计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200800187

一种基于分布式编码的卷积优化算法

苑晨宇　谢在鹏　朱晓瑞　屈志昊　徐媛媛
河海大学计算机与信息学院　南京 211100
(chenyu_yuan@hhu.edu.cn)

摘　要　卷积在统计学、信号处理、图像处理、深度学习等领域有着广泛的应用,且起到了至关重要的作用。在深度神经网络中,使用卷积运算对输入信息进行特征提取的方法是实现神经网络的基础计算单元之一。如何优化卷积的运算速度,提高卷积计算效率一直是亟需探讨的问题。近年来,很多研究指出分布式计算架构可以提高卷积神经网络的计算速度,进而优化深度学习的训练效率,然而由于分布式系统中普遍存在落跑者问题(straggler),该问题可能会拖慢整个系统执行任务的时间,因此该问题也成为了分布式深度学习中一个待解决的问题。文中针对二维卷积计算,结合 Winograd 算法和分布式编码,提出了一种优化的分布式二维卷积算法。Winograd 算法能够有效地加速单次二维卷积计算的速度,分布式编码通过使用一种基于分布式冗余的编码方式能够缓解 straggler 节点对整个分布式系统计算延迟的影响。因此,提出的分布式二维卷积算法可以在加速二维卷积计算的同时有效缓解分布式系统中的 straggler 问题,有效提高了分布式卷积的计算效率。

关键词:卷积;分布式计算;分布式编码;Winograd
中图法分类号　TP338.8

Convolutional Optimization Algorithm Based on Distributed Coding

YUAN Chen-yu, XIE Zai-peng, ZHU Xiao-rui, QU Zhi-hao and XU Yuan-yuan
School of Computer and Information, Hohai University, Nanjing 211100, China

Abstract　Convolution operation plays a vital role in statistics, signal processing, image processing and deep learning. It is also an important operation in deepneural networks where it is the basis of information filter and characteristics extraction. The exploration of methods to speed up the convolutional operations has become an open research topic in recent years. Many studies have pointed out that distributed computing framework may improve the computational time of convolution operations and hence optimize the training efficiency for deep learning. But stragglers in distributed systems may slow down the overall system. This paper proposes a distributed coding based Winograd algorithm for implementing 2D convolution operations. In this algorithm, the Winograd algorithm can effectively accelerate the speed of 2D convolution calculation, while distributed encoding can mitigate the impact of stragglers by using a redundancy-based encoding strategy. Therefore, the proposed distributed 2D convolution algorithm can effectively mitigate the straggler problem in distributed systems while improving the 2D convolution calculation, hence it may effectively improve the computational efficiency of distributed 2D convolution algorithms.

Keywords　Convolution, Distributed computing, Distributed coding, Winograd

1　引言

卷积是一种积分变换的数学方法,其定义为两个变量在某范围内相乘后求和的结果。卷积需要大量地使用乘法和加法运算,因此占用的资源大,且运算速度慢。例如,FIR[1]滤波算法基本上就是卷积运算,程序的全部执行时间都是卷积的计算时间;卷积神经网络模型中卷积层的计算量约占整个模型计算量的 85% 以上[2]。因此,如果能够加快卷积这一步骤的运算速度,则能够大幅提升数字滤波的速度,进而有效加速如图像处理中的平滑、去噪或者深度学习[3]中的图像特征提取等应用。

随着大数据时代的到来,大数据应用对计算机计算能力以及计算性能的要求也越来越高,从而促进了分布式计算[4]的发展,分布式计算是一种利用多个计算节点进行计算的计算方法。分布式计算由于高效的计算性能而备受关注,本文尝试优化传统的复杂二维卷积运算操作并移植至分布式系统上,使用多个节点协同计算二维卷积以加快卷积的计算速度。但是,分布式系统本身存在的一些问题会影响整个任务的计算效率,例如分布式计算中各节点的通信以及由于等待最慢节点而带来的时延,这些都会对分布式计算产生影响,使得整

到稿日期:2020-08-28　返修日期:2020-11-07　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发课题(2016YFC0402710);国家自然科学基金重点项目(61832005)
This work was supported by the National Key R&D Program of China(2016YFC0402710) and Key Project of National Nature Science Foundation of China(61832005).
通信作者:谢在鹏(zaipengxie@hhu.edu.cn)

个任务的计算效率降低。而一些特殊的分布式编码[5-6]方法可以有效地缩短这些问题所消耗的时间，分布式编码通过在分布式系统中增加计算冗余来实现对分布式系统的优化，这是一种有效的算法优化方法。

本文第2节对前期的相关工作进行了叙述；第3节提出了一种基于分布式编码的卷积优化算法——分布式编码卷积算法(Distributed Coding Convolution Algorithm，DCCA)，并对该方法进行了详细的说明；第4节对本文算法执行卷积运算时的性能进行了实验，并将其与其他常用的算法进行了比较，通过两个应用实例验证了所提算法的性能；最后总结全文并展望未来。

2 相关工作

近年来，有越来越多的研究者开始关注卷积运算的优化算法，并针对使用最为广泛的二维卷积算法开展了大量的研究工作[7-33]。这些工作主要围绕二维卷积算法优化[7-16]和通过分布式编码优化矩阵乘法这两方面来展开[17-33]。

文献[7]通常将各种卷积算法分为两大类：直接卷积算法(如最为常见的一般卷积算法和im2row算法)和间接卷积算法(如FFT或Winograd算法)。文献[8]利用FFT变换和卷积定理来计算二维卷积，即将输入矩阵和卷积核通过FFT转换到频域做乘法后再通过IFFT变换回时域，进而得到卷积后的结果。文献[9]对该方法进行了提炼和优化，最终在NVIDIA cuDNN库中得以实现和应用，但目前更为常用的二维卷积的卷积核尺寸通常都较小，因此在此种情况下，FFT的转换开销会导致该方法的时间开销较大。文献[10]使用Strassen算法[11]来减少二维卷积所需要的乘法计算数量，从而降低二维卷积计算的复杂度，Strassen算法能将矩阵乘法计算的时间复杂度由$O(n^3)$降低到$O(n^{2.81})$，因此该方法对于较大的矩阵拥有更为明显的加速效果。文献[12]提出了一种基于Winograd算法[13]的二维快速卷积算法，通过变换矩阵，将数据映射到另一个实数空间中，进而达到减少乘法运算的目的。文献[14]提出了一种基于Winograd算法的计算三维卷积的方法，并通过实验验证了该算法的性能。文献[15]通过实验验证了使用Winograd算法计算卷积能够降低处理器的能耗。但是，由于Winograd算法本身的特点，导致Winograd算法只适用于尺寸较小的卷积计算，如果计算尺寸较大，则会出现资源消耗增多、精度下降的问题[16]。因此，Winograd算法与分布式计算的结合成为了解决该问题的方法。

随着人工智能和大数据等领域的发展，现代的分布式系统如ApacheSpark[17]和分布式编程框架MapReduce[18]已经在很多领域有所应用。然而，分布式系统的性能受到系统异构性所带来的影响较为显著，文献[19]指出在计算节点上进行运算任务时，通常会由于网络延迟、共享资源和功率限制等造成节点运算任务的不可测延迟，还会造成分布式系统中的straggler(分布式系统中运行速度明显慢于其他节点的节点)问题。文献[20-21]通过异步并行执行的方式来减弱straggler的影响。文献[22-28]通过复制任务和调动副本的方法显著提升了分布式算法的运行速度，缩短了运行时间。另一种降低分布式系统中straggler影响的方法是分布式编码，文献[29]证明将冗余加入到更多的节点中可以降低系统的延迟。文献[30-31]证明编码可以有效利用创建的计算冗余来减弱straggler的影响。文献[30]引入了一种分布式编码方法，使用极大距离可分(Maximum Distance Separable，MDS)码[32]，通过对要进行乘法运算的两个矩阵中的一个矩阵注入计算冗余来达到抑制straggler的作用。文献[33]提出了一种多项式码的编码方式，其相比MDS码拥有更为出色的性能，但此编码方式并不适合于分布式卷积的运算。

上述研究分别是针对卷积或是分布式中加速计算任务方面的改进，然而尚未有研究将二者进行结合，如何将加速二维卷积计算速度的方法应用于分布式卷积计算，在分布式卷积计算中如何解决分布式自身的问题，以及如何改进才能使分布式编码适用于分布式卷积运算，这些都是本文关注的问题。本文提出的基于分布式编码的卷积优化算法将二者相结合，能够有效地加快二维卷积的计算速度。

3 分布式编码卷积算法

目前，二维卷积被大量应用于图像处理、深度学习等热门领域，其主要计算过程如图1所示。其利用卷积核在图像上进行滑动，将图像的像素灰度值与对应的卷积和数值相乘，之后将相乘后的值相加作为卷积的最终结果，滑动遍布全部图形后卷积结束。若将一个二维卷积运算任务交给一个分布式系统，使用多个节点共同完成此任务，那么就能够有效提升二维卷积的计算效率，其计算过程如图2所示。

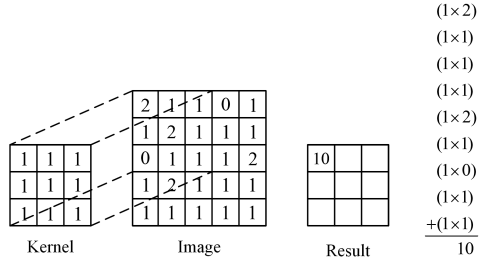

图1 二维卷积计算原理

Fig. 1 Principle of two dimensional convolution

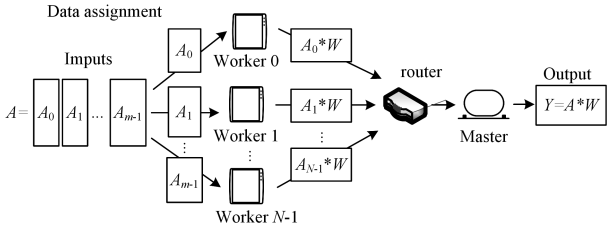

图2 二维分布式卷积计算的原理

Fig. 2 Principle of two dimensional convolution on distributed system

由图2可看出，若要加快此计算过程中的任务执行速度，可以从分节点计算二维卷积的方法和主节点对原始输入矩阵的处理方式两个方面来进行优化。

本文算法是一种优化二维卷积计算的算法，其流程如图3所示。本文算法中的主节点主要负责矩阵的分割、编码、解码和拼接部分，其中分为编码进程和解码进程这两个进程。编码进程在得到输入矩阵后首先对输入矩阵进行填充，以使输入矩阵能够被整分为若干个分矩阵，填充后对矩阵进行分割和编码，将编码后的矩阵发送给各分节点。解码进程在收到足够分节点返回的计算结果后开始解码和拼接矩阵，进而

得到最终的卷积计算结果。分节点主要负责使用 Winograd 算法来计算卷积，在接收到主节点发送来的编码矩阵后，使用 Winograd 算法计算矩阵，并将最终的计算结果返回主节点。

算法 1 描述了本文提出的分布式卷积优化算法，其中输入矩阵为 $\boldsymbol{A}$，卷积核为 $\boldsymbol{W}$，输出卷积结果为 $\boldsymbol{Y}$。算法 1 给出了分布式卷积算法的整体流程，其中包含了算法 2(矩阵分割)、算法 3(分矩阵编码)和算法 4(分节点卷积计算)。

算法 1 Distributed Coding Convolution Algorithm

Input：Input matrix **A**

Output：Output result **Y**

1. Initialize master and workers；
2. Divide **A** into pieces using Algorithm 2；
3. Encode matrixes using Algorithm 3 and each worker i gets data $\mathbf{A}_i$；
4. For each worker i in parallel，compute convolution result $\mathbf{Y}_i$ using Algorithm 4；
5. Master receives $\mathbf{Y}_i$ from each worker；
6. Decode and get result **Y**；
7. return **Y**.

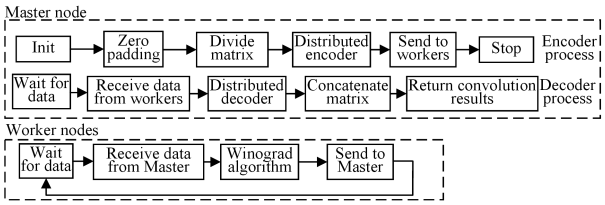

图 3 分布式编码卷积算法流程图

Fig. 3 Flow chart of DCCA

3.1 矩阵分割

本矩阵分割的逻辑如算法 2 所示。

算法 2 Matrix division

Input：Input matrix **D**

Output：Output result $\mathbf{D}_n$

1. Initialize row and col to the same value of the row and column of matrix **D**；
2. Initialize R and C by the value of row and col ；//Times that matrix **D** need to be divided
3. for i=1，2，…，R do
4. Calculate start and end row number using i；
5. for j=1，2，…，C do
6. Calculate start and end column number using j；
7. $\mathbf{D}_n$←divide(**D**)；
8. return $\mathbf{D}_n$.

算法 2 的输入为矩阵 $\boldsymbol{D}$，输出为经过矩阵分割后的 $\boldsymbol{D}_n$，子函数 *divide*($\boldsymbol{D}$)根据循环中计算的开始和结束行列数，将矩阵 $\boldsymbol{D}$ 分割成若干个分矩阵，并最终将分矩阵返回。

算法 2 中，当主节点得到输入矩阵后，首先根据分节点的个数确定分矩阵的个数，然后结合输入矩阵大小和分节点个数确定分矩阵的大小，最后按照确定后的值进行矩阵分割。

首先讨论输入矩阵的行数和列数可以整分的情况。假设系统的输入为 6×6 大小的矩阵，用 $\boldsymbol{D}$ 表示，卷积核尺寸为 3×3，用 $\boldsymbol{W}$ 表示。

$$\boldsymbol{D}=\begin{bmatrix} d_0 & d_1 & \cdots & d_5 \\ d_6 & d_7 & \cdots & d_{11} \\ \vdots & \vdots & \ddots & \vdots \\ d_{30} & d_{31} & \cdots & d_{35} \end{bmatrix},\boldsymbol{W}=\begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{bmatrix}$$

此时，若要通过多个节点协同完成矩阵的二维卷积运算，那么就要将矩阵 $\boldsymbol{D}$ 分割成若干个分矩阵，每个分节点负责计算一部分的卷积结果。将矩阵 $\boldsymbol{D}$ 分割为大小为 4×4 的分矩阵 $\boldsymbol{D}_n$，其分割结果如下：

$$\boldsymbol{D}_0=\begin{bmatrix} d_0 & d_1 & d_2 & d_3 \\ d_6 & d_7 & d_8 & d_9 \\ d_{12} & d_{13} & d_{14} & d_{15} \\ d_{18} & d_{19} & d_{20} & d_{21} \end{bmatrix},\boldsymbol{D}_1=\begin{bmatrix} d_2 & d_3 & d_4 & d_5 \\ d_8 & d_9 & d_{10} & d_{11} \\ d_{14} & d_{15} & d_{16} & d_{17} \\ d_{20} & d_{21} & d_{22} & d_{23} \end{bmatrix}$$

$$\boldsymbol{D}_2=\begin{bmatrix} d_{12} & d_{13} & d_{14} & d_{15} \\ d_{18} & d_{19} & d_{20} & d_{21} \\ d_{24} & d_{25} & d_{26} & d_{27} \\ d_{30} & d_{31} & d_{32} & d_{33} \end{bmatrix},\boldsymbol{D}_3=\begin{bmatrix} d_{14} & d_{15} & d_{16} & d_{17} \\ d_{20} & d_{21} & d_{22} & d_{23} \\ d_{26} & d_{27} & d_{28} & d_{29} \\ d_{32} & d_{33} & d_{34} & d_{35} \end{bmatrix}$$

下面，讨论输入行数和列数不可整分的情况。假设卷积核尺寸不变，为 3×3，输入矩阵 $\boldsymbol{D}$ 的大小为 5×5。

$$\boldsymbol{D}=\begin{bmatrix} d_0 & d_1 & \cdots & d_4 \\ d_5 & d_6 & \cdots & d_9 \\ \vdots & \vdots & \ddots & \vdots \\ d_{20} & d_{21} & \cdots & d_{24} \end{bmatrix},\boldsymbol{W}=\begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{bmatrix}$$

此时，若要对输入矩阵进行分割，则需要先将行数与列数补为偶数个，最为简单有效的方法是用 0 将矩阵填充为 6×6 大小的矩阵，再将填充后的矩阵进行分割，其分割结果如下：

$$\boldsymbol{D}_0=\begin{bmatrix} d_0 & d_1 & d_2 & d_3 \\ d_5 & d_6 & d_7 & d_8 \\ d_{10} & d_{11} & d_{12} & d_{13} \\ d_{15} & d_{16} & d_{17} & d_{18} \end{bmatrix},\boldsymbol{D}_1=\begin{bmatrix} d_2 & d_3 & d_4 & 0 \\ d_7 & d_8 & d_9 & 0 \\ d_{12} & d_{13} & d_{14} & 0 \\ d_{17} & d_{18} & d_{19} & 0 \end{bmatrix}$$

$$\boldsymbol{D}_2=\begin{bmatrix} d_{10} & d_{11} & d_{12} & d_{13} \\ d_{15} & d_{16} & d_{17} & d_{18} \\ d_{20} & d_{21} & d_{22} & d_{23} \\ 0 & 0 & 0 & 0 \end{bmatrix},\boldsymbol{D}_3=\begin{bmatrix} d_{12} & d_{13} & d_{14} & 0 \\ d_{17} & d_{18} & d_{19} & 0 \\ d_{22} & d_{23} & d_{24} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

将输入矩阵经过上述步骤分割成若干个分矩阵后，便可较为方便地发送给多个节点来共同完成矩阵乘法的计算任务，为接下来的编码做准备。

3.2 编码

编码策略的执行逻辑如算法 3 所示。

算法 3 Distributed coding

Input：Input matrix $\mathbf{D}_n$

Output：Output result $\widetilde{\mathbf{D}}_i$

1. Initializeall $\widetilde{\mathbf{D}}_i$ as empty；
2. Initialize i as the number of workers；
3. Initialize n as the number of matrix $\mathbf{D}_n$；
4. for j=0，1，…，i−1 do
5. for k=0，1，…，n−1 do
6. $\mathbf{D}_k'$←multiply(j，k，$\mathbf{D}_n$)；
7. $\widetilde{\mathbf{D}}_i$←add($\mathbf{D}_k'$)；
8. return $\widetilde{\mathbf{D}}_i$.

算法 3 的输入为经算法 2 分解后所得的所有分矩阵 $\boldsymbol{D}_n$，输出为经过编码后的需要发送至分节点 i 进行进一步卷积计

算的编码矩阵$\widetilde{\boldsymbol{D}}_i$，子函数 $multiply(j,k,\boldsymbol{D}_n)$和 $add(\boldsymbol{D}_k')$共同完成对输入分矩阵的编码工作并最终获得编码矩阵。

以注入冗余的形式对分矩阵进行编码，可以在保证数据发送量不变的前提下丰富每个矩阵所携带的信息，那么在主节点接收分节点返回的计算结果时，往往不需要等待所有分节点全部返回结果，在部分结果返回的情况下便可通过解码恢复最终的卷积计算结果。如此，当分布式系统中出现straggler时，主节点不需要等待其返回结果，整个系统的任务执行时间会得到有效的提升。本文对系统中 straggler 节点的出现频率也进行了测试，选取了拥有 10 个分节点的分布式系统，每个分节点执行 500 次卷积运算，主节点负责记录每个分节点每次进行矩阵发送、接收和计算的时间。图 4 给出了各分节点计算卷积时间的累计概率分布图。

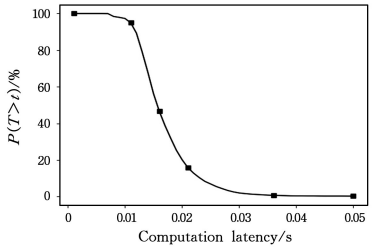

图 4 straggler 节点出现的概率

Fig. 4 Probability of straggler's appearance

图 4 中，横坐标为计算延迟，纵坐标为累计概率，曲线上的点对应的纵坐标值表示在 5000 个计算延迟数据中，大于该点所对应横坐标值的数据个数占全部数据个数的比例。从图 4 可以看出，分节点从接收主节点发送来的矩阵到计算卷积结果，再到将卷积结果传送回主节点的平均时间为 0.015 s 左右，而后 5% 的卷积任务的计算时间都超过了 0.015 s 的 1.5 倍，因此每个节点在每次计算中有 5% 的概率成为 straggler，那么在一个拥有 10 个分节点的分布式系统中，每次计算都不出现 straggler 的概率只有 60%。该结果充分说明了抑制 straggler 对于一个分布式系统的重要性。

按照算法 3，本节将给出编码的详细步骤。下面给出一个简单的例子，假设一个分布式系统拥有 1 个主节点和 5 个分节点来完成计算任务，系统初始接收到一个大小为 6×6 的输入矩阵 $\boldsymbol{D}$，并被要求计算其与大小为 3×3 的卷积核 $\boldsymbol{W}$ 的二维卷积结果。

经过算法 2 的矩阵分割后，可得到 4 个大小为 4×4 的分矩阵，分别记作$\boldsymbol{D}_0$，$\boldsymbol{D}_1$，$\boldsymbol{D}_2$和$\boldsymbol{D}_3$，之后对分矩阵进行编码，用$\widetilde{\boldsymbol{D}}_i$表示编码后的矩阵，其中 $i\in\{0,1,2,3,4\}$对应每个分节点的序号，其编码方法如式(1)所示：

$$\widetilde{\boldsymbol{D}}_i=\boldsymbol{D}_0+i\boldsymbol{D}_1+i^2\boldsymbol{D}_2+i^3\boldsymbol{D}_3 \tag{1}$$

完成分矩阵的编码后，可将编码后的矩阵一一对应地发送至分节点处进行二维卷积运算，卷积结果用$\widetilde{\boldsymbol{Y}}_i$表示，其计算式如式(2)所示。此处的二维卷积运算方法也是经过优化后的二维卷积运算方法，具体执行逻辑将在 3.3 节给出。

$$\begin{aligned}\widetilde{\boldsymbol{Y}}_i&=\widetilde{\boldsymbol{D}}_i*\boldsymbol{W}\\&=\boldsymbol{D}_0*\boldsymbol{W}+i(\boldsymbol{D}_1*\boldsymbol{W})+i^2(\boldsymbol{D}_2*\boldsymbol{W})+i^3(\boldsymbol{D}_3*\boldsymbol{W})\end{aligned} \tag{2}$$

各分节点完成卷积计算后，将计算结果$\widetilde{Y}_i$返回至主节点，此时主节点只需要收到 5 个分节点中的任意 4 个分节点返回值便可通过解码恢复最终的卷积结果。本文实验中使用的解码方式为：计算式(3)中范德蒙矩阵的逆矩阵后，将其与分节点返回结果做矩阵乘法运算，最终所得结果即为卷积结果。

$$\begin{bmatrix}\widetilde{\boldsymbol{Y}}_1\\\widetilde{\boldsymbol{Y}}_2\\\widetilde{\boldsymbol{Y}}_3\\\widetilde{\boldsymbol{Y}}_4\end{bmatrix}=\begin{bmatrix}1^0&1^1&1^2&1^3\\2^0&2^1&2^2&2^3\\3^0&3^1&3^2&3^3\\4^0&4^1&4^2&4^3\end{bmatrix}\begin{bmatrix}\boldsymbol{D}_0*\boldsymbol{W}\\\boldsymbol{D}_1*\boldsymbol{W}\\\boldsymbol{D}_2*\boldsymbol{W}\\\boldsymbol{D}_3*\boldsymbol{W}\end{bmatrix} \tag{3}$$

将分布式编码的方法进行推广，便可得到一种一般的分布式编码方法，假设输入矩阵 $\boldsymbol{D}$ 的尺寸为 $m\times n$(为表示方便，假设 m，n 都为偶数)，卷积核 $\boldsymbol{W}$ 的尺寸为 3×3，分节点个数为 N，每个分节点的序号 i 用变量x_i代替，则输入矩阵可分割成如下形式：

$$\boldsymbol{D}=\begin{bmatrix}\boldsymbol{D}_{0,0}&\boldsymbol{D}_{0,1}&\cdots&\boldsymbol{D}_{0,\frac{n-2}{2}-1}\\\boldsymbol{D}_{1,0}&\boldsymbol{D}_{1,1}&\cdots&\boldsymbol{D}_{1,\frac{n-2}{2}-1}\\\vdots&\vdots&\ddots&\vdots\\\boldsymbol{D}_{\frac{m-2}{2}-1,0}&\boldsymbol{D}_{\frac{m-2}{2}-1,1}&\cdots&\boldsymbol{D}_{\frac{m-2}{2}-1,\frac{n-2}{2}-1}\end{bmatrix}$$

分割后矩阵的编码如式(4)所示：

$$\widetilde{\boldsymbol{D}}_i=\sum_{j=0}^{\frac{n-2}{2}-1}\sum_{k=0}^{\frac{m-2}{2}-1}\boldsymbol{D}_{j,k}x_i^{k(\frac{n-2}{2}-1)+j} \tag{4}$$

将编码后的分矩阵发送至对应的分节点，在分节点完成卷积运算后将结果发送回主节点，主节点在接收到 l 个分节点的返回值后便可通过解码恢复最终的结果。

$$l=\frac{m-2}{2}\cdot\frac{n-2}{2} \tag{5}$$

3.3 分节点卷积运算

节点卷积运算的执行逻辑如算法 4 所示。

算法 4 Convolution based on Winograd algorithm

Input: Input matrix $\widetilde{\mathbf{D}}_i$

Output: Output result $\widetilde{\mathbf{Y}}_i$

1. Initialize $\widetilde{\mathbf{Y}}_i$ as empty;
2. Initialize $\mathbf{A}^T$, $\mathbf{G}$, $\mathbf{B}^T$ as transformation matrix;
3. Initialize $\mathbf{W}$ as the kernel;
4. $\mathbf{U}$←transformation($\mathbf{G}$, $\mathbf{W}$);
5. V←transformation($\mathbf{B}^T$, $\widetilde{\mathbf{D}}_i$)
6. M←dot_product($\mathbf{U}$, $\mathbf{V}$);
7. $\widetilde{\mathbf{Y}}_i$←transformation($\mathbf{A}^T$, $\mathbf{M}$);
8. return $\widetilde{\mathbf{Y}}_i$.

算法 4 的输入为经过编码后的矩阵$\widetilde{\boldsymbol{D}}_i$，输出为经过 Winograd 卷积算法计算出的二维卷积结果$\widetilde{\boldsymbol{Y}}_i$，子函数 $transformation(\boldsymbol{A},\boldsymbol{B})$实现矩阵转换的功能，子函数 $dot_product(\boldsymbol{U},\boldsymbol{V})$实现将转换后的矩阵进行点乘操作的功能，两个函数共同完成 Winograd 卷积计算，最终获得每个分节点的卷积计算结果$\widetilde{\boldsymbol{Y}}_i$并返回主节点。

对分矩阵进行编码后，需要将编码后的矩阵一一对应地发送至分节点并在分节点处进行二维卷积运算，但与一般情况不同的是，此时分节点计算二维卷积的方式不再采用一般的滑动卷积方式，而是使用 Winograd 算法计算二维卷积的结

果。算法4首先对编码矩阵和卷积核进行转换，之后进行矩阵乘法以获取最后的结果，使用此种方法计算二维卷积可以减少计算二维卷积时所需的乘法次数，加快二维卷积的速度。

下面讨论使用Winograd算法计算二维卷积的详细步骤。首先对一维Winograd算法进行说明，因为二维Winograd算法是对一维Winograd算法进行扩展和应用的结果。将一个输出结果的向量长度设为 m，卷积核长度为 r 的一维卷积运算记为 $F(m,r)$，使用一般方法计算此卷积需要的乘法次数为 $m\times r$，但若使用Winograd算法，只需要 $m+r-1$ 次乘法运算便可获得最终的卷积结果。

以 $F(2,3)$ 为例，系统的输入矩阵用 $\boldsymbol{D}$ 表示，卷积核用 $\boldsymbol{W}$ 表示，使用Winograd算法计算 $F(2,3)$ 的表达式如下：

$$F(2,3)=\begin{bmatrix}d_0 & d_1 & d_2\\ d_1 & d_2 & d_3\end{bmatrix}\begin{bmatrix}w_0\\ w_1\\ w_2\end{bmatrix}=\begin{bmatrix}m_1+m_2+m_3\\ m_2-m_3-m_4\end{bmatrix} \tag{6}$$

其中，m_1，m_2，m_3 和 m_4 可分别表示为：

$$m_1=(d_0-d_2)w_0,\ m_2=(d_1+d_2)\frac{w_0+w_1+w_2}{2}$$

$$m_4=(d_2-d_1)w_2,\ m_3=(d_2-d_1)\frac{w_0-w_1+w_2}{2}$$

接下来将整个运算过程用矩阵的形式进行表示，可得到式(7)所示的形式：

$$\boldsymbol{Y}=A^{\mathrm{T}}[(\boldsymbol{GW})\odot(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{D})] \tag{7}$$

其中，$\boldsymbol{A}^{\mathrm{T}}$，$\boldsymbol{G}$ 和 $\boldsymbol{B}^{\mathrm{T}}$ 为转换矩阵，其元素组成随着 m 和 r 的变化而变化，其推导需结合中国剩余定理[34]进行计算，其与本文内容关联不大，在此不再深入讨论。

$$\boldsymbol{G}=\begin{bmatrix}1 & 0 & 0\\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2}\\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2}\\ 0 & 0 & 1\end{bmatrix},\boldsymbol{B}^{\mathrm{T}}=\begin{bmatrix}1 & 0 & -1 & 0\\ 0 & 1 & 1 & 0\\ 0 & -1 & 1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}$$

$$\boldsymbol{A}^{\mathrm{T}}=\begin{bmatrix}1 & 1 & 1 & 0\\ 0 & 1 & -1 & -1\end{bmatrix}$$

通过此种方法计算 $F(2,3)$ 只需要4次乘法运算便可获得最终结果。下面将一维Winograd算法拓展至二维，将 $F(2\times2,3\times3)$ 表示成矩阵相乘的形式并进行合并，表达式如式(8)所示：

$$\begin{bmatrix}\boldsymbol{Y}_0\\ \boldsymbol{Y}_1\end{bmatrix}=\begin{bmatrix}\boldsymbol{D}_0 & \boldsymbol{D}_1 & \boldsymbol{D}_2\\ \boldsymbol{D}_1 & \boldsymbol{D}_2 & \boldsymbol{D}_3\end{bmatrix}\begin{bmatrix}\boldsymbol{W}_0\\ \boldsymbol{W}_1\\ \boldsymbol{W}_2\end{bmatrix}=\begin{bmatrix}\boldsymbol{D}_0\boldsymbol{W}_0+\boldsymbol{D}_1\boldsymbol{W}_1+\boldsymbol{D}_2\boldsymbol{W}_2\\ \boldsymbol{D}_1\boldsymbol{W}_0+\boldsymbol{D}_2\boldsymbol{W}_1+\boldsymbol{D}_3\boldsymbol{W}_2\end{bmatrix} \tag{8}$$

从对一维Winograd算法的推导过程中可以看出，每一个 $\boldsymbol{D}_n\boldsymbol{W}_n$ 都是一个 $F(2,3)$ 运算，因此可以结合式(7)推导出二维Winograd算法计算卷积的公式，如式(9)所示：

$$\begin{aligned}\boldsymbol{Y}&=\begin{bmatrix}\sum_{i=0}^{2}\boldsymbol{A}^{\mathrm{T}}[(\boldsymbol{GW}_i)\odot(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}_i)]\\ \sum_{i=0}^{2}\boldsymbol{A}^{\mathrm{T}}[(G\,W_i)\odot(B^{\mathrm{T}}D_{i+1})]\end{bmatrix}\\ &=\boldsymbol{A}^{\mathrm{T}}[[\boldsymbol{GWG}^{\mathrm{T}}]\odot[\boldsymbol{B}^{\mathrm{T}}\boldsymbol{DB}]]\boldsymbol{A}\end{aligned} \tag{9}$$

其中，转换矩阵与 $F(m,r)$ 相同，当 m 和 r 的值发生变化时，转换矩阵也需变化，使用直接卷积的方法和使用Winograd算法所需的乘法次数比较结果如表1所列。

表1 不同卷积核尺寸的Winograd算法比较

Table 1 Comparison of Winograd algorithm with different kernel sizes

Winograd	传统乘法次数	Winograd乘法次数	乘法加速比
$F(2,3)$	6	4	1.50
$F(4,3)$	12	6	2.00
$F(6,3)$	18	8	2.25
$F(2\times2,3\times3)$	36	16	2.25
$F(4\times4,3\times3)$	144	36	4.00
$F(6\times6,3\times3)$	324	64	5.06

需要注意，随着 $F(m,r)$ 中 m 的增大，Winograd算法的额外开销会变得越来越大，直到超出算法本身所带来的速度提升，且计算精度会下降[16]，因此Winograd算法在小尺寸卷积运算时往往效果更好。

分节点通过二维Winograd算法计算出卷积结果后，将其结果发送至主节点，当主节点接收到足够数量的返回结果后(无须等待所有分节点返回计算结果)便可通过解码的方式恢复最终的计算结果。

3.4 算法分析

本节将从3个指标衡量分布式卷积优化算法的性能，分别为恢复阈值、计算延迟和通信负载。

在恢复阈值方面，对于一个一般的分布式卷积运算，假设主节点最终得到的结果 $\boldsymbol{Y}$ 是一个尺寸为 $x\times y$ 的矩阵，其中 $\boldsymbol{Y}\in F_q^{x\times y}$（$F_q$ 为一个足够大的域），那么主节点就需要恢复一个熵为 $H(\boldsymbol{Y})$ bit的随机变量，主节点的结果需要从分节点获取。对于每一个分节点来说，其在计算后会返回给主节点 F_q 中的 k 个元素，即提供 $k\log_2 q$ bit的信息，将需要返回结果的分节点数量记为 l，对数量 l 的求解过程如下：

$$\begin{cases}H(Y)=xy\cdot\log_2 q\\ k=\dfrac{xy}{(m-2)/2\cdot(n-2)/2}\\ l=\dfrac{H(Y)}{k\log_2 q}=\dfrac{m-2}{2}\cdot\dfrac{n-2}{2}\end{cases} \tag{10}$$

可以看出，最少需要 l 个分节点返回其计算结果，主节点才能恢复最终的卷积结果，而分布式编码卷积算法恢复最终结果所需的返回节点数与此计算结果相同，因此达到了恢复最终结果的最小要求。

在计算延迟方面，由于分布式编码卷积算法中每个分节点都使用Winograd算法计算卷积，因此每个分节点计算卷积的速度快于同等条件下用一般方法计算相同规模卷积的速度，因此，从理论上来说，分布式编码卷积算法的计算延迟应小于一般分布式卷积算法的计算延迟。

在通信负载方面，在本文所提到的相同的矩阵分割方法和卷积核尺寸条件下，假设传统方法计算一个结果为 $x\times y$ 尺寸大小的卷积所需要的通信负载最小为 a bit，分布式编码卷积算法计算得到结果所需的最小通信负载为 b bit，两者可表示为：

$$\begin{cases}a=4xy\log_2 q\\ b=4(m-2)(n-2)\log_2 q\end{cases} \tag{11}$$

通过比较发现 a 和 b 相等，因此分布式编码卷积算法在接收分节点结果时达到了使用分布式方法计算卷积的最小通信

负载，但由于主节点向分节点分发编码矩阵的过程中，需要向额外的分节点发送编码矩阵，因此在矩阵分发时通信负载会有提升，主节点需要多发送$\frac{(x+2)(y+2)}{(m-2)/2\cdot(n-2)/2}\log_2 q$ bit 的数据。

4 实验

4.1 实验方法

本文进行了 3 个实验：1)测试 Winograd 算法对卷积计算的加速效果；2)测试分布式编码对分布式卷积计算的加速效果；3)将分布式编码卷积算法应用到卷积神经网络(CNN)中，使用 CNN 对 MNIST 数据集进行训练，并分别与其他 3 种算法进行比较。

实验中使用 Pycharm 作为 Python 的集成开发环境，每个节点的配置信息如表 2 所列。

表 2 计算节点的配置信息

Table 2 Configuration settings of computing node

项目	配置
CPU	Intel Xeon X5675
内存/GB	1
操作系统	Ubuntu14.04
Python 版本	3.7.3

实验 1 中，分别使用 Winograd 算法和传统卷积算法计算 2500 次 $F(2\times2,3\times3)$ 的卷积运算，并记录其每次的计算时间，将时间汇总后以概率分布图的形式展现结果。

实验 2 中，为了验证分布式编码的有效性，将进行两个实验，第一个实验将分别使用分布式编码卷积算法(DCCA)和分布式 Winograd 卷积算法(Distributed Winograd Convolution Algorithm，DWCA)进行图像处理中的高斯滤波，其中 DWCA 即为在分节点计算时使用 Winograd 算法代替传统卷积算法，且不使用分布式编码。在计算高斯滤波的过程中人为引入一个 straggler 节点，此时系统由 11 个节点组成，包括 1 个主节点和 10 个分节点，每个分节点的任务为接收主节点发送来的矩阵，使用不同的方法计算卷积结果并返回至主节点。主节点负责矩阵生成、分割、编码、时间统计等一系列任务，最终将时间汇总后以概率分布图的形式展现。第二个实验将测试不同数量的慢节点个数情况下的系统性能，此时系统由 11 个或 12 个节点组成，即有 10%和 20%的冗余度，主节点分别记录使用不同分节点个数计算高斯滤波的时间，汇总后以概率分布图的形式展现。

实验 3 将分布式编码卷积算法应用至 CNN 的卷积层，即卷积层的卷积部分使用分布式编码卷积算法(DCCA)进行计算，经卷积层提取图像特征后，将特征送入全连接网络进行训练；同时实验中分别使用了其他 3 种方法来替换卷积层的卷积计算，分别为分布式 FFT 卷积算法(Distributed FFT Convolution Algorithm，DFCA)、分布式 Winograd 卷积算法(DWCA)和分布式卷积算法(Distributed Convolution Algorithm，DCA)。分布式 FFT 卷积算法(DFCA)即为分节点计算时使用 FFT 代替传统卷积算法，且不使用分布式编码；分布式卷积算法(DCA)即为使用分布式计算和传统卷积算法计算二维卷积。主节点负责收集每种方法对图片进行特征提取的时间和训练时间，最终将数据汇总并以图的方式展现。

4.2 实验结果

实验 1 共使用 2500 个随机生成的矩阵，分别使用 Winograd 算法和传统算法计算卷积，将其计算时间汇总后绘制出图 5 所示的坐标图。

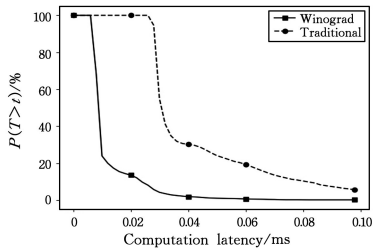

图 5 Winograd 算法与传统算法计算卷积的计算延迟比较

Fig. 5 Computational delay comparison of Winograd algorithm and traditional algorithm when computing convolution

图 5 中，横坐标为计算延迟，纵坐标为 Winograd 算法和传统算法计算延迟的累计概率，曲线上的点对应的纵坐标值表示在所有 2500 个计算延迟数据中，大于该点所对应横坐标值的数据占全部数据个数的比例。从图 5 可以明显地看出，使用 Winograd 算法计算卷积的性能优于传统算法，根据实验数据，使用 Winograd 算法计算 2500 次二维卷积时，每次卷积的平均时间为 1.23×10^{-5} s，而使用传统方法计算二维卷积的平均时间为 4.57×10^{-5} s。因此，Winograd 算法相比传统算法在计算二维卷积的时间消耗方面，性能提升了 73.0%。

实验 2 包含两个实验，第一个实验使用了两种方法计算二维卷积，分别是分布式 Winograd 卷积算法(DWCA)和本文提出的分布式编码卷积算法(DCCA)。该实验总共使用了 11 个节点，主节点记录了两种算法每一轮的计算时间并绘制了累计概率分布图，如图 6 所示。第二个实验改变分布式集群中的冗余度，分别使用 10%和 20%冗余度的分布式集群来计算高斯滤波，主节点记录了两种算法每一轮的计算时间并绘制了累计概率分布图，如图 7 所示。其中，纵坐标为冗余度为 10%和 20%的分布式集群使用分布式编码卷积算法(DCCA)计算延迟的累计概率，曲线上的点对应的纵坐标值表示在计算延迟数据中，大于该点所对应横坐标值的数据个数占全部数据个数的比例。

从图 6 中可以看出，人为增加 straggler 后，使用 DCCA 计算分布式卷积的分布式集群基本不受影响，而使用 DWCA 的分布式集群的性能因为 straggler 的存在而变差。DCCA 计算高斯滤波的平均时间为 0.0450 s，而 DWCA 计算高斯滤波的平均时间为 0.0756 s。从图 7 可以看出，拥有 20%冗余度的分布式集群进行卷积运算的速度快于只拥有 10%冗余度的分布式集群，因为分布式集群规模不大，多个节点同时为 straggler 的概率不高，又因通信和编码计算的增加，20%冗余度的分布式集群在性能上未大幅度提升，但仍然有小的性能提升。从实验 2 中的两个实验可以看出，由于分布式编码的存在，在 straggler 出现时，系统无须等待 straggler 的返回结果也可通过解码获得最终结果，这就使得整个分布式系统的性能得到了增强。通过此实验可以验证分布式编码卷积算法的有效性。

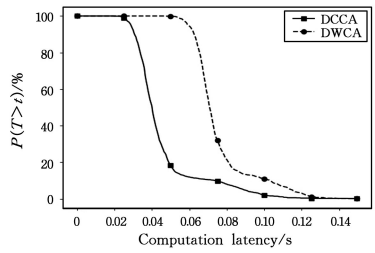

图 6 DCCA 算法与 DWCA 算法计算高斯去噪的计算延迟比较

Fig. 6 Computational delay comparison of DCCA and DWCA algorithm when computing gaussian filtering

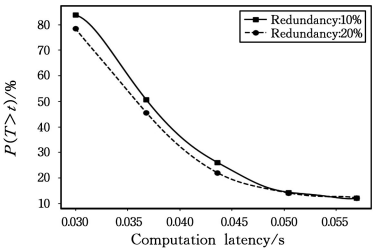

图 7 不同冗余度时 DCCA 算法计算高斯去噪的计算延迟比较

Fig. 7 Computational delay comparison of DCCA with different redundancy when computing gaussian filtering

实验 3 使用 4 种方法进行了 CNN 卷积层中的卷积计算，并通过 CNN 对 MNIST 数据集进行了训练。第 1 种方法为本文提出的分布式编码卷积算法；第 2 种方法为分布式 Winograd 卷积算法；第 3 种方法为分布式 FFT 卷积算法，计算卷积时首先对原图和卷积核进行二维傅里叶变换，将变换后的结果相乘再进行傅里叶逆变换，从而得到最终的二维卷积结果；第 4 种方法为传统分布式卷积算法。该实验共使用 11 个节点，主节点汇总每种方法处理每张图片所使用的时间并绘制出累计概率分布图，如图 8 所示。其中，横坐标为每张图片在不同方法下通过 CNN 卷积层的计算延迟，纵坐标为分布式编码卷积算法(DCCA)及其他算法计算卷积层的卷积计算延迟的累计概率，曲线上的点对应的纵坐标值表示在每种卷积计算方法的所有计算延迟中，大于该点所对应横坐标值的数据个数占全部数据个数的比例。

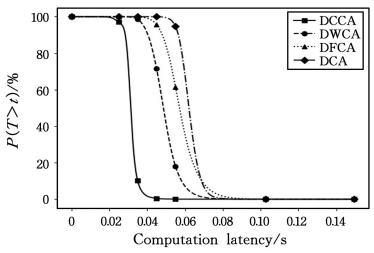

图 8 分布式编码卷积算法与其他算法计算卷积层卷积的计算延迟比较

Fig. 8 Comparison of DCCA and other algorithms when computing convolution inconvolutional layer

从图 8 可以较为明显地看出，分布式编码卷积算法(DCCA)的性能最佳，分布式 Winograd 卷积算法(DWCA)其次，分布式 FFT 卷积算法(DFCA)的速度相对较慢。经主节点统计数据，使用 DCCA 时，每张手写数字图片通过卷积层的平均时间为 0.0316 s，使用分布式 Winograd 卷积算法(DWCA)的平均时间为 0.0491 s，使用分布式 FFT 卷积算法(DWCA)的平均时间为 0.0580 s，使用分布式卷积算法(DCA)的平均时间为 0.0624 s。由此来看，分布式编码卷积算法相比其他 3 种算法，性能分别提升了 35.6%，45.5%和 49.3%。

主节点分别记录了使用 4 种算法的 CNN 进行训练且训练准确度达到 98%时的总用时，如表 3 所列。

表 3 卷积神经网络的训练时间比较

Table 3 Comparison on training time of CNN

算法	总用时/s
DCCA	1973.73
DWCA	3035.81
DFCA	3566.11
DCA	3824.29

从表 3 可以更为直观地看出 4 种算法之间的性能差距，DCCA 的用时小于其他 3 种算法。此实验首先验证了分布式编码卷积算法(DCCA)与目前较为热门的深度学习相结合的可能性，也更为直观地展现了该算法提升计算性能的能力，证实了 DCCA 的有效性，为该算法的进一步应用提供了更为广阔的前景。为了进一步验证本文算法在集群规模变化的情况下的性能，我们还测试了 10～16 个工作节点的分布式集群使用分布式编码卷积算法计算卷积的性能。结果显示，当节点个数超过 10 时，随着节点数的增加，性能开始持平且呈下降趋势。其原因为随着分布式集群节点个数的增加，卷积计算过程中需要的数据交换增加，分布式集群中出现 straggler 的概率也逐渐增加。由于分布式编码的计算时间逐渐超越了其为系统带来的性能优化，因此还需要对现有的编码方法进行改进，以改善分布式集群规模增大时算法计算卷积任务的性能衰减的问题。

结束语 如何优化卷积神经网络的运算速度及提高其计算效率一直是热门的研究课题。近年来，很多研究指出分布式计算架构可以提高卷积神经网络的计算速度，进而优化深度学习的训练效率，然而分布式系统中普遍存在落跑者问题，其可能会拖慢整个系统执行任务的时间。鉴于此，本文提出了一种基于分布式编码的卷积算法(DCCA)，该算法将分布式计算、Winograd 算法和分布式编码相结合，能够有效提升卷积的计算速度，且可以减轻落跑者问题带来的性能下降。本文通过实验验证了 DCCA 算法的性能，实验结果表明，本文提出的 DCCA 算法相比传统的分布式卷积算法，可以有效提升卷积的计算速度，而本文算法中使用的分布式编码能够较好地减小落跑者对分布式卷积计算的影响。在未来工作中，我们将研究如何把分布式编码卷积算法应用于三维卷积这类更高维度的卷积计算中，以达到将此种方法应用于如增强现实(Augmented Reality，AR)等如今较为热门的领域的目的。

参 考 文 献

[1] PARHI K K. VLSI digital signal processing systems：design and implementation[M]. John Wiley & Sons，2007.

[2] LIU S，DU Z，TAO J，et al. Cambricon：An instruction set architecture for neural networks[C]// 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). IEEE，2016：393-405.

[3] LECUN Y，BENGIO Y，HINTON G. Deep learning[J]. Nature，

2015,521(7553):436-444.
[4] LI Y,ZHANG Z. Parallel computing:review and perspective [C]// 2018 5th International Conference on Information Science and Control Engineering (ICISCE). IEEE,2018:365-369.
[5] LI S,MADDAH-ALI M A,YU Q,et al. A fundamental tradeoff between computation and communication in distributed computing [J]. IEEE Transactions on Information Theory, 2017, 64(1):109-128.
[6] LI S,MADDAH-ALI M A,AVESTIMEHRA S. Compressed coded distributed computing [C] // 2018 IEEE International Symposium on Information Theory (ISIT). IEEE, 2018: 2032-2036.
[7] ANDERSON A,GREGG D. Optimal DNN primitive selection with partitioned boolean quadratic programming [C] // Proceedings of the 2018 International Symposium on Code Generation and Optimization. 2018:340-351.
[8] MATHIEU M,HENAFF M,LECUN Y. Fast training of convolutional networks through ffts[J]. arXiv:1312.5851,2013.
[9] VASILACHE N,JOHNSON J,MATHIEU M,et al. Fast convolutional nets with fbfft: A GPU performance evaluation [J]. arXiv:1412.7580,2014.
[10] CONG J,XIAO B. Minimizing computation in convolutional neural networks [C] // International Conference on Artificial Neural Networks. Springer,Cham,2014:281-290.
[11] STRASSEN V. Gaussian elimination is not optimal[J]. Numerische mathematik,1969,13(4):354-356.
[12] LAVIN A,GRAY S. Fast algorithms for convolutional neural networks[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016:4013-4021.
[13] WINOGRAD S. Arithmetic complexity of computations[M]. Siam,1980.
[14] WANG Z,LAN Q,HE H,et al. Winograd algorithm for 3D convolution neural networks[C]// International Conference on Artificial Neural Networks. Springer,Cham,2017:609-616.
[15] ZHAO Y,WANG D,WANG L. Convolution accelerator designs using fast algorithms[J]. Algorithms,2019,12(5):112.
[16] FERNANDEZ-MARQUES J,WHATMOUGH P N,MUNDY A,et al. Searching for Winograd-aware Quantized Networks [J]. arXiv:2002.10711,2020.
[17] ZAHARIA M,CHOWDHURY M,FRANKLIN M J,et al. Spark:Cluster computing with working sets[C]// HotCloud'10 Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. 2010:4-10.
[18] DEAN J,GHEMAWAT S. MapReduce:simplified data processing on large clusters[J]. Communications of the ACM,2008, 51(1):107-113.
[19] DEAN J,BARROSOL A. The tail at scale[J]. Communications of the ACM,2013,56(2):74-80.
[20] AGARWAL A,DUCHIJ C. Distributed delayed stochastic optimization[C]// Advances in Neural Information Processing Systems. 2011:873-881.
[21] RECHT B,RE C,WRIGHT S,et al. Hogwild!:A lock-free approach to parallelizing stochastic gradient descent[J]. Advances in neural information processing systems,2011,24:693-701.
[22] ANANTHANARAYANAN G,GHODSI A,SHENKER S,et al. Effective straggler mitigation:Attack of the clones[C]// Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation. 2013:185-198.
[23] SHAH N B,LEE K,RAMCHANDRAN K. When do redundant requests reduce latency? [J]. IEEE Transactions on Communications,2015,64(2):715-722.
[24] WANG D,JOSHI G,WORNELL G. Efficient task replication for fast response times in parallel computation[C]// The 2014 ACM International Conference on Measurement and Modeling of Computer Systems. 2014:599-600.
[25] GARDNER K,ZBARSKY S,DOROUDI S,et al. Reducing latency via redundant requests: Exact analysis [J]. ACM SIGMETRICS Performance Evaluation Review, 2015, 43(1): 347-360.
[26] CHAUBEY M,SAULE E. Replicated data placement for uncertain scheduling[C]// 2015 IEEE International Parallel and Distributed Processing Symposium Workshop. IEEE, 2015: 464-472.
[27] LEE K,PEDARSANI R,RAMCHANDRAN K. On scheduling redundant requests with cancellation overheads[J]. IEEE/ACM Transactions on Networking,2016,25(2):1279-1290.
[28] JOSHI G,SOLJANIN E,WORNELL G. Efficient redundancy techniques for latency reduction in cloud systems [J]. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS),2017,2(2):1-30.
[29] JOSHI G,LIU Y,SOLJANIN E. On the delay-storage trade-off in content download from coded distributed storage systems[J]. IEEE Journal on Selected Areas in Communications, 2014, 32(5):989-997.
[30] LEE K,LAM M,PEDARSANI R,et al. Speeding up distributed machine learning using codes[J]. IEEE Transactions on Information Theory,2017,64(3):1514-1529.
[31] REISIZADEH A,PRAKASH S,PEDARSANI R,et al. Coded computation over heterogeneous clusters[J]. IEEE Transactions on Information Theory,2019,65(7):4227-4242.
[32] DIMAKIS A G,GODFREY P B,WU Y,et al. Network coding for distributed storage systems[J]. IEEE transactions on information theory,2010,56(9):4539-4551.
[33] YU Q,MADDAH-ALI M,AVESTIMEHR S. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication[C]// Advances in Neural Information Processing Systems. 2017:4403-4413.
[34] PEI D,SALOMAA A,DING C. Chinese remainder theorem:applications in computing,coding,cryptography[M]. World Scientific,1996.

YUAN Chen-yu, born in 1998, postgraduate. His main research interests include distributed computing and so on.

XIE Zai-peng, born in 1982, Ph.D, associate professor, is a member of China Computer Federation. His main research interests include distributed systems, embedded system and Internet of Things.