CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 40 卷第 12 期 2018 年 12 月
Vol. 40, No. 12, Dec. 2018

文章编号:1007-130X(2018)12-2103-09

一种基于 GPU 的高性能稀疏卷积神经网络优化*

方　程,邢座程,陈顼颢,张　洋
(国防科技大学计算机学院,湖南 长沙 410073)

摘　要:卷积神经网络 CNN 目前作为神经网络的一个重要分支,相比于其他神经网络方法更适合应用于图像特征的学习和表达。随着 CNN 的不断发展,CNN 将面临更多的挑战。CNN 参数规模变得越来越大,这使得 CNN 对计算的需求量变得非常大。因此,目前产生了许多种方式对 CNN 的规模进行压缩。然而压缩后的 CNN 模型往往产生了许多稀疏的数据结构,这种稀疏结构会影响 CNN 在 GPU 上的性能。为了解决该问题,采用直接稀疏卷积算法,来加速 GPU 处理稀疏数据。根据其算法特点将卷积运算转换为稀疏向量与稠密向量内积运算,并将其在 GPU 平台上实现。本文的优化方案充分利用数据稀疏性和网络结构来分配线程进行任务调度,利用数据局部性来管理内存替换,使得在稀疏卷积神经网络 SCNN 中的 GPU 仍能够高效地处理卷积层运算。相比 cuBLAS 的实现,在 AlexNet、GoogleNet、ResNet 上的性能提升分别达到 1.07×～1.23×、1.17×～3.51×、1.32×～5.00×的加速比。相比 cuSPARSE 的实现,在 AlexNet、GoogleNet、ResNet 上的性能提升分别达到 1.31×～1.42×、1.09×～2.00×、1.07×～3.22×的加速比。

关键词:卷积神经网络;稀疏;并行;优化;图形处理器
中图分类号:TP183　　**文献标志码**:A
doi:10.3969/j.issn.1007-130X.2018.12.002

A GPU-based high-performance optimization method of sparse convolutional neural networks

FANG Cheng, XING Zuo-cheng, CHEN Xu-hao, ZHANG Yang
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: As an important branch of neural networks, the convolutional neural network (CNN) is currently more suitable for learning and expressing image features than other neural network methods. With the continuous development of the CNN, there are more challenges. The parameters scale of the CNN is growing larger, which makes the demand for computation enormous. There are many ways to compress CNN scale, however, the compressed CNN usually introduces a number of sparse data structures. These sparse data structures can hurt the performance of the CNN on GPU. In order to solve this problem, we adopt the direct sparse convolution algorithm proposed in 2017 to accelerate GPU's processing of sparse data. According to the characteristics of this algorithm, we transform convolution operation into an inner product of the sparse vector and dense vector on GPU platform. Our optimization makes full use of the sparse data and network structure to allocate threads for task scheduling, and uses data locality to manage memory replacement. It enables the GPU to deal with the operation on the convolution layer efficiently in the sparse CNN. Compared with the cuBLAS, our proposal achieves a speed-up of 1.07×～1.23×, 1.17×～3.51× and 1.32×～5.00× on AlexNet, GoogleNet and ResNet respectively. Compared with the cuSPARSE, our method achieves a speed-up of 1.31×～1.42×, 1.09×

* 收稿日期:2018-06-21;修回日期:2018-08-15
基金项目:国家自然科学基金(61170083)
通信地址:410073 湖南省长沙市国防科技大学计算机学院
Address: College of Computer, National University of Defense Technology, Changsha 410073, Hunan, P. R. China

～2.00×and 1.07×～3.22× on AlexNet, GoogleNet, and ResNet respectively.

Key words: convolutional neural network; sparse; parallel; optimization; GPU

1 引言

CNN(Convolutional Neural Network)目前作为深度学习领域中的一个重要模型,在计算机图像[1]、语音识别[2]、游戏比赛[3]以及机器人[4]等方面都扮演着越来越重要的角色。但是,随着CNN的发展,CNN网络规模和网络层数不断增加,参数规模也变得越来越庞大。1990年,早期的卷积神经网络模型用于手写识别使用的参数数量不到100M[5]。20年后,AlexNet[1]和VGG[6]分别使用了61M和138M个参数来对1 000个图像进行分类。显而易见,这对CNN实现过程中的硬件资源、网络结构、算法优化等各方面都会产生诸多的挑战。CNN加速[7]、CNN参数的量化分析与研究[8]、参数规模的缩小与权重删减[9-12]都将成为热门的研究方向。

为了应对CNN对计算需求量的不断增加,采用高性能的GPU已经成为加速CNN[13,14]的一项重要措施。此外还有研究给出了压缩CNN的解决方案。压缩CNN方法主要分为两类:一类是基于分解[15,16],另一类是基于删减[17,18]。基于删减的方法是在保证网络训练和测试结果精确度[8]没有损失的前提下减少参数数量。权重删减下的深度压缩[9,10]方式能够使AlexNet和VGG-16的参数规模分别缩小9倍和13倍,在CPU和GPU的架构下实现了3～4倍的加速。但是,权重删减的性能提升远远落后于实际减少乘累加操作的性能提升,特别是在GPU的硬件设备上,这一类似的性能损失经常发生。同时,有研究提出了全新的直接稀疏卷积算法[19],它在CPU的架构下相比原有的稠密算法在AlexNet卷积层上实现了3.1×～7.3×的加速。对于训练好的CNN来说,卷积层的卷积运算是测试过程运行时间最主要的部分。所以,对卷积层的卷积运算优化成为解决该加速优化问题中的关键路径。

由于权重删减的方式牺牲了数据的规则性,SCNN(Sparse Convolutional Neural Network)内部产生了大量的稀疏计算成分。稀疏数据处理与GPU体系结构特性不匹配[20]。原有GPU架构下对卷积核提供卷积数学运算的cuBLAS和cuSPARSE并不能很好地应对这种不匹配。同时,GPU和CPU在体系结构上存在的差异使得很多针对CPU优化的稀疏卷积算法并不能在GPU上适用。我们采用了一个高效的直接稀疏卷积算法,对其在GPU的平台上进行优化,从而解决权重删减产生稀疏数据所带来的性能损失。

本篇论文主要贡献在以下几点:

(1)针对卷积层关键优化路径上完成直接稀疏卷积算法[19]在GPU架构上的并行化实现,打破GPU上采用传统稠密算法的局限性,给出了一种可行且高效的GPU加速SCNN方案。

(2)采用最大限度的线程映射,充分利用GPU的硬件计算资源,防止产生稀疏结构运算对GPU计算资源的浪费。

(3)采用最优的任务调度,合理安排每个单线程的任务工作,减少线程同步过程中某一部分线程等待时间,提高资源利用率。

(4)充分利用直接稀疏卷积算法数据处理过程中的数据局部性,增加数据复用,对于同一block下的所有线程,采用共享内存来减少数据访存时间。

最终我们在CAFFE(Convolutional Architecture for Fast Feature Embedding)架构下所实现的稀疏卷积神经网络,对同一训练好的AlexNet、GoogleNet、ResNet,在GPU GTX1060上,与CAFFE本身搭建的由cuBLAS和cuSPARSE所提供的数学库支持的卷积层进行测试对比。相比cuBLAS的实现,我们在AlexNet、GoogleNet、ResNet上性能提升分别达到1.07×～1.23×、1.17×～3.51×、1.32×～5.00×的加速比。相比cuSPARSE的实现,在AlexNet、GoogleNet、ResNet上性能提升分别达到1.31×～1.42×、1.09×～2.00×、1.07×～3.22×的加速比。

2 背景

这一节介绍实现卷积运算的几种方式,并说明了它们各自的局限性,从而阐述本文的研究意义和研究背景。

2.1 降维方式

目前很多CNN卷积层的卷积操作都是通过降维方式实现的[18]。图1所示是一个简单的用降维方式实现卷积的例子,图中参数可参考表1。

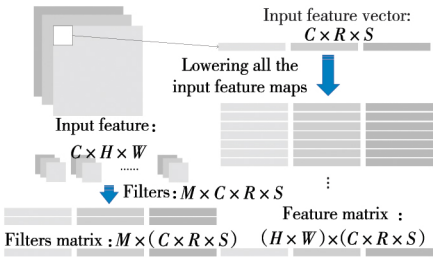


Figure 1　Lowering method performs convolution operation

图 1　降维展开方式实现卷积运算

Table 1　Description of convolution parameters

表 1　卷积参数描述

参数	描述
N	三维输入特征矩阵批处理任务的 *batchsize*
M	三维卷积核对应输出特征矩阵的通道数目
C	输入特征矩阵对应卷积核的通道数目
H/W	输入特征矩阵在该次处理中的大小
R/S	卷积核在该次处理中的大小(在全连接层中,应与 H/W 相等)
E/F	该次处理完成后输出特征矩阵大小(在全连接层中,应等于 1)
U	卷积核在输入特征矩阵上局部感知区域的移动步长

假设输入特征矩阵的 *batchsize* 为 1,其输入通道数为 C,输入特征矩阵大小为 $H\times W$,预输出通道数为 M,每一个卷积核的实际大小为 $R\times S$(在实际应用中,可以通过设置步长 U 来控制卷积核在输入特征矩阵上的局部感知区域的位置,后文我们假设 U 默认为 1)。那么一共有 M 个卷积核,每个卷积核包含 C 个通道。降维方式通过将输入特征矩阵和卷积核分别以行展开的方式生成新的特征矩阵 $\boldsymbol{I}^{\text{lowering}}$ 和卷积核矩阵 $\boldsymbol{W}^{\text{lowering}}$。那么,最终卷积的计算过程可以表示为:

$$\boldsymbol{O}=\boldsymbol{W}^{\text{lowering}}\boldsymbol{I}^{\text{lowering T}}$$

降维方式将卷积运算转换为矩阵乘法。在基础线性代数子程序库 BLAS(Basic Linear Algebra Subprograms)中,GEMM(GEneralized Matrix Multiplication)的函数接口实现了两个稠密矩阵的乘法运算。在 CAFFE 的框架下,CNN 卷积层中卷积运算所采用的方式也是降维方式,具体是通过 im2col 函数和 GEMM 函数实现。此外,CAFFE 还支持了 CUDA 版本下由 cuBLAS 所提供的 GPU 架构下并行实现的降维方式。

在降维方式展开生成新的特征矩阵 $\boldsymbol{I}^{\text{lowering}}$ 的过程中,卷积核所感知的局部区域重叠部分的元素都进行了多次重复复制,增加了存储开销。特别是在 SCNN 中,这种大量的数据重复复制浪费了大量的存储资源。此外,GEMM 是针对稠密矩阵实现的矩阵乘法,对于处理稀疏矩阵浪费了 GPU 大量的计算资源。所以,我们需要一个针对 GPU 实现的稀疏卷积运算。

2.2　直接稀疏卷积

直接稀疏卷积(Direct Sparse Convolutions)作为一种全新的卷积方式在 2017 年的 ICLR 会议上被首次提出[19]。该算法在 CPU 的架构下相比原有的算法在 AlexNet 卷积层上实现了 3.1×~7.3×的加速。

相比降维方式,直接稀疏卷积去除了输入特征矩阵中的数据重复复制。该算法将卷积核矩阵的规模扩展到输入矩阵的相同大小。对于延展后的卷积核行展开生成向量 $\boldsymbol{W}_m$,其长度为 $C\times H\times W$。由于有 M 个卷积核,对每一个卷积核进行延展后得到了 $M\times(C\times H\times W)$ 的权重矩阵。对于该批次任务下的输入矩阵以行展开的方式形成列向量 $\boldsymbol{I}$,其长度为 $C\times H\times W$。那么,在计算卷积的过程中,对于不同感知区域的元素可以通过调整向量 $\boldsymbol{I}$ 的起始指针,使得卷积核映射到正确的局部区域。其具体算法如图 2 所示。

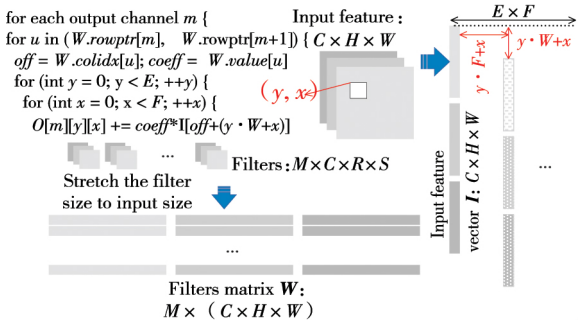
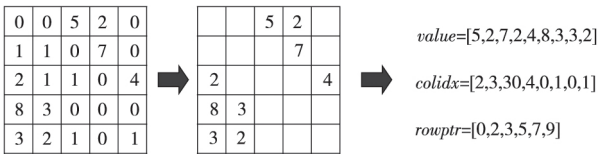


Figure 2　Direct sparse convolution

图 2　直接稀疏卷积

该批次任务下,直接稀疏卷积结果可以表示为:$\boldsymbol{O}_m=\boldsymbol{W}_m\cdot\boldsymbol{I}^{\text{virtual}}$。其中矩阵 $\boldsymbol{I}^{\text{virtual}}$ 是由列向量 $\boldsymbol{I}$ 调整起始指针所得到的。那么,我们可以进一步简化结果为:$\boldsymbol{O}_{m,y,x}=\boldsymbol{W}_m\cdot\boldsymbol{I}_{y\cdot W+x}$。所有输出通道下的稀疏向量 $\boldsymbol{W}_m$ 构成稀疏矩阵 $\boldsymbol{W}^{\text{Sparse}}$,采用行压缩存储 CSR(Compressed Spares Row)格式,存储如图 3 所示。数组 *value* 记录矩阵 $\boldsymbol{W}^{\text{sparse}}$ 中的非零元素。数组 *colidx* 记录每个非零元素在矩阵 $\boldsymbol{W}^{\text{sparse}}$ 中的列指针。数组 *rowptr* 记录矩阵 $\boldsymbol{W}^{\text{sparse}}$ 中每一行起始元素在 *value* 中的指针。

直接稀疏卷积将卷积运算抽象成稀疏向量 $\boldsymbol{W}_m$ 对稠密向量 $\boldsymbol{I}_{y\cdot W+x}$ 的内积。此外,由于 SCNN 采用 CSR 或 CSC(Compressed Sparse Column)的稀疏数据存储格式,对于运算过程中的延展实际上并没有增加存储开销,只是调整了矩阵中非零元素



a 稠密权重矩阵 b 稀疏权重矩阵 CSR format行压缩存储格式 c 稀疏矩阵的CSR格式

Figure 3 CSR format

图 3 行压缩存储格式

的行列指针。相比降维方式,直接稀疏卷积更适合在 GPU 上实现 SCNN。

3 设计与实现

本节介绍本文所提方法的具体实现和优化。由于权重删减后 SCNN 产生了大量稀疏数据结构,而传统的降维方式并不能保证稀疏矩阵卷积的计算性能,本文采用全新的直接稀疏卷积来替代降维方式,弥补性能损失。除此以外,GPU 的体系结构特征需要在实现过程中对线程映射、任务分配以及内存管理进行更多的考虑和优化。

3.1 概述

直接稀疏卷积的实现主要由两部分组成:(1)数据预处理,主要完成对卷积核矩阵的延展,生成稀疏向量 W_m 和稠密向量 I;(2)计算过程,主要完成所有的 MAC 操作,并准确更新计算过程中的指针。

第(1)部分如图 4 所示。在这里,权重矩阵为 $M\times(C\times R\times S)$ 的稀疏矩阵,按照 CSR 格式存储于物理内存中。对于输出通道 m 中的第 j 个非零元素 (c,y,x) 有:

$$x = col\%S$$
$$y = (col/S)\%R$$
$$c = col/(S*R)$$

其中 $col=colidx[j]$。那么,延展后的权重矩阵大小为 $M\times(C\times H\times W)$,同一个非零元素 (c,y,x) 的 CSR 格式存储下的列指针更新为:$colidx[j]=(c*H+y)*W+x$。

直接稀疏卷积的计算过程可以表示为:$O_{m,y,x}=W_m\cdot I_{y\cdot W+x}$。其核心在于实现稀疏向量 W_m 与稠密向量 $I_{y\cdot W+x}$ 的内积运算。对于计算输出矩阵中的点 (m,y,x),需要完成的 MAC 操作数取决于稀疏向量 W_m 的非零元素数目。由于对同一输出通道 m 中的所有点,稀疏向量 W_m 是恒定不变的,所以计算这些输出节点所需要的 MAC 操作数相等。在直接稀疏卷积算法中矩阵 $I^{virtual}$ 是由向量 I 生

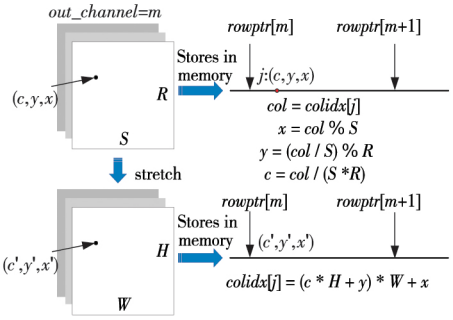


Figure 4 Weight stretched

图 4 权重延展

成,其中每一个列向量 $I_{y\cdot W+x}$ 的起始指针所指向的元素为 $I[y\cdot W+x]$。根据这一特点,我们仅将向量 I 的元素常驻内存,而不是存储整个稠密矩阵 $I^{virtual}$。

考虑到实际的 CNN 模型中,所有卷积层经过权重删减后的稀疏度存在差异,我们通过下列方式来计算当前卷积层的稀疏度:

$$sparsity = 1.0 - N_{nonzero}/M*kernel\_size$$

其中,$N_{nonzero}$ 为当前卷积层的所有非零元素数目,M 为当前卷积层输出通道数,$kernel\_size$ 为卷积核规模大小。

对于不同稀疏度的卷积层,我们设置一个阈值。稀疏度大于该阈值的卷积层采用优化后的直接稀疏卷积方式,小于该阈值的卷积层则仍采用原有的降维方式。对于稠密数据和稀疏数据的分别处理,使得对于任意稀疏度的卷积层都能够实现最佳的计算性能,可以最大限度提高整个网络的运行性能。由于在最终实验过程中采用了 IntelSkimcaffe 开源项目(https://github.com/IntelLabs/SkimCaffe)中的稀疏 CNN 网络结构,CNN 中的卷积层的稀疏度集中在 0 和 0.7~0.96 这两个区域,所以设置阈值仅仅是排除了稀疏度为 0 的稠密层。

3.2 并行策略

相比 CPU,GPU 拥有更多的处理核心,如何合理分配和充分利用这些处理核心是本文设计的关键。接下来我们将分别介绍直接稀疏卷积两个过程的并行策略。

对于过程一,即图 5 中所示 $oc=m$ 时所有非零元素的列指针更新。

将整个权重矩阵进行延展就是更新权重矩阵内所有非零元素的列指针 $colidx$。那么,我们设置线程 $Thread_m$ 完成稀疏向量 W_m 内所有非零元素的列指针更新。

对于过程二,每一个线程计算输出特征矩阵中的一个点 (m,y,x),如图 5 所示。由于输入特征矩

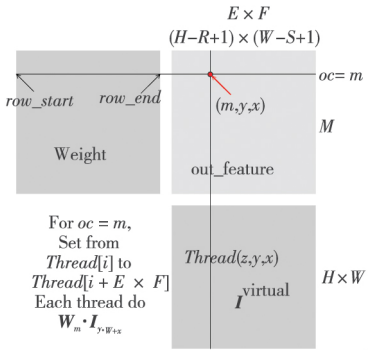


Figure 5　Mapping rules in the calculation process without optimization

图 5　计算过程映射规则(无优化)

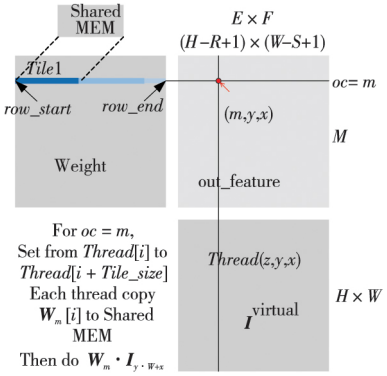


Figure 6　Mapping rules in the calculation process (with shared MEM)

图 6　计算过程映射规则(共享内存)

阵 $\boldsymbol{I}^{\text{virtual}}$ 中每一列向量是由稠密列向量 $\boldsymbol{I}$ 移动初始指针得到的,那么我们将稠密列向量 $\boldsymbol{I}$ 的所有数据常驻内存。当需要计算不同的输出点 (m,y,x) 时,计算其对应列向量 $\boldsymbol{I}_{y\cdot W+x}$ 相对向量 $\boldsymbol{I}$ 的偏移量 pos,其计算公式为:$pos=y\cdot W+x$。通过对向量 $\boldsymbol{I}$ 的初始指针增加 pos 偏移量得到对应向量 $\boldsymbol{I}_{y\cdot W+x}$:$*input_{\text{ptr}}=input+pos$。该过程避免了数据的重复复制,仅通过调整指针来完成当前输出通道的全部计算。由于 $\boldsymbol{W}^{\text{sparse}}$ 作为稀疏矩阵采用 CSR 的格式存储在物理内存中,第 m 个卷积核下所有非零元素对应存储在 $rowptr[m]$ 行。该行元素在物理内存中存储非零元素的一维数组 $value$ 中的起始位置为 row_strat($row\_start=rowptr[m]$),结束位置为 row_end($row\_end=rowptr[m+1]$)。那么,对于线程 $Thread_{(z,y,x)}$,需要完成下列计算:

$$\begin{aligned}&\text{for } j=row_{\text{start}};j<row_{\text{end}};++j\\&\quad output[(m\cdot E+y)\cdot F+x]+=\\&\quad value[j]\cdot input_{\text{ptr}}[colidx[j-row_{\text{start}}]]\end{aligned}$$

输出点 (m,y,x) 与线程 $Thread_{(z,y,x)}$ 一一对应。

通过分别对过程一和过程二实现并行化,我们在 GPU 的架构下实现了直接稀疏卷积。在实际的测试中,这一实现的具体性能并没有达到预期效果(这一点将在第 4 节具体说明)。所以,接下来增加了对数据局部性的考虑,对实现的并行策略进行了进一步优化。

3.3　局部性优化

由于输入特征向量的数据复用,我们采用了 $\boldsymbol{I}^{\text{virtual}}$ 的方式来减小带宽需求。通过更改访存指针来读取向量 $\boldsymbol{I}$ 中的值。同样地,为了增加 Cache 块的命中率,希望优先计算同一输出通道的值。由于实际测试性能达不到预期效果,我们增加了共享内存优化的版本,其具体映射规则如图 6 所示。

对于输出通道 m,需要 $E\times F$ 个线程来完成计算任务。但是在实际情况中,GPU 所能设置的最大 $block\_size$ 小于 $E\times F$,所以对于同一个 block 内的所有线程会在短时间内经常访问向量 $\boldsymbol{W}_m$,直到该 block 内的所有线程完成计算。此时内存常驻的数据仅仅只有向量 $\boldsymbol{I}$ 和向量 $\boldsymbol{W}_m$。

由于同一个 block 下的所有线程都要求对 $\boldsymbol{W}_m$ 进行数据访问,我们将 $\boldsymbol{W}_m$ 放入共享内存中,以减少 $\boldsymbol{W}_m$ 的数据访存时间。共享内存对于同一 block 块下的线程是共同可见的。考虑到 GPU 内共享内存大小的限制,将 $\boldsymbol{W}_m$ 分块化,块 $\boldsymbol{Tile}_i$ 为特定长度的一维数组。将 $\boldsymbol{Tile}_i$ 的长度设定为 block 下的线程总数,并使 $\boldsymbol{Tile}_i$ 包含的数据能够常驻共享内存。由于 $\boldsymbol{W}_m$ 采用 CSR 格式存储,那么仅需将对应数组 $value$ 和数组 $colidx$ 的值存入共享内存。在计算输出结果前,需要将 $\boldsymbol{Tile}_i$ 对应的 $value$ 和 $colidx$ 写入共享内存中的数组 $value_{\text{shared}}$ 和数组 $colidx_{\text{shared}}$。由于 $\boldsymbol{Tile}_i$ 长度与线程数相等,那么对于线程 $Thread_{(z,y,x)}$ 需要完成的读写工作如下所示:

$$\begin{aligned}&colidx_{\text{shared}}[y*blockDim.y+x]=\\&colidx[(y*blockDim.y+x)+i*Tile_{\text{size}}];\\&value_{\text{shared}}[y*blockDim.y+x]=\\&value[(y*blockDim.y+x)+i*Tile_{\text{size}}]\end{aligned}$$

每个线程只需要将 $\boldsymbol{Tile}_i$ 块内一个元素的 $value$ 和 $colidx$ 数组值写入共享内存。其中,$blockDim.y$ 为 GPU 线程设置中 block 块在 y 方向上的维度大小,$Tile_{\text{size}}$ 为设置的 Tile 块的长度。

为了防止读后写,为同一 block 下的所有线程增加同步操作。线程 $Thread_{(z,y,x)}$ 将 $\boldsymbol{Tile}_i$ 数据写入共享内存后进行等待,直到所有线程完成操作。当 block 块内所有线程完成同步后,线程 $Thread_{(z,y,x)}$ 需要完成共享内存内向量 $\boldsymbol{Tile}_i$ 与向

量 $I_{y\cdot W+x}$ 的内积运算。其具体计算如下所示：

$$\text{for } j = row_{\text{start}}; j < row_{\text{end}}; ++j$$
$$sum \mathrel{+}= value_{\text{shared}}[J]\cdot input_{\text{ptr}}[colidx_{\text{start}}[j - row_{\text{start}}]]$$

每一次累加操作后同步线程，当访问共享内存未命中时，替换下一个 Tile 块到共享内存。将每个块替换下来的部分和保存在寄存器 *sum* 中，这样当下一个块被换进共享内存时，线程能够正常工作。当输出通道 m 所有 Tile 块都被替换过后，将部分和 *sum* 输出：$output[(m\cdot E+y)\cdot F+x]=sum$。输出点 (m,y,x) 与线程 $Thread_{(z,y,x)}$ 的映射关系与之前的一样。

通过增加共享内存以及对数据局部性的考虑，实验结果最终达到了预期性能。相比未优化的直接稀疏卷积，本文在 GPU 上实现了更为高效的性能。

4 性能评估

这一节用于描述具体的实验设置，并对本文优化结果给出直观的性能评估。在 CAFFE 的框架下完成对整个设计的性能测试。采用 ImageNet 的数据集对已经训练好的 AlexNet、GoogleNet 和 ResNet 三个稀疏模型进行测试。4.1 小节我们给出了设计的整体性能，并将未优化版本和增加共享内存版本进行了性能对比；4.2 小节给出了最终优化版本与 CAFFE 框架提供的 cuBLAS 和 cuSPARSE 两种数学库所实现的卷积层的加速比，以及不同 *batchsize* 下的加速比。

4.1 总体性能

实验采用的 GPU 型号为 GTX 1060 3 GB。设置稀疏度阈值为 0.6，*batchsize* 为 128。训练好的 AlexNet 模型包含 5 层卷积层，每层的稀疏度根据公式计算的结果如表 2 所示。

Table 2 Parameters of AlexNet convolution layers

表 2 AlexNet 卷积层参数

Layer	Sparsity	Input size	Filter size
CONV1	0	3×227×227	96×3×11×11
CONV2	0.872 7	96×27×27	256×96×5×5
CONV3	0.930 9	256×13×13	384×256×3×3
CONV4	0.942 7	384×13×13	384×384×3×3
CONV5	0.912 1	384×13×13	256×384×3×3

对稀疏度大于 0.6 的 CONV2、CONV3、CONV4 和 CONV5 四个卷积层采用直接稀疏卷积的方式。设置一个 block 块下的总线程数为 1 024，那么每次替换进共享内存的 Tile 块长度为 1 024。实验结果记录了 50 000 次迭代中每一次迭代完成 Forward 过程所需的时间，每 100 次迭代的 Forward 执行时间取平均值，具体结果如图 7 所示。其中 Base 为未优化初始版本的执行时间曲线，Tiled 为增加共享内存优化后版本的执行时间曲线。

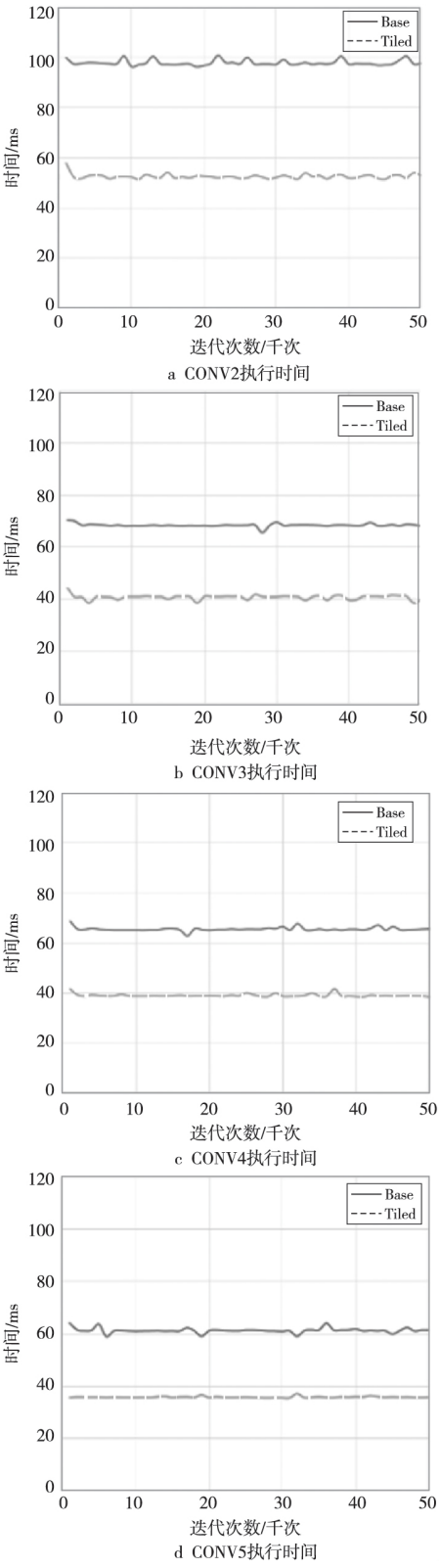


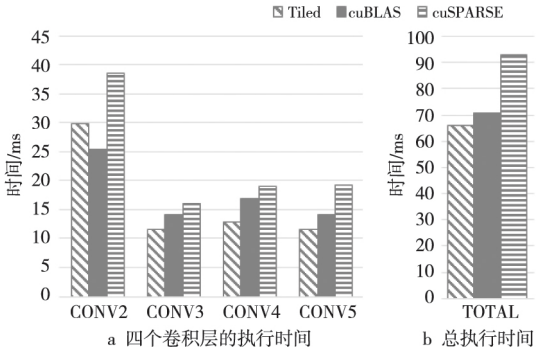
Figure 7 Execution time

图 7 执行时间

Tiled 版本相比 Base 版本在各层上都有较大的性能提升。在各层上 Tiled 版本的性能分别提升了 46.7%、41.1%、41.5%、42.6%。这说明本文的优化在 GPU 架构上起到了实质性作用。通过增加共享内存、合理分配线程、增加数据复用,在 GPU 架构上实现直接稀疏卷积,实现了高效的稀疏卷积神经网络优化。本文采用的直接稀疏卷积并行方式适应了 GPU 的体系结构特征,充分利用了硬件计算资源。为了进一步说明本文设计性能的优越性,将在 4.2 小节与现有的 CNN 卷积层实现进行性能对比。

4.2　执行时间分析

为了进一步说明优化后的性能提升,在 AlexNet 模型基础上对比本文的设计与原有 CAFFE 框架下所实现的卷积神经网络。CAFFE 通过 cuBLAS 提供的函数接口在 GPU 上主要实现了降维。cuBLAS 是在 GPU 上实现的 CUDA 数学函数库。此外,CAFFE 还采用了 cuSPARSE 库来优化处理稀疏卷积。给出了 *batchsize* 为 64 时的 AlexNet 模型各层执行时间对比,如图 8 所示。

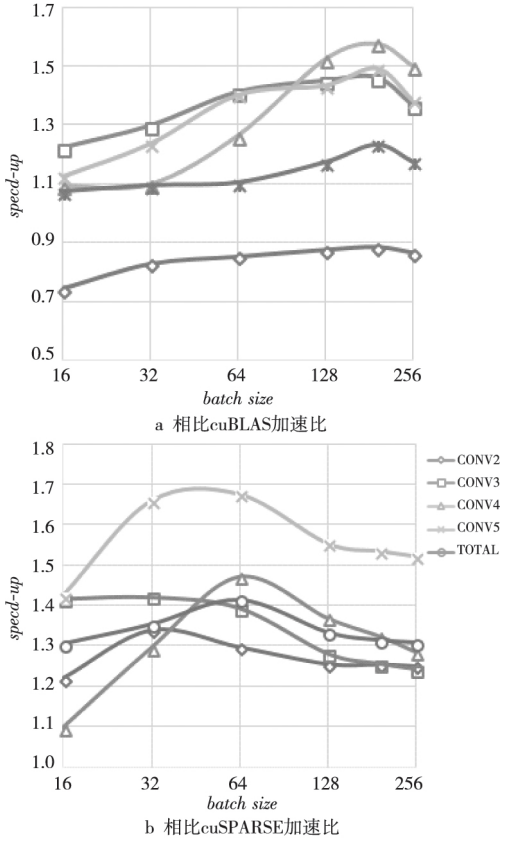


a 四个卷积层的执行时间　　b 总执行时间

Figure 8　Execution time of AlexNet sparse convolution layers in different implementation models

图 8　AlexNet 各稀疏卷积层不同实现模式的执行时间

同样地,只列出了 AlexNet 中稀疏度大于 0.6 的四个卷积层以及它们的总执行时间。从图 8 可以看到,相比 cuBLAS 实现方法,本文的优化方法仅在 CONV2 上有略微的性能损失,而在 CONV3、CONV4、CONV5 上的性能分别提升了 41.1%、26.2%、40.1%,且总体性能提升了 10%;相比 cuSPARSE 实现方法,本文的优化方法在 CONV2、CONV3、CONV4、CONV5 上的性能分别提升了 29.6%、39.2%、47.1%、67.4%,且总体性能提升了 41.1%。通过分析表 2 给出了未删减前 AlexNet 各层结构参数,包括输入特征矩阵大小、卷积核大小以及稀疏度。而对于 CONV3、CONV4、CONV5 这三层来说,其稀疏度均高于 CONV2 的稀疏度,从而证明了本文的设计针对大规模高稀疏度数据有显著优化效果。此外,图 9 给出了不同 *batchsize* 下 AlexNet 各层的加速比。相比 cuBLAS,本文的优化方法在 *batchsize* 为 192 时得到了最佳的加速比;相比 cuSPARSE,当 *batchsize* 在 32～64 时性能最佳,从网络整体性能来看,*batchsize* 为 64 时加速性能最佳。这是由于 *batchsize* 过小或过大都会使在线负载任务过轻或过重,不能合理利用硬件计算资源。



a 相比cuBLAS加速比

b 相比cuSPARSE加速比

Figure 9　Speed-up for different *batchsizes*

图 9　不同 *batchsizes* 下的加速比

本文还对 GoogleNet 和 ResNet 模型进行了测试,同样也只给出了稀疏度大于 0.6 的卷积层的加速比,如图 10 所示。

对于 GoogleNet,相比 cuBLAS,本文优化方法仅在低维度有 1 层性能有略微的损失,其余高维度稀疏层实现了 1.17×～3.51×加速;相比 cuSPARSE,本文优化方法仅在高维度和低维度各出现了 1 层性能损失,其余各层实现了 1.09×～2.00×加速;在总体性能上,相比 cuBLAS 和 cuSPARSE 的方式分别实现了 1.34×和 1.21×加速。对于 ResNet,相比 cuBLAS,本文优化方法在所有稀疏层实现了 1.32×～5.00×加速;相比 cuSPARSE,本文优化方法仅在高维度出现了 2 层性能损失,其余各层实现了 1.07×～3.22×加速;在

总体性能上,相比 cuBLAS 和 cuSPARSE 的方式分别实现了 2.43×和 1.97×加速。

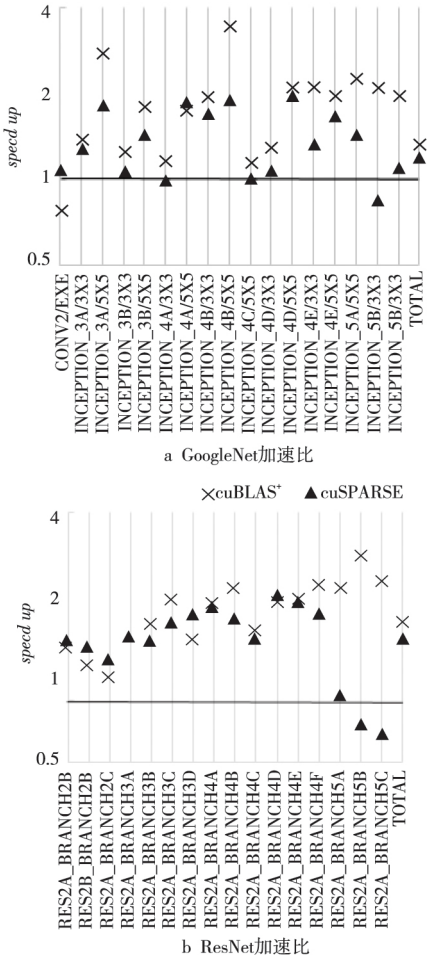


Figure 10 Speed-up of different models

图 10 不同模型下的加速比

由于各层删减后的数据规则性也会对实验结果产生一定的影响,所以在本文优化方案的测试结果中也出现了某些层的性能损失。但是,相比 cuBLAS 和 cuSPARSE,本文方法对高稀疏度层的优化加速效果显著。总体来说,本文优化方法实现了基于 GPU 架构的 SCNN 加速优化。

5 相关工作

相比传统意义上 GPU 加速 CNN 的实现方案[13,14],本文采用更优的数学内核和卷积运算算法,提高了整个系统的可优化程度。相比其他采用更合理的删减方式来切合 GPU 硬件特性[20]的加速方案,本文所提供的加速方案具有更好的可移植性和可靠性,对数据预处理的消耗小。此外,在文献[21]所提出的 Escort 优化版本上,本文改进了并行策略和映射规则,取得了更高的加速比。

6 结束语

本文通过在 GPU 上实现直接稀疏卷积算法,打破了 GPU 架构下传统稠密算法对于稀疏结构处理的局限性,有效解决了权重删减后 SCNN 在 GPU 上运行出现性能损失的问题。对于高稀疏度,甚至是 GPU 所不擅长处理的不规则数据,本文的设计仍然有着极大的优势。相比 CAFEE 下 cuBLAS 的实现,本文方法在 AlexNet、GoogleNet、ResNet 上的性能提升分别达到 1.07×~1.23×、1.17×~3.51×、1.32×~5.00×。相比 cuSPARSE 的实现,本文方法在 AlexNet、GoogleNet、ResNet 上的性能提升分别达到 1.31×~1.42×、1.09×~2.00×、1.07×~3.22×。

参考文献:

[1] Krizhevsky A,Sutskever I,Hinton G E. ImageNet classification with deep convolutional neural networks[C]// Proc of International Conference on Neural Information Processing Systems,2012:1097-1105.

[2] Sainath T N,Mohamed A R,Kingsbury B,et al. Deep convolutional neural networks for LVCSR[C]// Proc of IEEE International Conference on Acoustics,Speech and Signal Processing,2013:8614-8618.

[3] Silver D,Huang A,Maddison C J,et al. Mastering the game of go with deep neural networks and tree search[J]. Nature,2016,529(7587):484-489.

[4] Levine S,Finn C,Darrell T,et al. End-to-end training of deep visuomotor policies[J]. Journal of Machine Learning Research,2015,17(1):1334-1373.

[5] Lecun Y,Bottou L,Bengio Y,et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE,1998,86(11):2278-2324.

[6] Simonyan K,Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv:1409.1556,2014.

[7] Chetlur S,Woolley C,Vandermersch P,et al. cuDNN:Efficient primitives for deep learning[J]. arXiv:1410.0759,2014.

[8] Mao H,Han S,Pool J,et al. Exploring the granularity of sparsity in convolutional neural networks[C]// Proc of Computer Vision and Pattern Recognition Workshops,2017:1927-1934.

[9] Han S,Pool J,Tran J,et al. Learning both weights and connections for efficient neural networks[C]// Proc of the 28th International Conference on Neural Information Processing Systems,2015:1135-1143.

[10] Han S,Mao H,Dally W J. Deep compression:Compressing deep neural networks with pruning,trained quantization and Huffman coding[J]. Fiber,2015,56(4):3-7.

[11] Hassibi B, Stork D G, Wolff G J. Optimal brain surgeon and general network pruning[C]// Proc of IEEE International Conference on Neural Networks, 1993: 293-299.

[12] Cun Y L, Denker J S, Solla S A. Optimal brain damage[C]// Proc of International Conference on Neural Information Processing Systems, 1989: 598-605.

[13] Uetz R, Behnke S. Large-scale object recognition with CUDA-accelerated hierarchical neural networks[C]// Proc of 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, 2009: 536-541.

[14] An D C, Meier U, Masci J, et al. Flexible, high performance convolutional neural networks for image classification[C]// Proc of the International Joint Conference on Artificial Intelligence, 2011: 1237-1242.

[15] Zhang X, Zou J, He K, et al. Accelerating very deep convolutional networks for classification and detection[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2016, 38(10): 1943-1955.

[16] Lebedev V, Ganin Y, Rakhuba M, et al. Speeding-up convolutional neural networks using fine-tuned CP-decomposition [J]. arXiv: 1412.6553, 2014.

[17] Lavin A, Gray S. Fast algorithms for convolutional neural networks[C]// Proc of IEEE Conference on Computer Vision & Pattern Recognition, 2016: 4013-4021.

[18] Guo Y W, Yao A B, Chen Y R. Dynamic network surgery for efficient DNNs[C]// Proc of 2016 30th Conference on Neural Information Processing Systems, 2016: 1387-1395.

[19] Park J, Li S, Wen W, et al. Faster CNNs with direct sparse convolutions and guided pruning[C]// Proc of the 5th International Conference on Learning Representation, 2017: 1-11.

[20] Yu J, Lukefahr A, Palframan D, et al. Scalpel: Customizing DNN pruning to the underlying hardware parallelism[J]. ACM Sigarch Computer Architecture News, 2017, 45(2): 548-560.

[21] Chen X H. Escort: Efficient sparse convolutional neural networks on GPUs[J]. arXiv: 1802.10280, 2018.

作者简介:

方程(1994-),**男,湖南常德人,硕士,研究方向为高性能计算和深度学习。E-mail**: fchustc@163.com

FANG Cheng, born in 1994, MS, his research interests include high performance computing, and deep learning.

邢座程(1968-),**男,安徽无为人,博士,研究员,研究方向为高性能计算机体系结构。E-mail**: mxzhang@nudt.edu.cn

XING Zuo-cheng, born in 1968, PhD, research fellow, his research interest includes high performance computer architecture.

陈项颢(1988-),**男,湖南益阳人,博士,助理研究员,研究方向为计算机体系结构。E-mail**: chenxuhao@nudt.edu.cn

CHEN Xu-hao, born in 1988, PhD, associate research, his research interest includes computer architecture.

张洋(1988-),**男,四川绵阳人,博士生,助理研究员,研究方向为高性能计算和深度学习。E-mail**: zhangyang@nudt.edu.cn

ZHANG Yang, born in 1988, PhD candidate, associate research, his research interests include high performance computing, and deep learning.