计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20170932
56(4):825-835, 2019

基于 Intel 平台的 Winograd 快速卷积算法研究与优化

武 铮 安 虹 金 旭 迟孟贤 吕国锋 文 可 周 鑫
(中国科学技术大学 合肥 230022)
(zhengwu@mail.ustc.edu.cn)

Research and Optimization of Fast Convolution Algorithm Winograd on Intel Platform

Wu Zheng, An Hong, Jin Xu, Chi Mengxian, Lü Guofeng, Wen Ke, and Zhou Xin
(*University of Science and Technology of China*, *Hefei* 230022)

Abstract With the rapid development of deep learning, it's applied extensively for many fields, such as speech processing, image recognition, natural language understanding and so on, bringing great changes for scientific research and daily life. Intel which follows the trend of deep learning launched the second generation of Xeon Phi processor Intel KNL (knights landing), and released the third generation Intel KNM (knights mill), which brings new impetus and vitality for the prosperous development of deep learning. This paper mainly contributes to promoting perfect Intel MKL (math kernel library) DNN (deep neural network), and develops deep learning on Intel platform, according to research and optimization for the fast convolution algorithm Winograd. Combined with characteristics of Intel latest deep learning platform, such as AVX512, high-speed memory MCDRAM, various memory/SNC modes, two-dimensional grid-type cores structure and so on, this work aims to design and optimize the implementation of Winograd algorithm by analyzing memory allocation, data dependency, etc. Finally, on one hand, the typical CNN (convolutional neural network) model VGG19 is used to test and compare performance with Intel MKL convolution, achieving more than doubled acceleration of performance. On the other hand, the common different types of convolutions are used to test and compare performance with Intel MKL DNN and NVIDIA cuDNN, verifying applicability and objective use value about Winograd. The purpose of the paper is to provide important guiding significance for development of Intel platform in the field of deep learning.

Key words Winograd; deep learning; deep neural network (DNN); convolutional neural network (CNN); vectorization

摘 要 随着深度学习的快速发展,其在语音处理、图像识别和自然语言理解等领域被广泛应用,为科研产业以及日常生活带去了巨大的变革.Intel 紧跟深度学习的浪潮,推出了第 2 代 Xeon Phi 处理器 KNL (knights landing),其后又发布了第 3 代 Xeon Phi 处理器 KNM(knights mill),为深度学习的蓬勃发展带去了新的活力.通过在 Intel 平台上进行快速卷积算法 Winograd 的研究与优化,对比 Intel MKL

收稿日期:2017-12-04;**修回日期**:2018-04-25
基金项目:科技部国家重点研发计划基金项目(2016YFB1000403)
This work was supported by the National Key Research and Development Plan Foundation from Ministry of Science and Technology of China (2016YFB1000403).

(math kernel library) DNN(deep neural network)中的卷积性能,推动 Intel MKL DNN 中深度神经网络接口的完善以及 Intel 平台上深度学习的发展.研究中结合 Intel 最新深度学习平台的 AVX-512 指令集、高速内存 MCDRAM、多 Memory/SNC 模式、二维网格状内核结构等特性,并通过对内存分配、数据调度等情况的分析,设计优化 Winograd 算法,一方面选取典型的卷积神经网络(convolutional neural network, CNN)网络模型 VGG19,测试对比 Intel MKL DNN 的卷积实现,最终取得了 2 倍多的性能加速比;另一方面,通过测试常用卷积类型,对比 Intel MKL DNN 和 NVIDIA cuDNN,验证了实现的 Winograd 对于常用卷积类型具有很好的适用性且具有实际使用价值.该研究工作期望为 Intel 平台在深度学习领域的发展提供重要的指导意义.

关键词 Winograd;深度学习;深度神经网络;卷积神经网络;向量化

中图法分类号 TP183

卷积神经网络(convolutional neural network, CNN)[1]由于感受野、权值共享和池化等特点,大大降低了神经网络训练需要的参数数目,使得更深层神经网络的使用成为可能,提高了训练的效率以及收敛的效果,成为了深度学习中最为典型且性能优越的一种网络模型.近年来,CNN 更是在多个方向持续发力[2],语音处理、文本提取、图像识别、通用物体识别、运动分析、自然语言理解甚至电脑波分析等方面均具有巨大突破[3-4].同时 CNN 也推动着人工智能的快速发展,ImageNet 大赛[5]、Facebook 图像自动标签、Google 无人驾驶汽车以及第 1 个战胜世界围棋冠军的人工智能程序 AlphaGo,这些无一不彰显着智能化时代到来的可能.对于 CNN,其灵魂就是卷积[6],而卷积不仅仅局限于 CNN,在很多其他网络模型中也都有涉及.由此可见,卷积性能的好坏对于整个深度学习领域有着非凡的影响.对于卷积,当前主流的算法有 GEMM[7],FFT[8-9],Winograd[10].这 3 种算法中除了 GEMM,其他 2 种都是以增加一定的操作复杂度为代价获取更低的计算复杂度,从而提升卷积的性能,其中又以 Winograd 做卷积时计算复杂度最低[7].

尽管如今对于深度学习的探索和创新层出不穷,但是这些研究多数是基于 NVIDIA GPU 平台[11-14].本文另辟蹊径,选择 Intel 平台作为研究环境,其主要原因是 Intel 是高性能计算的支柱之一,但进军深度学习领域时间较短,因而具有更大的潜力和挖掘空间.以卷积举例,目前 Intel 平台仅对于 GEMM 算法有较高的支持力度.

本次研究选择 Intel 第 2 代 Xeon Phi 处理器 KNL(knigths landing)7250 作为研究平台,结合该平台特性对 Winograd 快速卷积进行实现与优化.一方面通过测试 VGG19,对比 Intel MKL(math kernel library) DNN(deep neural network)中卷积性能,最终取得了 2 倍多的加速比;另一方面通过测试常用卷积类型,对比 Intel MKL DNN 和 NVIDIA cuDNN[7],证明了本文的 Winograd 算法对于常用卷积类型具有很好的适应性且具有实际使用价值.本次研究,我们并不是要提供一个功能完善的卷积接口,而是借此让更多的研究者看到 Intel 平台在深度学习领域的潜力和价值,为 Intel 平台在深度学习领域的发展打开一道里程碑式的大门,也期望能够为深度学习领域的蓬勃发展带去更多的动力.

1 背景知识

1.1 卷积神经网络

20 世纪 60 年代,Hubel 和 Wiesel 在研究猫的大脑皮层中用于局部敏感和方向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂度,继而提出了卷积神经网络.

CNN 相比于其他的神经网络最大的优势在于局部权值共享的特征,使其在数据量较大的图像识别、语音处理等方面只需要非常少的网络参数,在保证收敛结果的同时大大降低了网络的运行时间,提高了网络的运行效率.

卷积是 CNN 的核心,而卷积的过程就是依据局部感受野的原理对输入通过卷积核进行特征提取,最终得到特征总结后的输出.

我们将输入记为 $IN_{n,c,inh,inw}(1\leqslant n\leqslant N,1\leqslant c\leqslant C,1\leqslant inh\leqslant H_{\text{in}},1\leqslant inw\leqslant W_{\text{in}})$,表示 N 个图片、C 个通道以及图片大小为 $H_{\text{in}}\times W_{\text{in}}$;卷积核记为 $FLT_{k,c,r,s}(1\leqslant k\leqslant K,1\leqslant r\leqslant R,1\leqslant s\leqslant S)$,表示 K 个特征映射、C 个通道以及卷积核大小 $R\times S$;输出记为 $OUT_{n,k,outh,outw}(1\leqslant outh\leqslant H_{\text{out}},1\leqslant outw\leqslant W_{\text{out}})$,

表示 N 个图片、K 个特征映射以及图片大小 $H_{out}\times W_{out}$. 由此,可将卷积公式记录为[15]

$$OUT_{n,k,outh,outw}=\sum_{c=1}^{C}\sum_{r=1}^{R}\sum_{s=1}^{S}IN_{n,c,outh+r,outw+s}FLT_{k,c,r,s}. \tag{1}$$

我们可将式(1)简写为

$$\boldsymbol{OUT}_{n,k}=\sum_{c=1}^{C}\boldsymbol{IN}_{n,c}*\boldsymbol{FLT}_{n,k}, \tag{2}$$

其中,$*$ 表示卷积操作.

1.2　Winograd 算法

Winograd 算法是基于 1980 年 Winograd 的最小滤波算法(minimal filtering algorithm)提出的一种快速卷积算法[16],适用于小尺寸的卷积核做卷积,主要是通过降低卷积的计算复杂度从而提升卷积效率.

对于 mininal filtering algorithm,以下简称为 mini-filter 算法. 一维数据的情况下,当输出大小为 m,过滤器大小为 r,可将该算法记为 $F(m,r)$. 此时,算法运算需要 $\mu F((m,r))=m+r-1$ 次乘法操作[10].

一维的 mini-filter 算法可以表示成矩阵:

$$\boldsymbol{y}=\boldsymbol{A}^{\mathrm{T}}((\boldsymbol{Gw})\odot(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{x})), \tag{3}$$

其中,$\odot$ 表示 hadamard 乘积;$\boldsymbol{x}$ 表示输入,$\boldsymbol{w}$ 表示过滤器,$\boldsymbol{y}$ 表示输出;$\boldsymbol{A}^{\mathrm{T}},\boldsymbol{G},\boldsymbol{B}^{\mathrm{T}}$ 表示该算法的系数矩阵,其值由 m 和 r 决定,例如 $F(2,3)$ 有:

$$\boldsymbol{B}^{\mathrm{T}}=\begin{pmatrix}1&0&-1&0\\0&1&1&0\\0&-1&1&0\\0&1&0&-1\end{pmatrix},$$

$$\boldsymbol{G}=\begin{pmatrix}1&0&0\\\frac{1}{2}&\frac{1}{2}&\frac{1}{2}\\\frac{1}{2}&-\frac{1}{2}&\frac{1}{2}\\0&0&1\end{pmatrix},$$

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

从上述计算过程可以看出,其就是跨度为 1 的卷积操作. 由此,我们可以把一维的 mini-filter 看成是一维的卷积操作. 通过嵌套一维的 mini-filter,可以得到特殊情况下的二维 mini-filter 矩阵:

$$\boldsymbol{y}=\boldsymbol{A}^{\mathrm{T}}((\boldsymbol{GwG}^{\mathrm{T}})\odot(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{xB}))\boldsymbol{A}, \tag{4}$$

可将此时的 mini-filter 算法表示成 $F(m\times m,r\times r)$.

将上述计算过程应用到 CNN 中的卷积,也就是算法 1[10].

算法 1. Winograd 快速卷积算法($F(m\times m,r\times r)$).

输入:

T 是输入块的数量,大小为 $N\lceil H_{out}/m\rceil\lceil W_{out}/m\rceil$;

α^2 是输入块的尺寸,$\alpha=m+r-1$;

相邻块之间的覆盖为 $r-1$;

$\boldsymbol{x}_{c,b}$ 是索引为 c 的通道上索引为 b 的输入块;

$\boldsymbol{X}_{\mathrm{Tile}}$ 是转换后的输入重新布局得到的对应矩阵;

$\boldsymbol{w}_{k,c}$ 是索引为 k 的特征映射上索引为 c 的卷积核;

$\boldsymbol{W}_{\mathrm{Tile}}$ 是转换后的卷积核重新布局得到的对应矩阵;

$\boldsymbol{B}^{\mathrm{T}},\boldsymbol{G},\boldsymbol{A}^{\mathrm{T}}$ 分别是输入、卷积核、输出的转换系数;

$\boldsymbol{y}_{k,b}$ 是 k 通道上索引为 b 的输出块;

$\boldsymbol{Y}_{\mathrm{Tile}}$ 是转换后的输出重新布局得到的对应矩阵;

输出:$\boldsymbol{y}_{k,b}$, $\boldsymbol{Y}_{\mathrm{Tile}}$.

```
for k=0 to K do
  for c=0 to C do
    w_Tile = G w_{k,c} G^T;
    scatter(w_Tile)→W_Tile;
      /* 散播 w_Tile 到 W_Tile */
  end for
end for
for c=0 to C do
  for b=0 to T do
    x_Tile = B^T x_{c,b} B;
    scatter(x_Tile)→X_Tile;
      /* 散播 x_Tile 到 X_Tile */
  end for
end for
for ζ=0 to α do
  for ν=0 to α do
    Y_Tile^(ξ,ν) = W_Tile^(ξ,ν) X_Tile^(ξ,ν);
  end for
end for
for k=0 to K do
  for b=0 to T do
    gather(Y_Tile)→y_Tile; /* 收集 Y_Tile 中对应位
      置的数据,组成需要的输出块 y_Tile */
    y_{k,b} = B^T y_Tile B;
  end for
end for
```

2 算法优化

对于二维数据,Winograd 的卷积如式(4)所示,其中 $\boldsymbol{x}$ 表示输入块,$\boldsymbol{w}$ 表示卷积核,$\boldsymbol{y}$ 表示输出块,矩阵 $\boldsymbol{A}^{\mathrm{T}},\boldsymbol{G},\boldsymbol{B}^{\mathrm{T}}$ 则表示转换时的系数矩阵. 由此,本次研究中 Winograd 设计原理如图 1 所示.

假设输入维度为$[N,C,H_{\mathrm{in}},W_{\mathrm{in}}]$,卷积核的维度为$[K,C,r,r]$,填充为 0,跨度为 1,使用 Winograd 算法 $F(m\times m,r\times r)$. 为了方便描述,设 $t=m+r-1$.

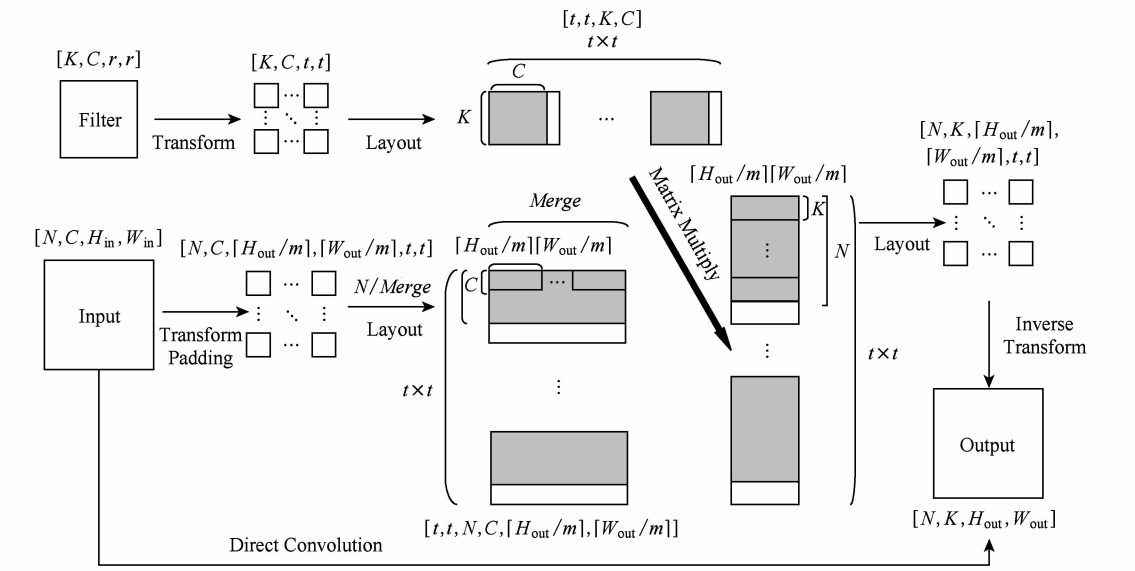


Fig. 1 The priciple of design for Winograd fast convolution

图 1 Winograd 快速卷积设计原理

步骤 1. 确定对应系数矩阵 $\boldsymbol{A}^{\mathrm{T}},\boldsymbol{G},\boldsymbol{B}^{\mathrm{T}}$,为了方便后续说明,我们假设 $\boldsymbol{w}_{\mathrm{Tile}}=\boldsymbol{G}\boldsymbol{w}\boldsymbol{G}^{\mathrm{T}},\boldsymbol{x}_{\mathrm{Tile}}=\boldsymbol{B}^{\mathrm{T}}\boldsymbol{x}\boldsymbol{B}$,所以 Winograd 算法变形为 $\boldsymbol{y}=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{w}_{\mathrm{Tile}}\odot\boldsymbol{x}_{\mathrm{Tile}})\boldsymbol{A}$.

步骤 2. 以 $t\times t$ 为模具对输入进行块映射,其中 2 个相邻的块间覆盖范围为 $(r-1)\times(r-1)$. 因此,每个通道上能够分出的块数量为 $\lceil H_{\mathrm{out}}/m\rceil\lceil W_{\mathrm{out}}/m\rceil$. 对每个块做 $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{x}\boldsymbol{B}$ 转换,并重排转换后的数据成一组矩阵. 这组矩阵由 t^2 个大矩阵组成,每个大矩阵又由 N 个 $C\lceil H_{\mathrm{out}}/m\rceil\lceil W_{\mathrm{out}}/m\rceil$ 的列优先的小矩阵组成,每个小矩阵索引为 $(t_{\mathrm{in}},n_{\mathrm{in}})$(其中 $1\leqslant t_{\mathrm{in}}\leqslant t^2,1\leqslant n_{\mathrm{in}}\leqslant N$).

步骤 3. 对每个卷积核做 $\boldsymbol{G}\boldsymbol{w}\boldsymbol{G}^{\mathrm{T}}$ 转换,并重排转换后的数据成 1 组矩阵. 这组矩阵由 t^2 个 $K\times C$ 的列优先的小矩阵组成,每个小矩阵索引为 (t_{flt})(其中 $1\leqslant t_{\mathrm{flt}}\leqslant t^2$).

步骤 4. 对步骤 2 和步骤 3 中由输入数据和卷积核转换重排得到的索引为 (t,n) 的输入数据小矩阵和索引为 (t) 的卷积核小矩阵做矩阵乘,最终得到 1 组输出矩阵. 该组矩阵由 t^2 个大矩阵组成,每个大矩阵又由 N 个 $K\lceil H_{\mathrm{out}}/m\rceil\lceil W_{\mathrm{out}}/m\rceil$ 的列优先的小矩阵组成,每个小矩阵索引为 $(t_{\mathrm{out}},n_{\mathrm{out}})$(其中 $1\leqslant t_{\mathrm{out}}\leqslant t^2,1\leqslant n_{\mathrm{out}}\leqslant N$). 最后对结果做对应数据 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}_{\mathrm{Tile}}\boldsymbol{A}$ 逆转换并重排得到维度为$[N,K,H_{\mathrm{out}},W_{\mathrm{out}}]$的输出数据.

出于计算效率的考虑,通过对输入和卷积核转换后的数据进行布局重排,将核心计算⊙转换为矩阵乘,调用 MKL 中的矩阵乘接口.

出于内存重用的考虑,每层卷积层使用共有的内存空间存储输入、卷积核和输出转换后数据,这样不仅能够节省大量的内存空间,同时也有利于提高数据的空间局部性.

2.1 虚拟化填充

在深度学习中,卷积层进行卷积时,大小为 1 的零填充是十分常见的现象,因为这样不仅能够保留输入的边缘特征,同时也能够防止输出大小缩减太快以至于神经网络模型层数不深.

对于传统零填充,往往需要开辟出一块新的内存,再将原始数据和填充数据按对应位置赋值给这段内存. 这种方式不仅访存代价高,而且会消耗大量内存.

为解决这一问题,本文提出了新的数据填充方式——虚拟化填充. 我们定义 2 种数据类型:实数据和虚数据,其中实数据来自原始输入,虚数据为填充时边缘填充为 0 的数据. 如图 2 所示,在模取输入数

据时,假设输入数据已经填充完成,通过对数据位置的判断,决定该位置是虚数据置 0,还是实数据对应原始输入数据值.

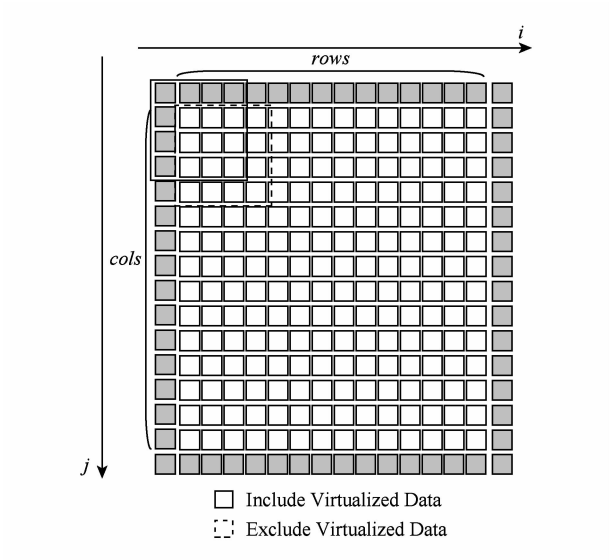


Fig. 2　Virtualized padding

图 2　虚拟化填充

2.2　分区填充

虚拟化填充解决了额外内存开销和填充后数据访存开销的问题,但是对于 Winograd 卷积中输入的转换部分,单纯的虚拟化填充需要在最内层循环中使用大量的分支判断语句来判定该位置是属于实数据还是虚数据,大大破坏了程序的连贯性.同时,分支预测失败会引起流水线空泡,降低程序的 IPC (instructions per cycle),影响程序的性能.

基于上述情况,提出了分区填充,旨在消除虚拟化填充中分支判断语句的存在,具体操作如图 3 所示.

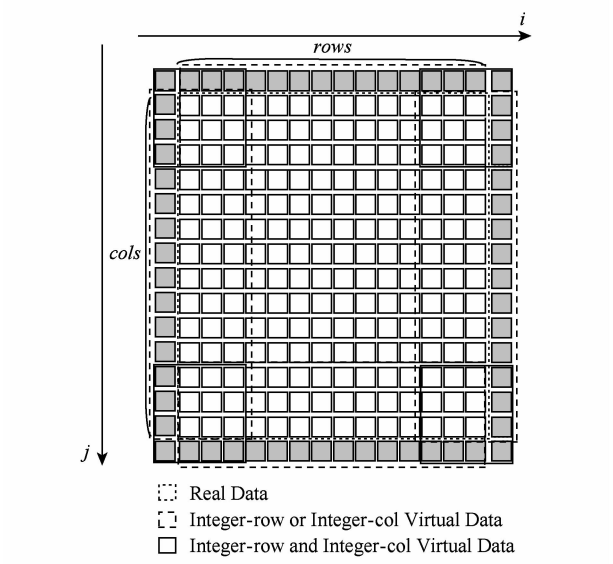


Fig. 3　Padding partition

图 3　分区填充

将输入数据划分为 3 个区域——块中所有数据都是实数据组成的区域、块中整数行或列是虚数据的区域、块中整数行且列都是虚数据的区域.通过对输入数据进行分区,可以消除分支判断语句,从而避免分支预测失败对程序整体连贯性的影响,提高指令级并行度,保证程序的执行效率.

2.3　填充规模策略

虚拟化填充结合分区填充的方式针对大规模的数据填充起到了非常好的效果.但当数据规模较小时,由于内存空间的需求不是很高,访存时间比较少,可以尝试使用传统填充方法.虽然会造成一定的内存浪费和访存开销,但是会大大降低操作的复杂度,反而更为有利.

为了更进一步提升传统填充的效率,可以从减少频繁的内存空间分配和释放入手.如图 4 所示,在输入数据转换前分配一块内存空间,大小为 $nt(H+2ph)(W+2pw)$,并将其通过线程索引进行分块绑定.对于某个线程,用于存储转换数据的内存空间大小为 $t(H+2ph)(W+2pw)$,起始地址为 $initAddr+tIndex\ (H+2ph)(W+2pw)$ (其中,nt 表示线程数,ph 和 pw 表示横纵向的填充大小,$initAddr$ 表示起始地址,$tIndex$ 表示线程索引).

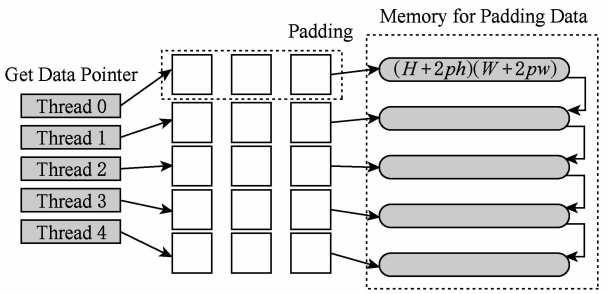


Fig. 4　The strategy for the scale of padding

图 4　填充规模策略

通过引入填充规模策略,针对不同规模的输入数据采用不同的填充方式:大规模数据采用虚拟化填充加分区填充,小规模数据采用优化后的传统填充,从而尽可能实现不同数据规模下的填充最优化.

2.4　合并策略

根据图 1 中 Winograd 快速卷积原理,在进行核心计算时,其本质是做 $t^2\times N$ 个小矩阵乘,其中每个卷积核的小矩阵会同 N 个输入数的小矩阵相乘,因此可以考虑将这 N 个输入数据小矩阵合并到

一起变成一个大矩阵再同卷积核的小矩阵做矩阵乘,从而增加矩阵乘的规模,提升矩阵运算的效率.

因为在做核心计算时的矩阵乘是并行执行的,一个线程负责数个矩阵乘.因此如图 5 所示,在做合并时,可以选择不同尺度的合并.以 *Merge* 值为合并尺度,从而将 N 个输入小矩阵合并为 $N/Merge$ 个 $C\lceil H_{out}/m\rceil\lceil W_{out}/m\rceil$ *Merge* 的大矩阵,同对应的卷积核小矩阵做矩阵乘.一方面兼顾了并行时的多线程粒度,另一方面也兼顾了 MKL 矩阵乘接口的性能.

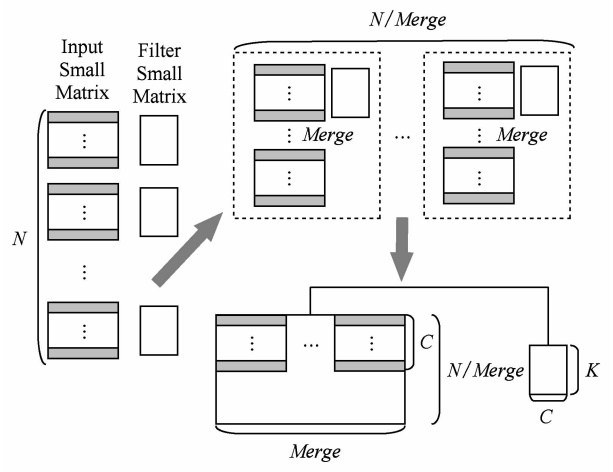


Fig. 5 Merge strategy

图 5 合并策略

2.5 不同尺度 Winograd 算法混合

Winograd 快速卷积算法在某种意义上可以看成是传统卷积的扩展,不同点在于传统卷积在计算 Output 数据时是一次 1 个元素(对于二维数据,可以看成 1×1 的块),而 Winograd 则是一次 n 个元素(其中 $n>1$,同样对于二维数据,可以看成 $n\times n$ 的块).

通过 $F(2\times2,3\times3)$同 MKL 卷积的比较中我们知道,两者对于不同规模的卷积各有优势.由此,不难猜想通过扩展 Winograd 算法的尺度,可以更好地适应不同规模的卷积(对于 Winograd 算法 $F(m\times m,r\times r)$,m 值的大小决定着算法的尺度).

本次研究中使用 3 种不同尺度的 Winograd 算法,分别为 $F(2\times2,3\times3)$,$F(3\times3,3\times3)$,$F(4\times4,3\times3)$.在混合使用时,可以分为结构级混合和数据级混合,前者是指当前卷积层选用 3 种 Winograd 算法中的一种做卷积,后者是指当前卷积层的卷积分成多个部分,每个部分分别使用 3 种 Winograd 算法中的一种.

2.6 分离转换替换统一转换

Winograd 算法中,对于输入、卷积核和输出的转换可以表述为

$$\boldsymbol{x}_{\text{Tile}}=\boldsymbol{B}^{\text{T}}\boldsymbol{x}\boldsymbol{B},$$

$$\boldsymbol{w}_{\text{Tile}}=\boldsymbol{G}\boldsymbol{w}\boldsymbol{G}^{\text{T}},$$

$$\boldsymbol{y}=\boldsymbol{A}^{\text{T}}\boldsymbol{y}_{\text{Tile}}\boldsymbol{A},$$

其中,$\boldsymbol{A}^{\text{T}}$,$\boldsymbol{G}$,$\boldsymbol{B}^{\text{T}}$ 为 Winograd 算法的转换系数,是由 $F(m\times m,r\times r)$确定的已知数组.因此,在进行数据转换时,可以分为 2 种转换方式:分离转换和统一转换.

以输入数据为例说明,统一转换是将 Winograd 算法 $\boldsymbol{B}^{\text{T}}\boldsymbol{x}\boldsymbol{B}$ 转换过程中的 2 次矩阵乘合并到一起,以输入模取的数据作为未知量 $\boldsymbol{x}$,由此获取转换后的数据 $\boldsymbol{x}_{\text{Tile}}$关于 $\boldsymbol{x}$ 的线性关系.

以输入数据为例说明,分离转换是将 Winograd 算法转换过程中的 2 次矩阵乘分开进行,以输入模取的数据为未知量 $\boldsymbol{x}$,由此获取第 1 次矩阵运算后得到的桥数据 $\boldsymbol{b}$ 关于 $\boldsymbol{x}$ 的线性关系;再以 $\boldsymbol{b}$ 为未知量,得到转换后的数据 $\boldsymbol{x}_{\text{Tile}}$关于 $\boldsymbol{b}$ 的线性关系.

对于统一转换,计算量少,但是所有的线性关系呈横向锯齿形,杂乱无规律,难以优化.对于分离转换,计算量较多,但所有的线性关系呈平面方形,平整有序,非常有利于向量化操作.

本次研究中,将前期的统一转换用分离转换替换,可以大大提升程序的性能.

2.7 混合统一批处理和分块批处理

随着 batch 的变大,处理的数据变得越来越多,使用到的内存也变得越来越大.当该层实际使用到的内存超出 MCDRAM 大小 16 GB 时,程序性能将会急速下降.

因此,对于大 batch 的卷积层,可以将 batch 分块处理,从而保证程序的性能.依据此,可以将 batch 的处理分为 2 种方式:统一批处理和分块批处理,前者是一次性处理所有 batch,后者是将 batch 分块再处理.

分块的性能并不总是最优的,对于小 batch,在进行 Winograd 卷积时,实际使用到的内存也会很小,此时如果仍然使用分块批处理,反而会因为无法发挥 MCDRAM 的优势以及多线程的效率,导致程序性能下降.因此,Winograd 算法卷积实际使用到的内存在不超过 MCDRAM 大小时,选用统一批处理;实际使用到的内存超过 MCDRAM 大小时,选用分块批处理.

3　结果与分析

3.1　Intel KNL

Intel KNL 是 Intel 第 2 代 MIC 架构 Xeon Phi 至强融合处理器[17],是 Intel 首款专门针对高度并行工作负载而设计的可独立自启动的处理器,可继续做协处理器,也可独立做中央处理器.

如图 6 所示,Intel KNL 处理器架构衍生于 Intel Atom 处理器,采用 Silvermont 架构的改进定制版和 14 nm 工艺,由 38 个 Tile 组成(最多有 36 个 Tile 处于活动状态),核心数量可达 72 个,最大支持线程数为 288,双精度浮点性能超 3 TFLOPS,单精度浮点性能超 6 TFLOPS. 每个 Tile 包含 2 个 Core,每个 Core 拥有 2 个向量化处理单元 VPU,相同 Tile 里面的 2 个 Core 共享 1 块大小为 1 MB 的 L2 缓存.

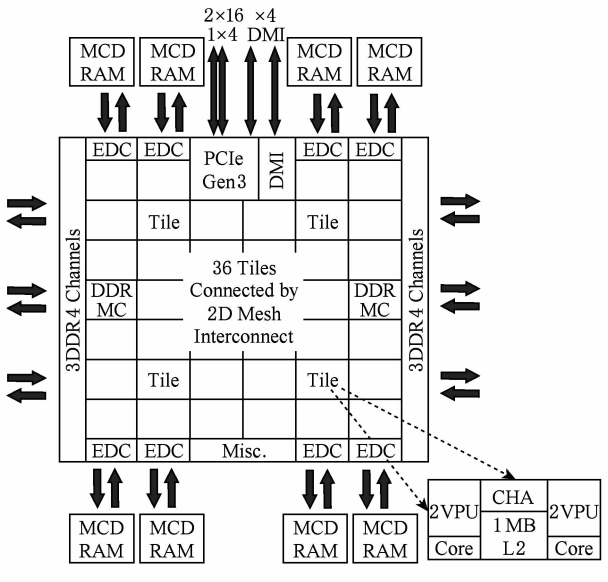


Fig. 6　The structure of Intel KNL processor[18]

图 6　Intel KNL 处理器架构[18]

Intel KNL 相比于第 1 代 Xeon Phi 至强融合处理器,为了能够获取更高的性能,在架构方面有着非常显著的改变. Intel KNL 的核布局不再是传统的环形布局,而变成了二维网格状布局,可以说是从线的布局方式提升到了面的布局方式,在进行核与核之间交互时具备了更高的效率;内存分成 2 部分:1) 2 个 3 通道的 DDR4,每个通道可承担 64 GB DIMMs,总容量可达 384 GB;2) 8 个 2 GB 的 MCDRAM 设备,共 16 GB. 其中 DDR4 的访存带宽是 90 GBps,MCDRAM 带宽远超 DDR4,可达 450 GBps;MCDRAM 既可做内存也可做缓存,据此,在启动设置时可以有 3 种模式:1) 当 MCDRAM 做内存使用时,为 Flat 模式;2) 当 MCDRAM 做缓存使用时,为 Cache 模式;3) 当 MCDRAM 一部分做内存使用且一部分做缓存使用时,为 Hybrid 模式;单节点 KNL 可模拟成多节点,称之为 SNC 模式,分别是 No SNC(All2All, Heimisphere, Quadrant), SNC-2, SNC-4;支持 AVX-512 指令集,向量化操作更加完善.

本次研究中使用的是 Intel KNL 7250,配置如表 1 所示:

Table 1　Configure Parameters for Intel KNL 7250

表 1　Intel KNL 7250 配置参数

Parameter	Value
CPU	Intel® Xeon Phi™ Processor 7250
CPU Frequency/GHz	1.40
Cores	68
Threads/Core	4
L1d Cache/KB	32
L1i Cache/KB	32
L2 Cache/KB	1 024
MCDRAM/GB	16
DDR4/GB	98
SP/TFLOPS	3+
DP/TFLOPS	6+

3.2　性能测试

为了更好地验证本文 Winograd 的实现效果,故将实验过程分为 3 个阶段:

1) 选择 VGG19 作为测试网络,验证 Winograd 各级优化方案累加效果.

2) 同样选用 VGG19 作为测试网络,测试 Intel KNL 不同 SNC 模式和 Memory 模式对本文 Winograd 卷积性能的影响.

3) 抽取典型卷积网络模型 AlexNet,GoogleNet,ResNet,VGG11,VGG16,VGG19 中卷积层的卷积参数,测试本文的 Winograd,Intel MKL DNN,NVIDIA cuDNN 三者的卷积性能,对比验证本文 Winograd 对于常用卷积类型的适用性以及是否具有实际使用价值.

3.2.1 各级优化累加

通过测试 VGG19 中卷积在 Winograd 算法各级优化累加下的运行时间,对比 Intel MKL DNN,如图 7 所示. 其中,柱状图显示的是运行的总时间大小,折线图显示的是 Winograd 的加速比.

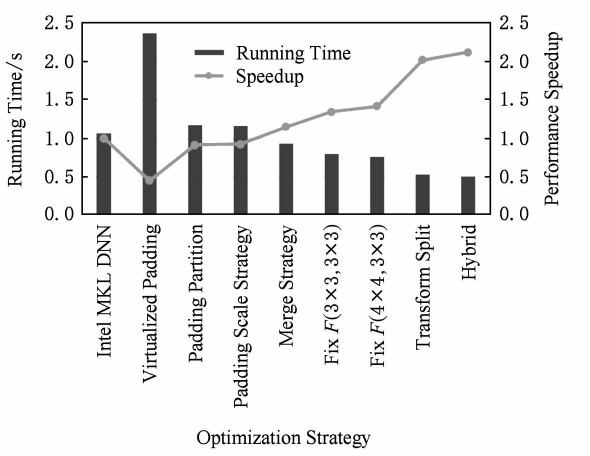


Fig. 7 Comparison of the whole running time about VGG19 forward

图 7 VGG19 卷积层前向运行总时间对比

由图 7 可以看出:虚拟化填充在避免传统填充方式额外内存开销和访存弊端的同时引入大量分支判断,严重破坏了程序的连贯性,使得程序的整体性能很差,加速比仅有 0.45;分区填充避免了填充虚拟化造成的大量分支判断,大大提升了程序的性能,使得加速比达到 0.91;填充规模策略考虑小规模卷积使用优化后的传统填充方式,此时额外的内存开销和访存不会对程序性能造成太大的影响,通过该策略加速比达到 0.92;合并策略通过配置最佳的合并尺度,充分发挥了矩阵乘的性能,加速比达到 1.15;不同尺度 Winograd 算法对于不同规模的卷积具有更好的适应性,通过混合 $F(3\times3,3\times3)$,加速比达到 1.34. 通过混合 $F(4\times4,3\times3)$,加速比达到 1.41;分离转换将数据转换分成 2 部分,使得转换过程具有工整对齐的计算公式,向量化得以充分发挥,加速比达到 2.01;理论上,如果将 Winograd 快速卷积同 Intel MKL DNN 混合使用,最佳加速比可以达到 2.11.

3.2.2 混合模式

Intel KNL 新的架构特征,在很大程度上提升了访存的性能. 同时,为了进一步提高核与内存、核与核之间的交互效率,加入了 Memory 模式和 SNC 模式[18].

Memory 模式又分为 Flat,Cache,Hybrid,取决于 MCDRAM 在处理器中的角色(memory 还是 cache). SNC 模式又分为 No SNC(All2All,Heimisphere, Quadrant),SNC-2,SNC-4,取决于 KNL 处理器 NUMA 后的节点数.

通过混合 2 种模式,我们期望找出 Winograd 算法的最适情况. 如表 2 所示:

Table 2 The Performance of Winograd with Different Modes

表 2 不同模式下 Winograd 性能 ms

Memory Mode	SNC Mode				
	Quadrant	All2All	Heimisphere	SNC-2	SNC-4
Flat	526.82	533.89	529.87	644.38	1835.36
Cache	526.91	533.01	529.57	557.24	616.69
Hybrid	531.34	539.14	534.39	642.27	1814.27

SNC-2 以及 SNC-4 模式下,程序性能均有很明显的下降,其他模式下性能略有差距. 对于本文的 Winograd 算法,Intel KNL 的最适 SNC 模式是 Quadrant,对于 Memory 模式,Flat 和 Cache 差距极小,考虑到误差的可能,可以认为两者皆为最佳模式.

3.2.3 常用卷积类型

为了进一步验证本文的 Winograd 算法对于常用卷积是否具有实际意义,我们抽取典型卷积网络模型 AlexNet,GoogleNet,ResNet,VGG11,VGG16,VGG19 中卷积层的卷积参数,测试对比本文 Winograd,Intel MKL DNN,NVIDIA cuDNN 的卷积性能.

在 GPU 的选择上,我们选取的是 Intel KNL 的同代 GPU Tesla M40. 因此在对比时要综合考虑硬件平台本身性能的差距,其中 KNL7250 的单精浮点性能是 6.1 TFLOPS,Tesla M40 的单精浮点性能是 7 TFLOPS,所以时间计算为

$$Time_{\text{Intel MKL DNN}}=Time_{\text{Intel MKL DNN}}\times\frac{6.1}{7},$$

$$Time_{\text{NVIDIA cuDNN}}=Time_{\text{NVIDIA cuDNN}},$$

$$Time_{\text{Winograd}}=Time_{\text{Winograd}}\times\frac{6.1}{7}.$$

考虑不同卷积运行时间差距太大,可以将 Intel MKL DNN 的卷积时间作为标准,从而直接对比三者间的性能比

$$Performance_{\text{Intel MKL DNN}}=Time_{\text{Intel MKL DNN}}/Time_{\text{Intel MKL DNN}},$$

$$Performance_{\text{NVIDIA cuDNN}}=Time_{\text{Intel MKL DNN}}/Time_{\text{NVIDIA cuDNN}},$$

$$Performance_{\text{Winograd}}=Time_{\text{Intel MKL DNN}}/Time_{\text{Winograd}}.$$

我们总共抽取了 46 种不同规模的卷积,并按照计算量从小到大排列,分别就无填充和填充 2 种情况做测试,结果如图 8 和图 9 所示:

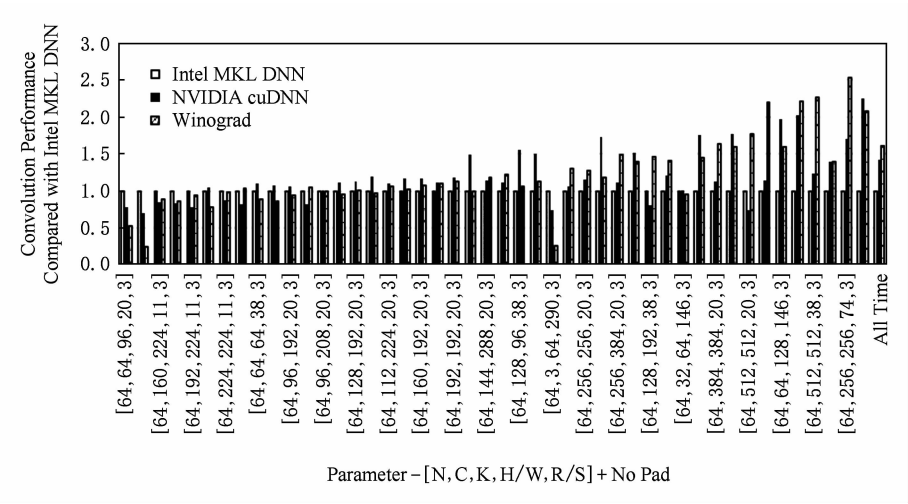


Fig. 8　Different convolution performance comparison with no pad

图 8　无填充下的不同卷积性能对比

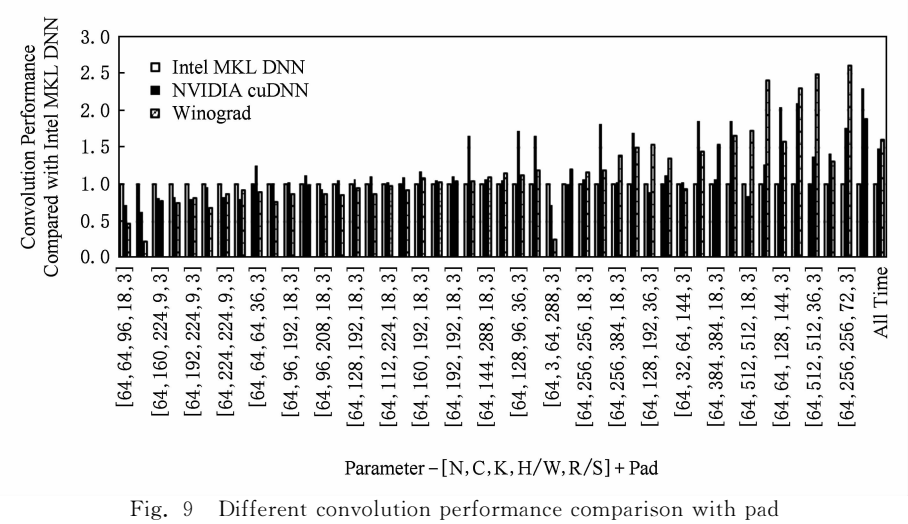


Fig. 9　Different convolution performance comparison with pad

图 9　填充下的不同卷积性能对比

从总体卷积来看,无填充和填充情况下,本文的 Winograd 算法相比 Intel MKL DNN 有 61% 和 60%的性能提升,相比 NVIDIA cuDNN 也有 13% 和 8%的性能提升.

从单个卷积来看,无填充和填充情况下,有性能提升的卷积占总数的 67.4%和 58.7%,性能相当的卷积占总数的 17.4% 和 13%;卷积的规模越大,Winograd 算法的性能往往越好.

综上可得,本文的 Winograd 快速卷积对于大多数常用卷积类型具有性能提升,且大规模卷积性能提升最为明显.在对比 Intel MKL DNN 时具有明显优势,即使对比 NVIDIA cuDNN 也是略胜一筹,可见本文的 Winograd 快速卷积具有实际使用价值.

4　总结展望

通过在 Intel KNL 平台上对 Winograd 快速卷积

的研究与优化,一方面以 VGG19 作为测试网络,对比 Intel MKL DNN,最终提升了 1 倍多(加速比 2.01)的卷积性能;另一方面,测试常用卷积类型性能,对比 Intel MKL DNN 和 NVIDIA cuDNN,证明了本文的 Winograd 算法对于常用卷积类型具有很好的适用性且具有实际使用价值.但这并不是结束,相反这仅仅是一个开始,未来仍有许多的路要走,比如:提升算法对于小规模卷积的适用性;合并策略在进行尺度判断时,仍需要手动配置,无法自动决定最佳的合并尺度;统一批处理和分块批处理的判断仍然需要进行大量测试以决定最终的边界,简单地以 MCDRAM 的大小作为边界并不是最合适的方案;Intel MKL DNN 和 Winograd 最适情况的标准仍需研究界定等.同时,基于算法的角度考虑,不同尺度的 Winograd 算法对于不同规模的卷积适应性更强,可以从这一方面出发,通过进一步扩大 Winograd 算法的尺度提升性能;基于 Intel 平台的考虑,可以从数据预取出发,进一步提升算法性能.未来我们将从这些方面入手,对 Winograd 算法做进一步的完善.

本文研究工作并不是去优化一个完善的快速卷积算法,更多的是彰显 Intel 平台在深度学习领域的潜力和价值,为其后续发展提供重要的指导意义,借此为人工智能领域的蓬勃发展贡献一份力量.

参 考 文 献

[1] Gu Jiuxiang, Wang Zhenhua, Kuen J, et al. Rencent advance in convolutional neural networks [J]. Journal of Pattern Recognition, 2018, 77(2): 354-377

[2] Zhou Feiyan, Jin Linpeng, Dong Jun. Review of convolutional neural network [J]. Chinese Journal of Computers, 2017, 40(6): 1229-1251 (in Chinese)
(周飞燕, 金林鹏, 董军. 卷积神经网络研究综述[J]. 计算机学报, 2017, 40(6): 1229-1251)

[3] Wang Jichen, Lin Jun, Wang Zhongfeng. Efficient convolution architectures for convolutional neural network [C] //Proc of the 8th IEEE Wireless Communications & Signal Processing. Piscataway, NJ: IEEE, 2016: 1-5

[4] Zhang Yong, Er M J, Pratama M, et al. Extractive document summarization based on convolutional neural networks [C] //Proc of the 42nd Annual Conf of the IEEE Industrial Electronics Society. Piscataway, NJ: IEEE, 2016: 918-922

[5] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks [C] //Proc of the 26th Neural Information Processing Systems. Cambridge, MA: MIT Press, 2012: 1106-1114

[6] Jaderberg M, Vedaldi A, Zisserman A. Speeding up convolutional neural networks with low rank expansions [C] //Proc of British Machine Vision Conf. Cambridge, UK: BMVA, 2014: 073

[7] CoRR. cuDNN: Efficient primitives for deep learning [OL]. [2018-01-03]. https://arxiv.org/pdf/1410.0759.pdf

[8] Nguyen-Thanh N, Le-Duc H, Ta D T, et al. Energy efficient techniques using FFT for deep convolutional neural networks [C] //Proc of IEEE Advanced Technologies for Communications. Piscataway, NJ: IEEE, 2016: 231-236

[9] CoRR. Fast training of convolutional networks through FFTs [OL]. [2017-07-12]. https://arxiv.org/pdf/1312.5851.pdf

[10] Lavin A, Gray S. Fast algorithms for convolution neural networks [C] //Proc of the 22nd IEEE Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2016: 4013-4021

[11] St-Charles P L, Biodeau G A, Bergevin R. Fast image gradients using binary feature convolutions [C] //Proc of the 22nd IEEE Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2016: 1074-1081

[12] Zhang Xiangyu, Zou Jianhua, Ming Xiang, et al. Efficient and accurate approximations of nonlinear convolutional networks [C] //Proc of the 21st IEEE Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE, 2015: 1984-1992

[13] Wang Min, Liu Baoyuan, Foroosh B. Factorized convolutional neural network [C] //Proc of IEEE Int Conf on Computer Vision. Piscataway, NJ: IEEE, 2017: 545-553

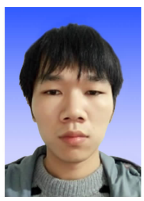
[14] Jiang Wenbin, Chen Yiming, Zheng Ran, et al. A novel GPU-based efficient approach for convolutional neural networks [J]. Journal of Signal Processing Systems, 2017, 86(3), 313-325

[15] Cong Jason, Xiao Bingjun. Minimizing computation in convolutional neural networks [C] //Proc of the 24th Artificial Neural Networks. Berlin: Springer, 2014: 281-290

[16] Selesnick I W, Burrus C S. Extending Winograd's small convolution algorithm to longer lengths [C] //Proc of IEEE Int Symp on Circuits and Systems. Piscataway, NJ: IEEE, 1994: 449-452

[17] Jeffers J, Reinders J, Sodani A. Intel® Xeon Phi™ Processor High Performance Programming [M]. Burlington: Morgan Kaufmann, 2016: 46-50

[18] Sodani A. Knights landing (KNL): 2nd generation Intel® Xeon Phi™ processor, 16124655 [R]. Piscataway. NJ: IEEE, 2015



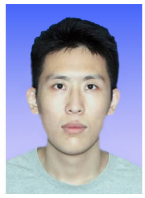
Wu Zheng, born in 1992. PhD candidate. His main research interests include parallel compute system architecture, and machine learning.

An Hong, born in 1963. PhD, professor, PhD supervisor. Her main research interests include chip multiprocessor architecture, parallel compute system architecture, parallel programming environment and tools, large data parallel storage and processing, and so on. (han@ustc. edu. cn)

Jin Xu, born in 1992. PhD candidate. His main research interests include machine learning and distributed system. (jinxu@mail. ustc. edu. cn)

Chi Mengxian, born in 1991. PhD candidate. His main research interests include machine learning, parallel computing, and micro-architecture. (mxchi10@mail. ustc. edu. cn)

Lü Guofeng, born in 1995. Master candidate. His main research interests include machine learning, parallel computing, and micro-architecture. (lvguofeng@mail. ustc. edu. cn)



Wen Ke, born in 1991. Master candidate. His main research interests include machine learning and parallel computing. (wenke51@mail. ustc. edu. cn)

Zhou Xin, born in 1993. Master candidate. Her main research interests include parallel programming and optimizing, deep neural network. (zxustc@mail. ustc. edu. cn)