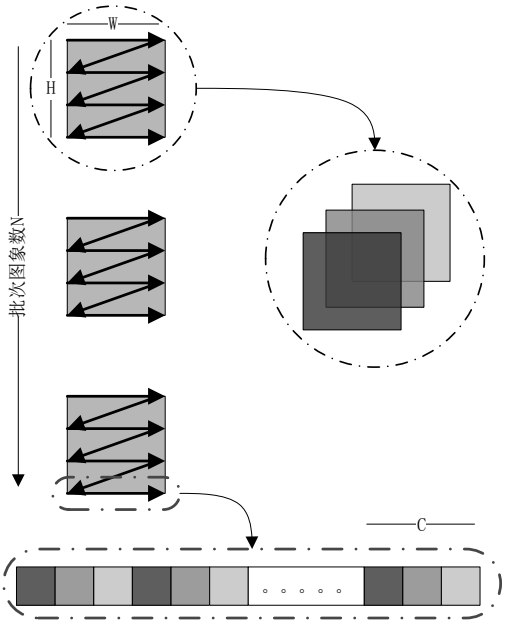

图 6 NHWC 存储示意图

首先，我们分析卷积核转化为矩阵的过程，限制其为左乘矩阵 A，参数设置同图 3 一致，卷积核的大小为 3*3，卷积核的个数为 8，卷积核的通道数为 3，考虑利用数据空间局部性，选择行主序格式，具体数据顺序见图 7.依旧有 m=8, k=27.

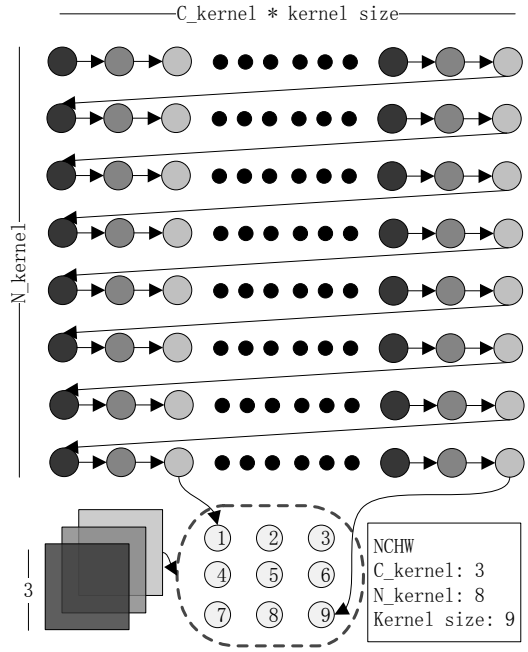

图 7NHWC 存储下多通道多卷积核转化矩阵示意图

接着，我们分析输入特征图转化为矩阵的过程，限制其为右乘矩阵，参数设置同图 4 一致。输入特征图的通道数和卷积核的通道数相同，输入特征图纵向和横向的像素 H=W=6。考虑数据连续性，我们认为列主序是最优的选择，假设选择行主序，那么相邻的两个元素在原始输入特征图中必然不是相邻的，这会造成 Im2Col 算法实现的效率较低。

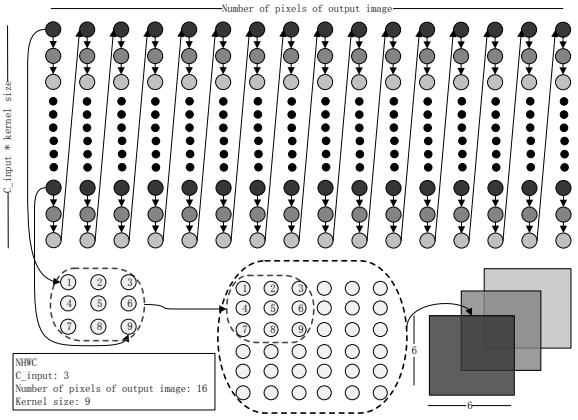

图 8　NHWC 存储格式下多通道输入特征图矩阵示意图

最后，我们给出输出特征图对应的矩阵的示意图，见图 9.由于 NHWC 的限制，显然列主序是必然的选择，行数为输出特征图的通道数，每一行依然对应一个通道。

NHWC 存储格式下，通过分析我们得出矩阵 A 为行主序，矩阵 B 和 C 为列主序是最优的。但是这不符合标准的 BLAS 接口，标准接口要求指定 A，B 和 C 具有相同的排序规则，所以需要对 A 做行主序到列主序的转化操作，这个操作的花费一定的时间，但是由于卷积核不大，A 矩阵的尺度也就不大，且考虑 A 矩阵的复用性，这个转化操作的花费还是可以接受的。

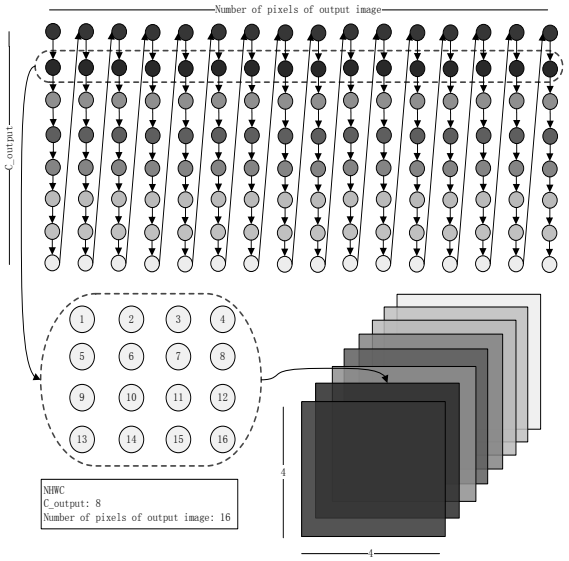

图 9　NHWC 存储格式下多通道输出特征图的矩阵示意图

本章简述了当前深度学习领域中对于矩阵乘法的使用情况，分析了应用较为广泛的 NCHW 和 NHWC 两种数据存储格式下的矩阵乘法所需的存储格式和特点，探讨了卷积计算多级并行和矩阵乘法多级并行的对应关系，并指出了当前矩阵乘法实现卷积的限制条件等。

3 面向深度学习的批处理矩阵乘法

深度学习中在经典的 NCHW 和 NHWC 两种存储格式下，对于每批次 N_input=1 张输入特征图情况，卷积可以便捷的转化为矩阵乘法实现。而对每批次 N_input>1 的情况，多数深度学习框架采用循环的方法处理，每次只通过 GEMM 计算一张图的卷积计算结果。这样往往导致 GEMM 的尺度不是很大，即 m，k 和 n 的值相对较小，针对当前的多核处理器很难发挥出计算性能优势。如果采用多进程实现来处理，则没有很好的复用卷积核转化的矩阵，同时多进程实现对于多级 Cache 共享的处理器会造成数据冲突，影响整体效率。矩阵乘法 GEMM 内部通过多线程 OpenMP 或 Pthread 实现其并行性，所以采用多线程计算批处理卷积在实现上也是存在困难的。因此，针对深度学习的需求，改进 BLAS

库的函数接口和内部实现，使其更好的匹配深度学习的卷积计算过程，充分发挥处理器的计算性能是迫切和必须的。

3.1 批处理矩阵乘法接口分析

3.1.1 NCHW 存储结构下批处理矩阵乘法

根据图 3、图 4 和图 5 所示的 NCHW 数据存储结构下矩阵存储结构示意图，我们认为批处理的矩阵乘法满足图 10 所示的排序规则。其中 B=[B1 B2 B3]矩阵和 C=[C1 C2 C3]矩阵可以看成是分块行主序矩阵。由于 B 和 C 矩阵不是整体行主序的矩阵，因此不能直接调用 BLAS 库中的 GEMM 来计算。

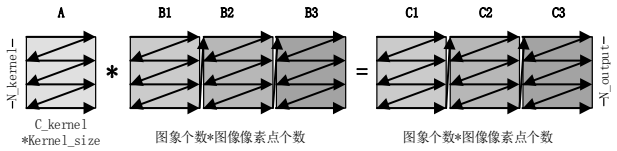

图 10 NCHW 存储格式下批量 GEMM 存储示意图

考虑到输出特征图是 NCHW 数据结构，因此 C 不可以是整体行主序，考虑到输入特征图通过 Im2Col 转化的 B 矩阵和 C 矩阵的对应关系，B 矩阵也不可能是整体行主序。所以需要在 BLAS 库中提供类似图 10 的批处理矩阵乘法函数接口，满足同一个左乘矩阵和多个尺度相同的右乘矩阵相乘得到多个尺度相同的矩阵。即满足：

C1=A*B1; C2=A*B2; C3=A*B3.

显然可以只拷贝 A 一次，之后进行多次复用。如果用显示的多次调用 GEMM 函数，在开源的 BLAS 库如 OpenBLAS 或 BLIS 内部会针对 A 多次进行数据分配 buffer 和数据重排操作，对于尺度中等的矩阵，这种拷贝的花费相对于其计算量来讲，就显得尤为明显。Intel 针对这类矩阵乘法，在 MKL 中提出了将 A 预先分配一块 buffer 的方法，通过调用 cblas_sgemm_pack 和 cblas_sgemm_compute 等函数来实现这个过程。在深度学习中 A 矩阵是卷积核，其尺度往往不大，数据量有限，且 A 是持续性的循环被复用，所以在空间局部性和时间局部性方面有优势，非常适合多次复用的情况。同时，此类问题显然不适合调用 cblas_sgemm_batch 函数来计算，因为该函数关注的主要是多个 A 和多个 B 一一对应的矩阵乘法操作。

3.1.2NHWC 存储结构下批处理矩阵乘法

根据图 7、图 8 和图 9 所示的 NHWC 数据存储结构下矩阵存储结构示意图，我们认为对应的批处理的矩阵乘法满足图 11 所示的排序规则。我们发现在 NHWC 格式下矩阵 C 和 B 具有整体列主序的特点，这天然适配了通用 BLAS 库的 GEMM 标准接口，唯一不一样的是矩阵 A 是行主序，而矩阵 B 和 C 是列主序。考虑到 BLAS 内部对于 A 还要根据 Cache 接口进行 packing 操作，我们认为提供一种新的 GEMM 接口是最为简便的方式，即在 cblas_sgemm 分别指定 A，B 和 C 的数据存储方式是行主序还是列主序。

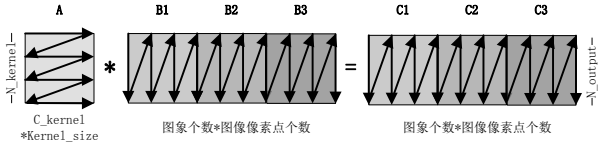

图 11 NHWC 存储格式下 GEMM 存储示意图

3.2批处理矩阵乘法实现

根据上一节的分析，我们分别针对 NCHW 和 NHWC 存储方式，针对标准 BLAS 的 GEMM 函数提出了新的要求，即具有共享左乘矩阵的批处理行主序矩阵乘法，以及行主序和列主序混合的矩阵乘法。最后基于这两类批处理矩阵乘法，提出了统一框架的面向深度学习的批处理矩阵乘法实现。

3.2.1 共享左乘矩阵的行主序批处理矩阵乘法

本文基于开源软件 OpenBLAS 实现共享左乘矩阵的批处理矩阵乘法，要求存储格式为行主序。由于 OpenBLAS 中对于行主序的格式，内部通过矩阵转置互换转化为列主序格式实现，即行主序的 C=A*B 可以看成列主序的 $C^T=B^T*A^T$ 来实现。所以图 10 所示的矩阵乘法将转化为图 12。

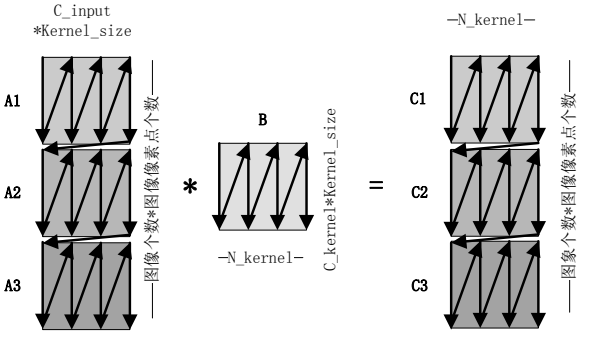

图 12NCHW 存储格式下转置后批处理 GEMM 示意图

如图 12 所示，我们所需的批处理的 GEMM 和原始 GEMM 相比，A 和 C 矩阵为多个小矩阵上下拼接而成，小矩阵内部满足列主序。同一般的批处理 GEMM 相比，我们的批处理矩阵的小矩阵地址

首尾相接，我们的批处理矩阵乘法共用一个B矩阵。这些特点使得我们的批处理矩阵乘法可以更好的利用通用矩阵乘法的数据复用思想，以达到更好的性能发挥。根据 OpenBLAS 的设计思路，我们给出了批处理矩阵乘法的算法实现图，见图 13.

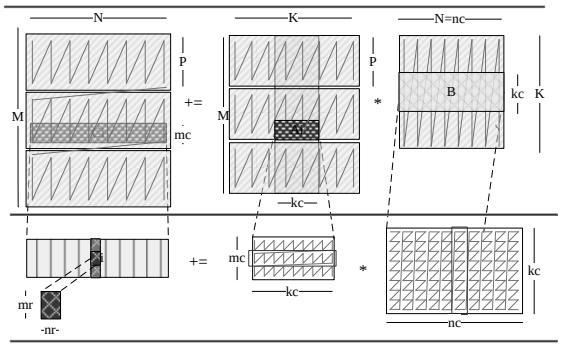

图 13 第一类批处理的 GEMM 分块算法示意图

分块矩阵乘法的核心思想是将原始矩阵分块后，使得块内数据地址连续存储于 L1Cache 或 L2Cache 内，因此我们重点设计优化两个部分，一是数据分块打包的设计，二是数据分块计算存储优化。考虑到 OpenBLAS 的数据多级并行设计，以及批处理矩阵乘法以图像数据点个数 P 为纵向长度循环的数据组织方式，我们在数据打包 Packing 的过程中，必须保证数据块在纵向自适应的刚好分割 P 长度，见图 14，换句话说就是，保证数据打包不会跨图像的。图 14 中 GEMM_P 是由具体平台的 Cache 体系结构特点决定，和 GEMM_Q,GEMM_R 一起作为该平台 GEMM 的分块参数。这样打包过程的输入数据局部连续，同时分块矩阵乘法的输出矩阵的也保证了连续存储。在最内层的分块矩阵乘法的实现和原始 OpenBLAS 的实现具有同样的打包重排序的输入矩阵 sa 和 sb，仅仅针对输入输出的分块子矩阵 C 的首地址和跳步，需要考虑根据批处理矩阵的数据排序方式做相应的修改。

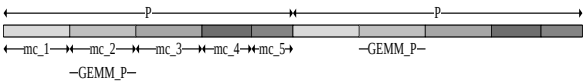

图 14 分块参数自适应图像像素点个数示意图

从图 12 可以发现：如果 A2 矩阵的首地址和 A1 矩阵的末尾数据地址不是连续的，在上述自适应分割优化技术下，只需要在打包重排序的过程中，基于 A2 的首地址计算分块矩阵的首地址偏移即可。因此标准的批处理矩阵乘法的接口 API 同样适用于此类问题的求解。可以预见：本节设计的批处理的矩阵乘法的性能应该无限接近于 M 值较大的通用矩阵乘法的性能，最大限度的发挥平台的计算性能。

3.2.2 混和行列主序矩阵乘法实现

针对图 11 数据组织的矩阵乘法实现，基于深度学习所用到的矩阵乘法尺度，参考 OpenBLAS 的分块实现模式，我们认为将图 11 做左右矩阵交换 $C^T=B^T*A^T$ 来实现是最优的，这个时候 A 和 B 互换。由于 M 较 K 和 N 都大一些，这样处理后，分块矩阵乘法对于 Cache 的利用会更好，同时这样处理也利用后续并行矩阵乘法的实现。其实现如图 15 所示。

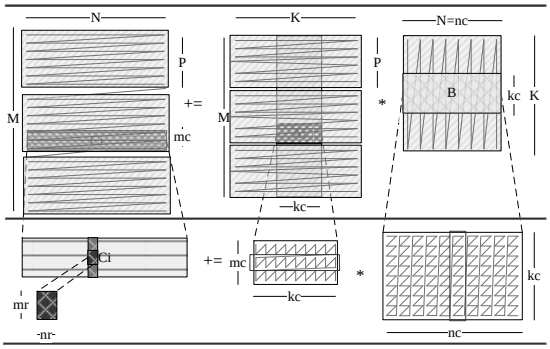

图 15 第二类批处理的 GEMM 分块算法示意图

对比图 14 和图 15 我们发现，这两类批处理 GEMM 的右乘矩阵是一致的，左乘矩阵由若干 P*K 的小矩阵纵向堆叠而成。区别在于小矩阵的排序方式分别为列主序和行主序。这一区别导致第二类的批处理矩阵乘法的左乘矩阵整体上看是行主序的，因此可以不必考虑 P 分割问题。但是这样一来输入输出矩阵 C 为行主序，其汇编内核的向量寄存器使用就必须做相应修改。因为汇编内核设计时默认 C 为列主序的时候，其在纵向 load 一组连续数据存入向量寄存器，并 store 向量寄存器的值到一组连续的纵向地址空间。

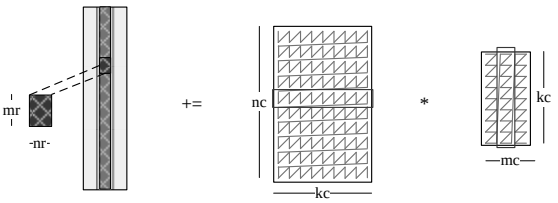

图 16 GEMM 分块算法转置交换示意图

幸运的是这里有一个非常实用的小技巧可以 g 规避上述问题：一般的汇编内核的最内层分块参数一般要求满足 mc 是 mr 的整数倍，nc 是 nr 的整数

倍数，如果还同时满足 mc 是 nr 的整数倍数，nc 是 mr 的整数倍数，那么就可以采取矩阵转置置换的方式，通过将 mc*kc 矩阵和 kc*nc 矩阵做 $C^T=B^T*A^T$ 变换来快速实现，这样可以直接调用原来的汇编内核，这个时候 mr 和 nr 实际上是互换了的。具体见图 16.对比图 16 和图 15，我们注意到在重排序 kc*nc 的矩阵的时候，原始的方法是根据 nr 的大小做排序，但是这里需要根据 mr 的大小做排序；相应的重拍 mc*kc 矩阵的时候也从根据 mr 做排序改成根据 nr 大小做排序。重排序过程中对于行列序的方向是没有变化的，这样一来汇编内核是不变的。对于混合行列主序的矩阵实现过程中，虽然看似两次转置交换操作是相互抵消的，但是本文方法更匹配 OpenBLAS 的多级并行分块算法设计框架，即最主要的循环为 M 方向维度的循环，能获得更好的计算性能。

3.2.3 统一框架的深度学习批处理矩阵乘法实现

基于上两节的工作，我们发现可以用一个统一框架解决我们的问题。针对深度学习卷积的批处理矩阵乘法，其两个矩阵分别对应卷积核和输入特征图所转化的矩阵，在尺度大小上天然就不一致，因而不适于随意交换。本文在第二章解释矩阵乘法实现卷积的过程中采用行主序描述，因此设定左乘矩阵 A 为卷积核矩阵。但是在本章节的代码实现阶段，本文强制设定右乘矩阵 B 为卷积核转化的列主序排列的矩阵。A[i]对应批量输入特征图纵向排列的一组小矩阵，小矩阵可以是列主序也可以是行主序，对应 layoutA[i]={Rowmajor，Colmajor}，每个小矩阵的尺度是 p[i]*k，跳步是 ldA[i]。C 和 A 情况类似。考虑到深度学习训练需要单精度矩阵乘法 sgemm，我们给出如下函数接口，其函数接口说明见表 2.

```
cblas_sgemm_dnn_batch(count, *layoutA,layout,
                      *layout, *transA, transB, *p,
                      n, k,*alpha, **A, *ldA,*B,
                      ldB, *beta, **C, *ldC)
```

本文提出的统一框架的深度学习批处理矩阵乘法具有如下特点：

1. 输入左乘子矩阵有自己私有的首地址。也就是说输入数据流可以不连续，Im2Col转化的矩阵可以具有不同的首地址，方便Im2Col的多线程实现优化，可充分利NUMA特性，保证数据空间局部性。
2. 输入左乘子矩阵有自己私有的排序选择。即可以第一个子矩阵是行主序，第二个子矩阵是列主序，也就是说子矩阵可以是NCHW和NHWC任意一种数据格式转化而来，使得输入数据流具有混合存储格式特点。
3. 输出子矩阵可以选择私有排序格式。也就是说可以选择是NCHW还是NHWC格式，可以避免之后额外需要的转置操作；同时，本文算法可以转化特征图的存储格式，如当输入特征图为NCHW格式，利用本文的批处理矩阵乘法可以输出NHWC格式的输出特征图。
4. 左乘子矩阵的行数可以不一致，但是列数必须一致。也就是说输入的特征图可以大小不同，可以同时对不同尺度规格的图片做卷积操作，但是卷积核大小要求一致的。

表 2 函数接口说明

	类型	输入输出	描述
count	int	In	子矩阵数量
layoutA	int[count]	In	{行主序，列主序}
layoutB	int	In	{行主序，列主序}
layoutC	int[count]	In	{行主序，列主序}
transA	int[count]	In	{转置，不转置}
transB	int	In	{转置，不转置}
p	int[count]	In	输入左乘子矩阵行数
n	int	In	输入右乘矩阵列数
k	int	In	输入左乘矩阵列数
alpha	float[count]	In	参数
A	float[count]	In	左乘矩阵指针
ldA	int[count]	In	左乘子矩阵跳步
B	float	In	右乘矩阵指针
ldB	Int	In	右乘矩阵跳步
beta	float[count]	In	参数
C	float[count]	In/Out	输出矩阵指针
ldC	int[count]	In	输出子矩阵跳步

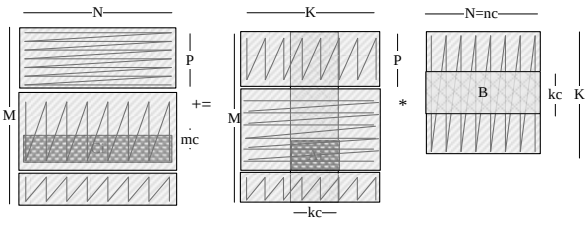

图17 统一框架下批处理的GEMM分块算法示意图

统一框架批处理的矩阵乘法的分块算法如图17所示，其计算流程图见图18，与图13和图15比较

会发现，其核心汇编计算的过程是高度相似的。只是数据组织上有了修改，可以预见整体的程序的性能应该是一致的，不会因为复杂的程序接口设置而导致性能损失。如图18所示的计算流程所示，首先将计算顺序重排序，根据C[i]和右乘B的排序方式分为四类，然后每一类中又根据左乘矩阵A的数据排列方式，选择不同类型的子矩阵重排方式，最终调用一致的汇编内核函数。整体上看总共有8中情形。注意观察到当C[i]为行主序的Case3和Case4时候，B的重排矩阵为sa，而A[i]的重排矩阵为sb，与Case1和Case2的情况正好相反，这个过程即为3.2.2章节最后一段论述的实现技巧，使得最后调用的汇编Kernel规格统一为mr*nr，避免了内核代码和重排函数的重复开发，降低了整体性能优化的难度。图19选取了图18中Case3的第二类情况作为例子，展示算法具体的实现方式。其本质就是如何利用转置和重排实现相同汇编内核的调用，而汇编内核默认输出矩阵为列主序。需要强调的是矩阵sa和其转置矩阵sa^T在一维数据角度看是没有区别的，仅在二维角度看实现了转置，满足$C[i]=sb*sa$到$C[i]^T=sa^T*sb^T$的转变。

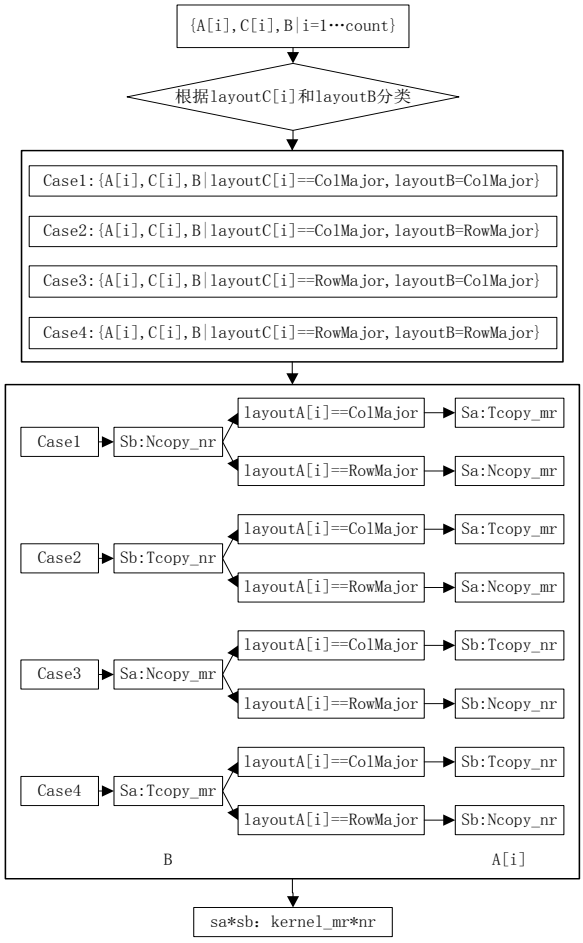

图 18　统一框架下批处理的 GEMM 算法流程图

统一框架批处理的矩阵乘法的并行实现过程中，要求每个线程分配的计算范围的纵向宽度按照子矩阵整分割。换句话说就是每个线程计算如图13，图 15 或图 17 的矩阵乘法，保证每个线程都使用完整的若干个子矩阵作为输入。多线程批量矩阵乘法在汇编内核上和单线程批量矩阵乘法是一致的。整体上采用的原来 OpenBLAS 自带的并行框架设计。

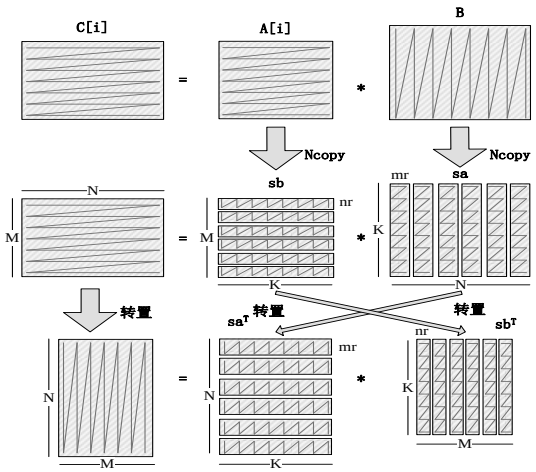

图 19Case3 中第二类情况举例说明

与一般通用批处理单精度矩阵乘法[9]接口相比（如下面的 cblas_sgemm_batch 所示），本文提出的面向深度学习的批处理做了一些裁剪和优化，使得其更匹配深度学习中卷积的计算特征。如取消了组的概念，因为分组卷积的操作是一组特征图和对应的一组卷积核做卷积，不同组的特征图对应不同的一组卷积核。在矩阵乘法上可以看成是通用批处理矩阵乘法，在整体上可以看成是块对角矩阵和块对角矩阵的乘法，这类矩阵乘法因为块矩阵乘法之间没有共享数据，因此优化手段有限，可以转化为多次调用本文算法来实现。

```
cblas_sgemm_batch(group_count,*group_sizes,layout
                  ,*transA, *transB, *m, *n, *k, *alpha,
                  **arrayA, *ldA, **arrayB, *ldB, *beta,
                  **arrayC, *ldC,*info )
```

同原始单精度矩阵乘法相比（如下面的 cblas_sgemm 所示），本文方法可以看成是多次调用原始单精度矩阵乘法，且每次调用相同的 B 矩阵。不同的是本文方法可以要求 A[i]和 B 的数据排序方式不同，即可以分别是行主序和列主序，且输出矩阵 C[i]可以是行主序和列主序中的任意一种。

```
cblas_sgemm (layout,transA, transB, m, n, k, *alpha,
```

*arrayA, ldA, *arrayB, ldB, beta,
*arrayC, ldC)

4 数值实验

4.1 测试环境

为了展示本文算法的通用性，本文选取的实验平台的 CPU 分别为 ARM 架构的 ThunderX2 和 FT2000+以及 X86 架构的 AMD Ryzen 7 2700x 和 Intel Xeon Gold 6130，共 4 款服务器作为硬件平台。详细参数如表 3 所示。测试的软件环境和工具情况在表 4 中列出。本文主体工作基于 OpenBLAS 开源软件设计实现。

表 3 测试平台参数

CPU	主频	峰值（单核单精度）
ThunderX2	2.5GHz	40Gflops
FT2000+	2.2GHz	17.6GHz
AMD Ryzen 7 2700x	3.7GHz	59.2Gflops
Intel Xeon Gold 6130	2.1GHz	67.2Gflops

表 4 测试软件环境和工具

编译器	Gcc8.3.0
设计开发软件	OpenBLAS-0.3.7
对比测试软件	MKL-20.04，APL-20.03
数据处理工具	Python3.6.1
深度学习框架	Mxnet-1.5.0

4.2 性能测试

4.2.1 与开源软件对比测试

在各平台上测试本文提出的面向深度学习的批处理单精度矩阵乘法的性能，并和循环调用原始单精度矩阵乘法作对比。图 20，图 21、图 22 和图 23 分别展示了 4 款平台上的数值性能测试结果。其中，黑色方柱代表本文提出的批处理单精度矩阵乘法的测试结果，白色方柱代表循环调用原始单精度矩阵乘法的测试结果。测试程序的单精度矩阵乘法为 OpenBLAS 开源软件针对具体测试平台所自动匹配的架构来选择的分块参数和内核。x 轴表示批次内矩阵数量为 8 到 256，y 轴表示子矩阵尺度 m=n=k 从 40 到 1280，z 轴表示性能。测试的矩阵规格为图 18 中 Case1 的第一类情况，即所有的矩阵 A[i],C[i]和 B 矩阵都为列主序。由于原始单精度矩阵只能处理 Case1 的第一类情况和 Case4 的第二类情况，所以我们仅列出了 Case1 的数值实验的性能对比结果。

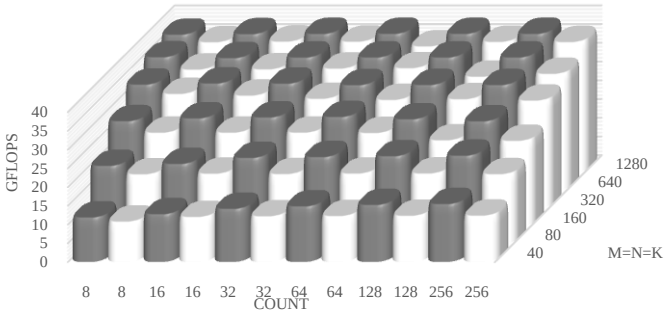

图 20ThunderX2 平台矩阵乘法性能对比实验

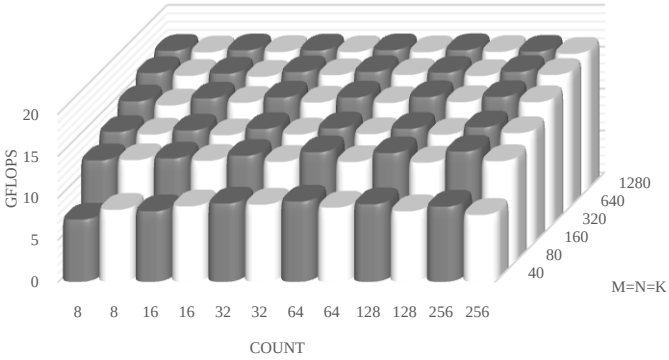

图 21FT-2000+平台矩阵乘法性能对比实验

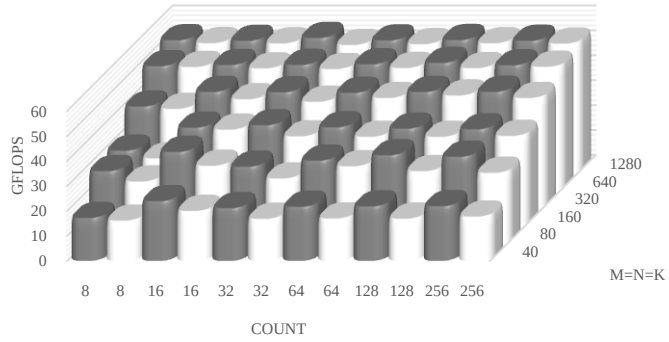

图 22AMD Ryzen 7 2700x 平台矩阵乘法性能对比实验

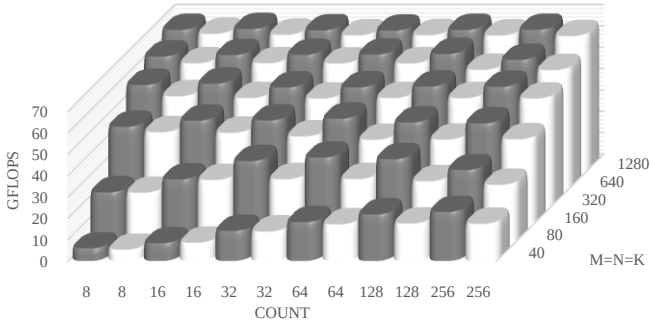

图 23Intel XeonGold 6130 平台矩阵乘法性能对比实验

从上面四图可以看出：整体上批处理单精度矩阵乘法的性能相对于循环调用原始单精度矩阵乘法有了明显提高，但是个别处性能有所抖动。测试的时候我们采用的线程绑定核的设定，避免的线程迁移造成的性能波动。需要强调的是这四个计算平台采用的是同一套基于 OpenBLAS 修改的代码，只是最后调用的 Kernel 内核汇编代码根据不同平台有所不同。这样的测试保证了无论是批处理的单精度矩阵乘法还是循环调用单精度矩阵乘法，两者最后都是调用了相同的汇编内核，摒弃了其它因素对于结果的影响，说明了本文算法改进策略的有效性和可信性。

本文对于统一框架批处理矩阵乘法中的 8 种（见图 18）展开性能测试，其结果非常接近，符合我们理论分析的预期。即批处理矩阵乘法的 8 种情况调用相同的汇编内核，采用相同的分块参数，理论上性能相同。测试中所有的批处理单精度矩阵乘法较循环调用原始单精度矩阵乘法的性能提升明显，本文对不同平台的性能相对提升比率情况进行了统计分析，结果如表 5 所示。

表 5　各平台性能优化结果统计

CPU	min	max	avg
ThunderX2	5.69%	26.57%	14.37%
FT2000+	-12.8 %	4.80%	2.37%
AMD Ryzen 7 2700x	0.70 %	29.27 %	9.89 %
Intel Xeon Gold 6130	2.31%	25.55%	15.72%

从表 5 可以看到，对于各个平台，从平均性能角度看，本文提出的批处理矩阵计算性能有了明显提升。但是最小值的统计结果表明，在某些局部情况，本文算法的优势有限。结合前面的结果分析，可以发现这样的点属于较为极端的情况，总体上性能是提升的，适用于大批量小尺度和中等尺度矩阵的运算。对于尺度 1280 情况，所有平台上的算法性能提升都不是那么显著了。此外，针对 FT2000+处理器平台，本文算法性能提升不是特别明显，这里应该和 FT2000+没有 L3cache 和 4 核共享 L2Cache 的体系结构特点密切相关。同其它三种计算平台相比，本文批处理矩阵乘算法的数据共享方法在 FT2000+上适配性较差，需要针对性的深度优化。

4.2.2 批处理矩阵乘法性能变化规律

为了更好地观察计算性能优化随矩阵数量（count）和矩阵尺度（M=N=K）的变化规律，本文将批处理单精度矩阵乘法的性能值与循环调用原始单精度矩阵乘法的性能值做除法得到二者比值，画出了 ThunderX2 的性能变化的分布图如图 24 所示。其中，x 轴表示子矩阵尺度 m=n=k 从 40 到 1280，y 轴表示性能比值，不同方柱对应了批次内矩阵数量分别为 1,8,16,32,64,128 和 256。从图 24 中可以看到，性能优化的效果从左往右依次递减，即对于矩阵数量较多的中小尺度矩阵，批处理单精度矩阵乘法有着较好的性能优化效果。

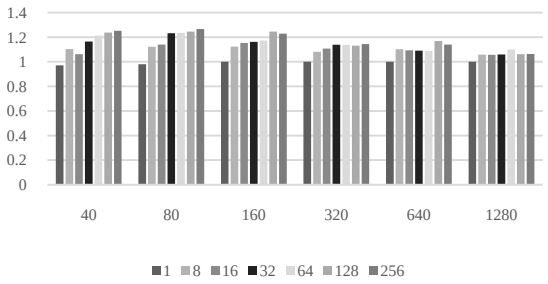

图 24ThunderX2 平台算法性能比值随 count 值和矩阵尺度（M=N=K）的变化规律。

在 count=1 时，本文提出的批处理单精度矩阵乘法将退化为原始单精度矩阵乘法，其计算性能结果如图 24 中第一根方柱所示。根据算法，此时二者的计算性能应该是一致的。图 24 中 count=1 的对应的性能比值非常接近于 1，在小尺度矩阵如情况时比值略小于 1，这是由于算法在实际的实现过程中，增加了少量的额外分支判断操作。在 count>32 时，在各个矩阵尺度的计算中，性能都有提升，特别是在中小尺度的情形中，性能提升较为明显。

此外，我们还测试了批量大尺度矩阵的性能，即 M，K，N 都大于 2000 以上，其性能和循环调用原始单精度矩阵乘法的性能几乎一样。考虑到大尺度矩阵乘法的性能已经是所能达到单核性能的极限，即其矩阵尺度已经能够充分利用 Cache 大小，批处理矩阵乘法很难在此基础上进一步提升性能。同样深度学习中绝大部分情况下矩阵都为中小尺度矩阵，因此批量大尺度矩阵乘法意义不是很大。

4.2.3 与商业软件对比测试

Intel 的商业软件 MKL 已经实现了批处理的矩阵乘法，ARM 的商业软件 ARM Performance Library（APL）也实现了类似功能。本文在 Intel Xeon Gold 6130 上对比 MKL 进行了数值实验；在 FT2000+和 ThunderX2 上对比 APL 进行了数值实验。在对标 MKL 的实验中，我们采用了两种方式：一种先调用 cblas_sgemm_pack，将矩阵 A 预先分配一块

buffer，然后调用 cblas_sgemm_compute，重复利用矩阵 A 和不同的矩阵 B 进行矩阵计算的方式，简记为 MKL-PACK；另一种是调用 cblas_sgemm_batch 接口方式，简记为 MKL-BATCH。在对标 APL 的实验中，由于其未提供 MKL 类似的 pack 函数接口，我们只采用了 sgemm_batch 批处理函数接口，简记为 APL-BATCH。

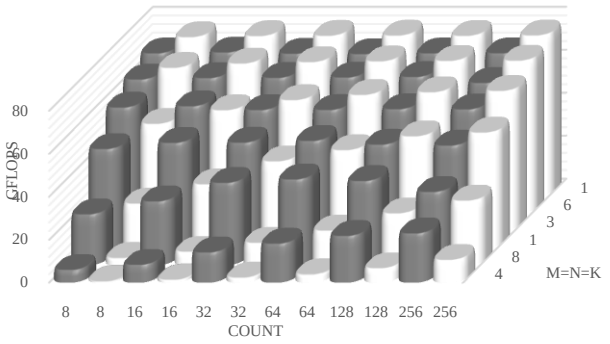

图 25MKL-PACK 对比实验结果

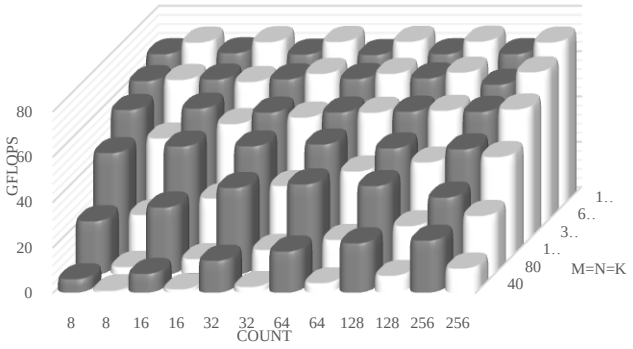

图 26MKL-BATCH 对比实验结果。

图 25 和图 26 分别展示了本文批处理算法与 MKL-PACK 和 MKL-BATCH 的对比性能测试结果。图 27 和图 28 分别展示了本文批处理算法与 APL-BATCH 在 FT2000+和 ThunderX2 平台的对比结果。其中，黑色方柱代表本文提出的批处理单精度矩阵乘法性能，白色方柱代表 MKL 数学库性能。

从图 25 和图 26 中可以看出，对于矩阵尺度较小时本文提出的批处理单精度矩阵乘法相较于 MKL 批处理的两种实现有明显优势，但在矩阵尺度较大时无优势。对于不同尺度规模的矩阵，MKL 有针对性的优化。从图 27 可以看到，随着矩阵尺度的变大，本文提出的批处理单精度矩阵乘法更具优势，总体趋势明显优于 APL 中的批处理实现。从图 28 可见，本文算法与 APL 中批处理算法性能相当。此外，结合图 21 和图 27 我们发现，在 FT2000+平台上 APL 批处理实现几乎没有任何优势，这是因为 APL 没有针对 FT2000+做任何优化。同时，结合图 20 和图 28，我们发现在 ThunderX2 平台上，APL 应该是有一些针对性的优化，APL 的批处理实现展示了一定的优越性。

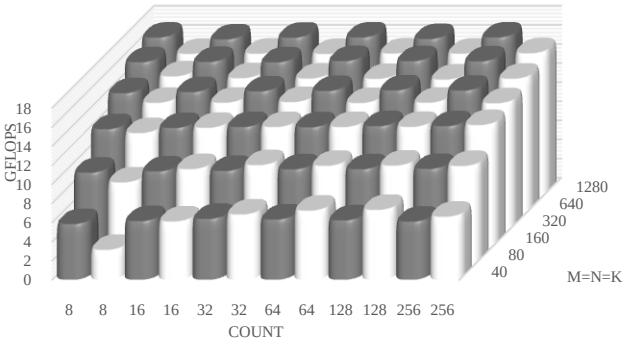

图 27FT2000+平台上与 APL-BATCH 对比实验

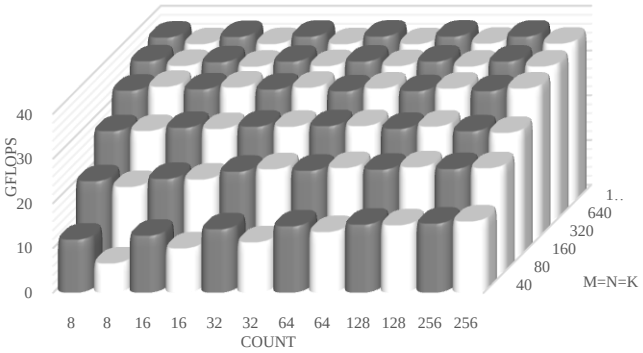

图 28ThunderX2 平台上与 APL-BACTH 对比实验

由于 MKL 以及 APL 都是闭源的，我们无法知其内部的具体优化实现，但对于符合深度学习特征的中小尺度矩阵，本文提出的算法有着不弱于商业软件的性能，且在若干情形下，性能优于商业软件。

4.2.4 针对实际应用场景测试

在深度学习的实际应用中，由于采用的卷积核不同，卷积算法转化为的矩阵乘法时，会产生的不同规格的矩阵。不同的神经网络结构中，矩阵的规模尺度也各不相同。我们统计了 Lenet、AlexNet、GoogleNet 等网络结构前向传播中的矩阵尺寸，并针对这些尺寸的矩阵，测试本文提出的批处理矩阵乘算法，数值结果展示在图 29 中。其中，count=64，“计算平台-batch”代表了本文提出的批处理单精度矩阵乘法性能，“计算平台-for”代表了循环调用单精度矩阵乘法性能，x 轴代表 M, N 和 K 的不同取值，y 轴代表测试性能。可以看出，对于符合深度学习特征的中小尺度矩阵，本文提出的批处理矩

阵乘法比循环调用单精度矩阵乘法有了一定的提升，因此从应用角度上看，本文的批处理算法具有一定优势。

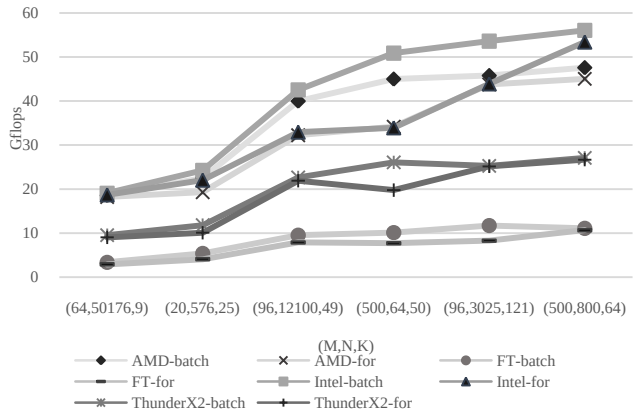

图 29 四个计算平台上对特定尺度矩阵的性能测试

进一步，我们将本文提出的批处理矩阵乘法应用到具体的神经网络训练中，我们修改了神经网络框架 MXNet 的卷积函数实现源码，将卷积过程的前向传播中循环调用单精度矩阵乘法的部分替换成本文提出的批处理单精度矩阵乘法，同时将 Im2col 算法做了批量处理的修改。实验使用 Lenet 网络结构训练 Mnist 手写数字集，通过改变不同的输入 batch_size 大小，记录下不同平台训练网络时的每秒处理的图片数量，结果如表 6 所示，其中 Mx 代表原始的 MXNet，Mx-batch 代表利用批处理矩阵乘法的 MXNet。

表 6 不同平台下应用批处理矩阵乘法的 MXNet 性能对比(单位：images/s)

Batch Size	ThunderX2		FT2000+		AMD-Rzen		Intel-Gold	
	Mx	Mx batch	Mx	Mx batch	Mx	Mx batch	Mx	Mx batch
16	335	340	225	230	430	590	330	325
32	373	385	252	262	460	620	341	342
64	406	412	272	280	485	650	350	354
128	421	425	280	297	495	653	357	366
256	426	432	290	313	502	656	365	373

从表 6 可以看到，相对于原始 MXNet，利用本文提出的批处理单精度矩阵乘法改进 MXNet，机器学习中数据训练过程的性能有了提高，尤其在 AMD 平台上优化性能最为明显，但在其他平台上提升不大。我们经过分析认为：在原始的 MXNet 中，每张图片依次经过 Im2Col 函数和卷积函数处理，Im2Col 函数一定程度上对数据进行了预热，保证了后续卷积实现中单精度矩阵乘法 SGEMM 的性能。改进的 MXNet 可以看成是循环 Im2Col 处理图片后，然后调用批处理 SGEMM，由于 Im2Col 函数原因，性能提升在某些情况不明显。考虑到针对 CPU 的 GEMM 优化过程中，为了充分利用 Cache，往往需要数据重排序 PACKING 过程，我们计划在下一步的工作中将 Im2Col 和 Packing 相融合，实现更好的卷积算法。

5 结论

本文针对深度学习领域中计算量最大的卷积运算展开深入研究，分析了矩阵实现卷积过程中数据的组织排序，根据卷积矩阵乘法的特性，提出了针对多个矩阵和一个矩阵相乘的批处理的矩阵乘法。本文提供了统一框架的批量矩阵乘法接口和实现，能够处理并输出深度学习中任意排序格式的数据。本文基于分块矩阵乘法做数据分割组织优化，没有增加额外数据重组工作。数值实验验证了本文算法的有效性和高效性。

本文的批量矩阵来源于批量的 Im2Col 操作，需要深度学习框架做适当的匹配修改工作。此外，Im2Col 的数据重排可以隐藏在批量矩阵乘法的计算过程中，下一步准备将 Im2Col 和批量矩阵乘法融合在一起，并分析 NHWC 和 NCHW 格式以外的其它格式，设计效率更优的高性能批量卷积操作。从功能上讲，MKL 的 Pack-GEMM 是本文统一框架批量矩阵乘法算法的子集，其仅仅提供列主序或行主序的矩阵乘法，对于混合主序的情况不支持。但是其针对 Intel 体系结构做了更多的底层优化工作，使得其性能优异。下一步准备针对具体平台的体系结构展开更进一步的底层优化，进一步提升本文算法的性能。除了面向深度学习训练过程的单精度批处理矩阵乘法，本文方法还可以扩展到应用于推理阶段的整数批处理矩阵乘法。

致谢 感谢 NASAC 会议审稿人和计算机学报审稿人的宝贵意见。

参考文献

[1] Jia yang-qing. Learning semantic image representations at a large scale. Berkeley: EECS Department, University of California, Technical Report(Phd Thesis): No.UCB/EECS-2014-93, 2014.

[2] Chellapilla K, Puri S, and Simard P. High performance convolutional neural networks for document processing. //Proceedings of the Tenth International Workshop on Frontiers in Handwriting Recognition. La Baule, France, 2006，1-6.

[3] Vasilache N, Johnson J, Mathieu M, Chintala S, PiantinoS, and LeCunY. Fast convolutionalnets with fbfft: A gpu performance evaluation. arXivpreprint arXiv:1412.7580, 2014.

[4] Lavin A and Gray S. Fast algorithms for convolutional neural networks. //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition(CVPR).Las Vegas, US, 2016, 4013–4021.

[5]Liang xiao-yao. Ascend AI Processor Architecture and Programming: Principles and Application of CANN. Beijing: Tsinghua University Press, 2019.(in Chinese)
(梁晓峣.昇腾AI处理器架构与编程: 深入理解CANN技术原理及应用. 北京: 清华大学出版社, 2019.)

[6] Goto K and Van De Geijn R, Anatomy of high-performance matrix Multiplication, ACM Trans on Mathematical Software, 2008, 34(3):1-25.

[7] Goto K and Van De Geijn R, High-performance implementation of the Level-3 BLAS, ACM Trans on Mathematical Software, 2008, 35(1):1-14.

[8] OpenBLAS, http://www.openblas.net.

[9] Abdelfattah A, Costa T, DongarraJ and et al. A Set of Batched Basic Linear Algebra Subprograms and LAPACK Routines. ACM Transactions on Mathematical Software, 2021, 47(3):1-23.

[10] Masliah I, Abdelfattah A, Haidar A and et al. Algorithms and optimization technique for high-performance matrix-matrix multiplications of very small matrices. Parallel Computing, 2019 81:1-21.

[11] Villa O, Fatica M, Gawande N, and Tumeo A. Power/Performance Trade-Offs of Small Batched LU Based Solvers on GPUs.//Proceedings of the International Conference on Parallel Processing (ICPP). Lyon, France, 2013, 813–825.

[12]HaidarA, LuszczekP, Tomov S, and Dongarra J. Towards batched linear solvers onaccelerated hardware platforms. //Proceedings of the 20th ACM SIGPLAN Symposium on Principles andPractice of Parallel Programming (PPoPP), San Francisco, USA, 2015,261-262.

[13] Abdelfattah A, Haidar A, Tomov S, and Dongarra J. Performance, design, and autotuning of batched GEMM for GPUs. //Proceedings of the International Conference on High Performance Computing, Frankfurt, Germany, 2016, 21-38.

[14] cuBLAS. http://developer.nvidia.com/cublas.

[15] BLAS-like extensions in Intel Math Kernel Library (MKL).http://software.intel.com/content/www/us/en/develop/tools/math-kernel-library/linear-algebra.html.

[16] ARM Performance Library (APL). https://www.arm.com/products/development-tools/server-and-hpc/allinea-studio/performance-libraries.

[17] Kim K, Costa T.B., Deveci M and et al. Designing Vector-Friendly Compact BLAS and LAPACK Kernels, //Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, Utah, 2017,55:1-12.

[18] Anderson A, Vasudevan A, Keane C,and Gregg D. Low-memory GEMM-based convolutionalgorithms for deep neural networks. arXiv preprintarXiv:1709.03395, 2017.

[19] Anderson A, Vasudevan A, and Gregg D. Parallel multi channel convolution using general matrix multiplication. //Proceedings of the 28th IEEE international conference on application-specific systems, architectures and processors(ASAP), Seattle, USA, 2017, 19-24.

[20]Cho M and Brand D. Mec: memory-efficient convolutionfor deep neural network. //Proceedings of the 34thInternational Conference on Machine Learning(ICML), Sydney, Australia,2017,815–824.

[21] Dukhan M. The indirect convolution algorithm. arXiv preprint arXiv:1907.02129v1,2019.

HUANGCHUN， Ph.D., professor. Her interest includes optimization and compilation systems.

JIANG Hao, Ph. D., assistant researcher. His research interests include high performance computing, rounding error analysis, numerical computation.

QUAN Zhe, Ph. D., associate professor. His research interests include AI, algorithm and compiler.

ZUO Ke,Ph. D., associate researcher. His research interests include internet and complier.

HE Nan, MS. His research interests include deep learning and algorithm.

LIU Wenchao, MS, assistant engineer. His research interests include high performance computation and computational physics.

Background

Deep neural networks (DNNs) commonly implement convolution operator with GEMM-based algorithm. DNN may require Batched GEMM (general matrix-matrix multiplication) in convolution layer to reuse the convolution kernels. NVIDIA and Intel are already providing some optimized Batched BLAS implementations in CUBLAS and MKL, respectively. However, the original batched GEMM is not very suite to the batched convolution, since DNN only need that one small matrix multiplies multi-matrices.

This article introduces the unified framework of batched GEMMwith the mixed row-column major, which computes the special batched GEMM motivated by DNN's batched convolution. The numerical tests show that our batched GEMM for deep learning run much faster than the original GEMM for some cases.

The purpose of the project is providing theoretical and technicalsupport to build the high-performance numericalalgebra algorithm library. As a basic and common research, the project will greatly enhance the machine learning application.

This research is partly supported by the National Key Research and Development Program of China under Grant2020YFA0709803, 173 Program under Grant 2020-JCJQ-ZD-029, the ScienceChallenge Project under Grant TZ2016002, the National Natural Science Foundation of Hunan under Grant 2018JJ3616.

Our research group has published several related research papers on journals and conferences, such as Applied Mathematics and Computation,Internal Conference on Parallel Processing, Chinese Journal of Computers.