基于FPGA的卷积神经网络硬件加速器设计

秦华标* 曹钦平

(华南理工大学电子与信息学院 广州 510641)

摘 要：针对卷积神经网络(CNN)计算量大、计算时间长的问题，该文提出一种基于现场可编程逻辑门阵列(FPGA)的卷积神经网络硬件加速器。首先通过深入分析卷积层的前向运算原理和探索卷积层运算的并行性，设计了一种输入通道并行、输出通道并行以及卷积窗口深度流水的硬件架构。然后在上述架构中设计了全并行乘法-加法树模块来加速卷积运算和高效的窗口缓存模块来实现卷积窗口的流水线操作。最后实验结果表明，该文提出的加速器能效比达到32.73 GOPS/W，比现有的解决方案高了34%，同时性能达到了317.86 GOPS。

关键词：卷积神经网络；硬件加速；现场可编程逻辑门阵列；计算并行；深度流水

中图分类号：TP331 **文献标识码**：A **文章编号**：1009-5896(2019)11-2599-07

DOI: 10.11999/JEIT190058

Design of Convolutional Neural Networks Hardware Acceleration Based on FPGA

QIN Huabiao CAO Qinping

(*School of Electronics and Information Engineering, South China University of Technology, Guangzhou* 510641, *China*)

Abstract: Considering the large computational complexity and the long-time calculation of Convolutional Neural Networks (CNN), an Field-Programmable Gate Array(FPGA)-based CNN hardware accelerator is proposed. Firstly, by deeply analyzing the forward computing principle and exploring the parallelism of convolutional layer, a hardware architecture in which parallel for the input channel and output channel, deep pipeline for the convolution window is presented. Then, a full parallel multi-addition tree is designed to accelerate convolution and efficient window buffer to implement deep pipelining operation of convolution window. The experimental results show that the energy efficiency ratio of proposed accelerator reaches 32.73 GOPS/W, which is 34% higher than the existing solutions, as the performance reaches 317.86 GOPS.

Key words: Convolutional Neural Networks (CNN); Hardware acceleration; FPGA; Parallel computation; Deep pipeline

1 引言

近年来，卷积神经网络(Convolutional Neural Network, CNN)在人工智能领域得到了广泛的应用[1]。但是CNN在不断逼近任务精度极限的同时，其网络深度与参数的数量也在迅速地增长，越来越耗费计算资源与内存资源[2]。

目前实现卷积神经网络加速的平台主要有3种：图形处理器(Graphics Processing Unit, GPU)，其软件可编程和多CUDA架构非常适合加速卷积神经网络，但其显著的功耗使其难以集成到功耗受限的嵌入式平台[3]；专用集成电路(Application-Specific Integrated Circuit, ASIC)，具有高性能、低功耗的特点，但设计周期长，制造成本高[4]；现场可编程逻辑门阵列(Field-Programmable Gate Array, FPGA)，具有低功耗和高灵活性的特点，从而成为研究卷积神经网络硬件加速最热门的平台。

基于FPGA的卷积神经网络加速器设计的研究，主要集中在并行计算和内存带宽优化两个方面。在并行计算方面，Motamedi等人[5]详细总结了卷积层并行计算优化方法，提出了输出间并行、卷积核间并行和卷积核内并行3种可并行计算方法。如果FPGA的面积、带宽和片上存储都不受限制，则理论上可利用上述所有方法来最大地加速神经网络，但实际上是不能的。因此，面临的挑战在于研

收稿日期：2019-01-22；改回日期：2019-06-10；网络出版：2019-06-20
*通信作者： 秦华标 eehbqin@scut.edu.cn
基金项目：广东省科技计划项目(2014B090910002)
Foundation Item: The Science and Technology Project of Guangdong Provience (2014B090910002)

究多种并行机制的最佳组合。在内存带宽优化方面，Zhang等人[6]先将输入特征图与权重数据全部存储在BRAM中，再从中读取数据进行卷积运算，虽然对权重数据进行了重用，但没有对输入特征图的卷积窗口进行重用，内存利用率不高。此外以上所有设计在并行计算时采用了大量的经典加法树单元，所需的带宽大、占用资源多。

为了解决以上问题，本文完成了以下研究工作：

(1) 分析了卷积层并行加速可行性，提出了卷积内并行、输入通道并行和输出通道并行3种并行计算方法；

(2) 提出了一种新的加法树设计方法，设计了全并行乘法-加法树模块，减少了计算资源与内存资源并保持相同的计算性能；

(3) 对卷积窗口采用流水线操作，设计的高效窗口缓存模块每个时钟周期生成一个卷积窗口，从而提高流水的性能；

(4) 设计了卷积层硬件加速器，该加速器对输入通道并行、输出通道并行以及卷积窗口进行流水操作，极大地提高了计算性能。

2 卷积层前向运算原理

卷积神经网络中的计算主要集中在卷积层[7]，图1为卷积层运算的整体过程。

图1中输入特征图$\boldsymbol{X}$是一个3维矩阵，其形状为$[N,H,W]$，其中N为输入特征图的通道数，H和W分别为输入特征图的高和宽。卷积核$\boldsymbol{W}$是一个4维矩阵，其形状为$[M,N,H_{\mathrm{k}},W_{\mathrm{k}}]$，其中$M$为卷积核的个数，$N$为卷积核的通道数即输入特征图通道数，$H_{\mathrm{k}}$和$W_{\mathrm{k}}$卷积核的高和宽。卷积核在垂直方向与水平方向的步长分别为H_{s}和W_{s}。偏置是一个长度为M的1维向量记为$\boldsymbol{b}$。输出特征图$\boldsymbol{O}$是一个3维矩阵，其形状为$[M,H_{\mathrm{o}},W_{\mathrm{o}}]$，其中$M$为输出特征图的通道数即卷积核的个数，$H_{\mathrm{o}}$和$W_{\mathrm{o}}$分别为输出特征图的高和宽。有关系式为

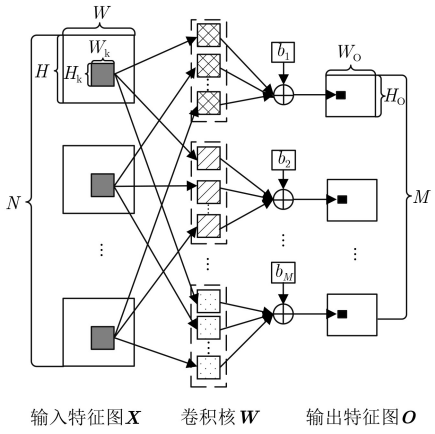

图 1 卷积层运算过程

$$H_o=\left\lfloor\frac{H-H_{\mathrm{k}}}{H_{\mathrm{s}}}\right\rfloor+1 \tag{1}$$

$$W_{\mathrm{o}}=\left\lfloor\frac{W-W_{\mathrm{k}}}{W_{\mathrm{s}}}\right\rfloor+1 \tag{2}$$

卷积输出特征图第m个通道第i'行第j'列数据$O_{mi'j'}$满足

$$O_{mi'j'}=\sum_{n=1}^{N}\sum_{i=1}^{H_{\mathrm{k}}}\sum_{j=1}^{W_{\mathrm{k}}}X_{n(i+(i'-1)H_{\mathrm{s}})(j+(j'-1)W_{\mathrm{s}})}\cdot W_{mnij}+b_m \tag{3}$$

其中，$i'\in[1,H_{\mathrm{o}}],j'\in[1,W_{\mathrm{o}}],m\in[1,M]$。

若某个卷积核的通道大小为3×3，输入特征图的通道大小为5×5，垂直与水平方向的步长都为2，则其得到2×2大小输出通道分量的步骤如图2所示。

3 卷积层硬件加速设计

3.1 卷积层并行加速可行性分析

由于每个卷积窗口的运算方式一样，从而针对位于输入特征图左上角的卷积窗口，即$i'=1,j'=1$，式(3)可以改写为

$$O_m=\sum_{n=1}^{N}\sum_{i=1}^{H_{\mathrm{k}}}\sum_{j=1}^{W_{\mathrm{k}}}X_{nij}W_{mnij}+b_m \tag{4}$$

其中，$m\in[1,M]$，该卷积窗口对应的输出值有M个，分别是$O_1,O_2,\cdots,O_M$。图3为上述计算过程的示意图。

根据式(4)，令

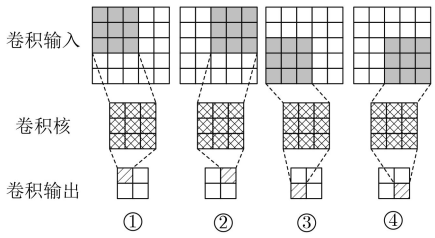

图 2 1个输入通道的卷积运算过程

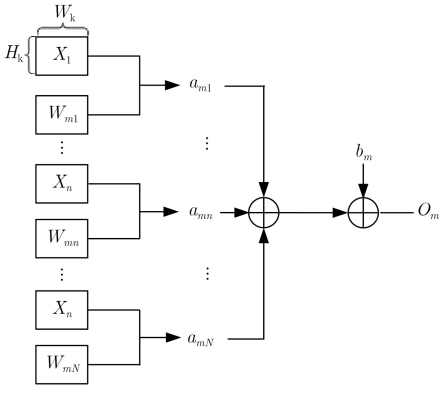

图 3 N个输入通道的卷积窗口并行计算

$$a_{mn}=\sum_{i=1}^{H_{\mathrm{k}}}\sum_{j=1}^{W_{\mathrm{k}}}X_{nij}W_{mnij} \tag{5}$$

则有

$$O_m=\sum_{n=1}^{N}a_{mn}+b_m \tag{6}$$

计算$O_1, O_2, \cdots, O_M$这M个值，可以分别采用以下两种方法：

(1) 由式(6)可知，先计算出$a_{m1}, a_{m2}, \cdots, a_{mN}$，然后对这$N$个数求和再加上$b_m$，得到$O_m$，令$m$取$[1, M]$，就可以计算出$O_1, O_2, \cdots, O_M$。

(2) 由式(6)可知，

$$\begin{aligned}\boldsymbol{O}&=\begin{bmatrix}O_1\\O_2\\\vdots\\O_M\end{bmatrix}=\begin{bmatrix}a_{11}\\a_{21}\\\vdots\\a_{M1}\end{bmatrix}+\begin{bmatrix}a_{12}\\a_{22}\\\vdots\\a_{M2}\end{bmatrix}+\cdots\\&+\begin{bmatrix}a_{1N}\\a_{2N}\\\vdots\\a_{MN}\end{bmatrix}+\begin{bmatrix}b_1\\b_2\\\vdots\\b_M\end{bmatrix}\\&=\hat{\boldsymbol{O}}_1+\hat{\boldsymbol{O}}_2+\cdots+\hat{\boldsymbol{O}}_N+\boldsymbol{b}\end{aligned} \tag{7}$$

其中，$\hat{\boldsymbol{O}}_n$为卷积输出$\boldsymbol{O}$的第n个分量，满足

$$\hat{\boldsymbol{O}}_n=\begin{bmatrix}a_{1n}\\a_{2n}\\\vdots\\a_{Mn}\end{bmatrix} \tag{8}$$

其中，$n\in[1, N]$。

如图4所示，使用M个累加器，先求出$a_{11}, a_{21}, \cdots, a_{M1}$并分别存储到$M$个寄存器中，再求出并累加上$a_{12}, a_{22}, \cdots, a_{M2}$，到第$N$次时求出并累加上$a_{1N}, a_{2N}, \cdots, a_{MN}$，最后加上偏置$b_1, b_2, \cdots, b_M$，就可以求出$O_1, O_2, \cdots, O_M$。

由此可见卷积层中可并行计算的部分有3个：

(1) 式(5)中，$H_{\mathrm{k}}W_{\mathrm{k}}$个乘法可以并行计算，此并行位于卷积核内部，称为卷积内并行；

(2) 式(6)中，在计算第m个卷积输出O_m时，与其对应N个输入通道的卷积可以并行计算，并得到$a_{m1}, a_{m2}, \cdots, a_{mN}$这$N$个中间结果，最后将$N$个值求和得到$O_m$。此为输入通道并行；

(3) 式(7)中，在计算卷积输出$\boldsymbol{O}$的第n个分量$\hat{\boldsymbol{O}}_n$时，第n个输入通道与M个卷积核中对应通道的卷积可以并行计算，此为输出通道并行。

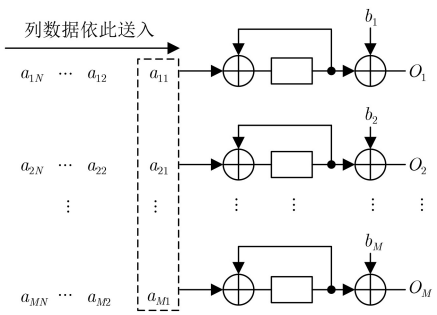

图 4 累加器并行运算

3.2 基本模块设计

3.2.1 全并行乘法-加法树模块

由式(3)可知，卷积运算每个输出通道的每个卷积窗口含有$N\times H_{\mathrm{k}}\times W_{\mathrm{k}}$个乘法，所以整个卷积层有$M\times G\times N\times H_{\mathrm{k}}\times W_{\mathrm{k}}$个乘法，式(3)中相应的求和符号即是加法计算，且每个输出通道的每个窗口需要对$N\times H_{\mathrm{k}}\times W_{\mathrm{k}}+1$个数求和，所以整个卷积层需要对$M\times G\times(N\times H_{\mathrm{k}}\times W_{\mathrm{k}}+1)$个数求和。从而必须对此类乘加运算进行并行优化。

卷积核形状一般为正方形，假设卷积核大小$W_{\mathrm{k}}=H_{\mathrm{k}}=K$，由式(3)可以得到卷积输出$y$

$$y=\sum_{i=1}^{K}\sum_{j=1}^{K}x_{ij}w_{ij} \tag{9}$$

由式(9)可知，运算含有K^2个乘法运算和K^2个数的加法运算。对于K^2个乘法运算使用K^2个乘法器全并行计算；对于K^2个数的加法运算，在硬件加速设计时，一般采用经典的加法树来实现[8]，这种加法树首先将输入个数通过填补0的方式从K^2个扩充至$2^{\lceil\log_2(K^2)\rceil}$个，然后每两个数相加的和作为第2层的输入，通过这种方式，逐级累加直到最后一层得到总和。如图5所示为$K=3$时这种设计的结构。

这种加法树结构虽然极大提升了加法的并行性，但同时存在以下两个缺点：

(1) 消耗过多硬件资源

对于η个数的加法，这种加法树需要的加法器个数$f_1(\eta)$，所需的寄存器个数$g_1(\eta)$，所需的时钟周期$h_1(\eta)$分别为

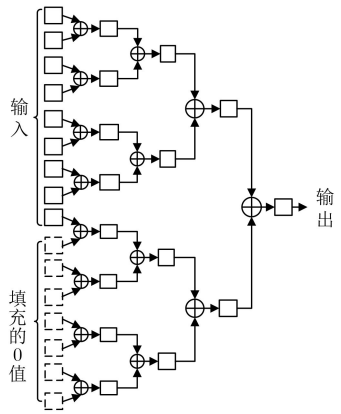

图 5 经典加法树

$$f_1(\eta)=2^{\lceil\log_2\eta\rceil}-1,\quad g_1(\eta)=2^{\lceil\log_2\eta\rceil+1}-1,$$
$$h_1(\eta)=\lceil\log_2\eta\rceil \tag{10}$$

比如对于144个数($K=12$)和256个数($K=14$)的加法运算，这种加法树需要加法器的个数都为255，寄存器个数都为511，时钟周期都为8。可见以上两种情况下，虽然它们加法输入的个数不一样，但它们消耗的加法器个数、寄存器个数和时钟周期一样。显然这种方法对于加法输入个数比2的幂次方稍微大时，是特别浪费计算资源与内存资源的。

(2) 带宽需求大

当$K=12$时，经典加法树需要同时计算的数据个数从144提升至256，带宽需求几乎扩大了1倍。

针对上述问题，本文设计的加法树的改进如下：

(1) 如果当前层的输入个数是偶数，则可以与经典加法树一样，每两个数相加；

(2) 如果当前层的输入个数是奇数，那么先对偶数个数使用(1)的方法进行并行计算，最后剩下的一个加数直接输出到下一层。

这种加法树不需要添补0即额外的存储器，也不需要额外的加法器，且需要的时钟周期与经典方法一样。对于9个数($K=3$)的加法运算，本文设计的加法树如图6所示。

从图6中可知，对于9个数的加法运算，本文设计的加法树需要加法器的个数为8，寄存器个数为20，计算的时钟周期为4。相应经典加法树需要加法器个数为15个，寄存器个数为31个，计算的时钟周期为4。

设加法树输入个数为η，那么下一层输入个数为$\lceil\eta/2\rceil$，依次类推，并用数学归纳法可证：

本文设计的加法树所需的加法器个数$f_2(\eta)$，所需的时钟周期$h_2(\eta)$，所需的寄存器个数$g_2(\eta)$分别为

$$f_2(\eta)=\eta-1,\quad h_2(\eta)=\lceil\log_2\eta\rceil,$$
$$g_2(\eta)=\sum_{i=0}^{h(\eta)}\left\lceil\frac{\eta}{2^i}\right\rceil \tag{11}$$

将全并行乘法模块与本文设计的加法树模块结合后的乘法-加法树模块结构如图7所示。

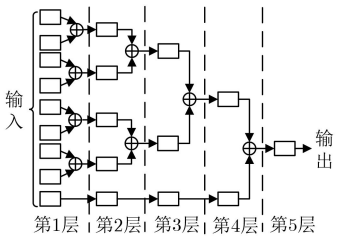

图 6 本文设计的加法树

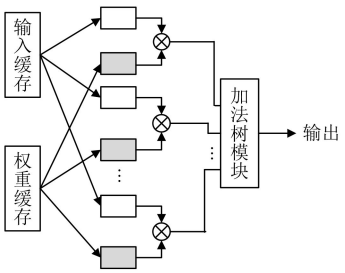

图 7 乘法-加法树模块

图7中乘法-加法树模块运算步骤为：

(1) 将输入特征图矩阵与卷积核权重矩阵存储到缓存区中，分别称为输入缓存与权重缓存；

(2) 从输入缓存与权重缓存中读取出数据，使用K^2个乘法器并行计算式(9)中的K^2个乘法，得到K^2个中间结果；

(3) 最后利用本文设计加法树的方法构建一个加法输入个数为K^2的加法树，加法树输入为(2)中得到的K^2个中间结果，其输出为乘法-加法模块的最终结果。

3.2.2 高效窗口缓存模块

根据式(3)可知，卷积层在计算时，会生成许多卷积窗口。由式(1)和式(2)可知，生成的卷积窗口数为$G=H_{\rm o}W_{\rm o}$。设卷积核大小$W_{\rm k}=H_{\rm k}=K$，输入特征图的一个通道矩阵$\boldsymbol{x}$形状为$[H,W]$，令

$$\boldsymbol{x}=\begin{bmatrix}x_{11}&x_{12}&\cdots&x_{1W}\\x_{21}&x_{22}&\cdots&x_{2W}\\\vdots&\vdots&\ddots&\vdots\\x_{H1}&x_{H2}&\cdots&x_{HW}\end{bmatrix} \tag{12}$$

为了便于说明，将矩阵$\boldsymbol{x}$的下标变为连续，即第i行第j列元素的下标为$j+(i-1)W$，那么矩阵$\boldsymbol{x}$重新表示为

$$\boldsymbol{x}=\begin{bmatrix}x_1&x_2&\cdots&x_W\\x_{W+1}&x_{W+2}&\cdots&x_{2\times W}\\\vdots&\vdots&\ddots&\vdots\\x_{(H-1)W+1}&x_{(H-1)W+2}&\cdots&x_{H\times W}\end{bmatrix} \tag{13}$$

定义$\boldsymbol{x}_{(i)}$表示第i个卷积窗口，其中$i\in[1,G]$。如图8所示，$\boldsymbol{x}_{(1)}$的第2列到第K列和$\boldsymbol{x}_{(2)}$第1列到第$K-1$列数据完全相同。第1个卷积窗口与第2个卷积窗口所有数据有$2\times K\times K$个，而共用的数据有$2\times K\times(K-1)$个，所以数据重复占比为$(K-1)/K$，并且当K值越大时，两个窗口之间数据重复占比也越大。例如当$K=3$，数据重复占比达到66.7%。

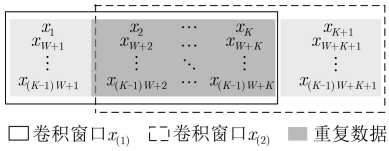

图 8 卷积窗口数据重用

若对所有卷积窗口都采用硬件并行必然会消耗过多的计算资源。所以本文考虑对卷积窗口进行流水线操作，这样可以大量减少存储资源和计算资源。这些卷积窗口使用同一个电路结构，在不同时间输入不同的卷积窗口数据得出卷积运算结果。为了对卷积窗口进行流水线操作，本文设计了一个窗口缓存模块，该模块由两个2维寄存器组成，其中一个是窗口缓存寄存器WINDOW_BUFFER，另一个是移位寄存器SHITF_BUFFER，如图9所示为窗口缓存结构示意图。

从图9中可知，WINDOW_BUFFER存储大小为$K\times K$, SHIFT_BUFFER存储大小为$(K-1)\times(W-K)$，每个时钟周期并行执行如下5个步骤：

(1) 从输入缓存INPUT_BUFFER中输入一个数据到WINDOW_BUFFER的第K行第1列；

(2) WINDOW_BUFFER每一行进行右移操作；

(3) WINDOW_BUFFER第2行至第K行的第K列的数据赋值给SHIFT_BUFFER的第1列所有行；

(4) SHIFT_BUFFER每一行进行右移操作；

(5) SHIFT_BUFFER最后一列的数据赋值给WINDOW_BUFFER第1行至第$K-1$行的第1列。

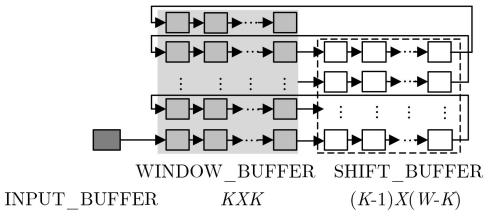

图 9 窗口缓存结构

如图10所示为上述步骤的时序图，在前$(K-1)\times W+K-1$时钟周期WINDOW_BUFFER寄存器中的数据无效，称为无效区，定义无效时钟周期$T_{\mathrm{u}}=(K-1)\times W+K-1$；在第$T_{\mathrm{u}}+1$个时钟周期时，WINDOW_BUFFER中存储的数据为$\boldsymbol{x}_{(1)}$；再经过1个时钟周期后，WINDOW_BUFFER中存储的数据为$\boldsymbol{x}_{(2)}$；依次类推，当在第$K\times W$个时钟周期时，WINDOW_BUFFER中存储的数据为$\boldsymbol{x}_{(W_{\mathrm{o}})}$；

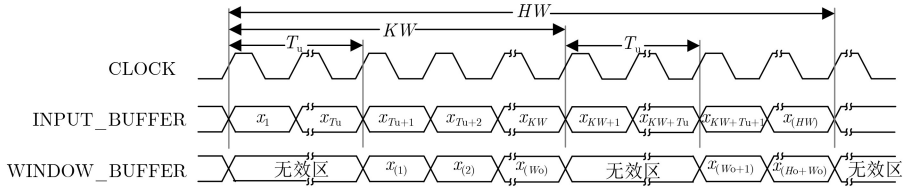

图 10 窗口缓存时序

最后，在第$H\times W$个时钟周期时，WINDOW_BUFFER中存储的数据为最后一个卷积窗口即$\boldsymbol{x}_{(H_{\mathrm{o}}W_{\mathrm{o}})}$。

从时序图10以及前面的分析中可知，本文设计的窗口缓存模块，可以在每个时钟周期生成一个卷积窗口进行后续运算，也即最高效的窗口流水方式。

3.3 输入与输出通道并行卷积层加速设计

在3.1节已经分析了卷积层并行加速的可行性，并提出3个可以并行的部分，本文结合3个并行部分，设计了一种卷积层并行加速方案。为了论述方便，下文中使用$\boldsymbol{X}_i$表示输入特征图矩阵$\boldsymbol{X}$的第i个通道的某个卷积窗口，$\boldsymbol{W}_{mni}$表示第m个卷积核的第n个通道的权重矩阵，则式(5)表示为

$$a_{mn}=\boldsymbol{X}_n\odot\boldsymbol{W}_{mn} \tag{14}$$

其中，$\odot$表示全并行乘法-加法树模块的运算。

如图11所示为对M个输出通道并行求出a_{1n}, $a_{2n},\cdots,a_{Mn}$的结构图，并将该模块封装为输出通道并行模块。

其中全并行乘法-加法树模块实现了卷积内并行模块计算。为了尽可能提升加速器性能，本文设计的并行加速方案同时对N个输入通道，M个输出通道进行并行计算，并使用之前设计的高效窗口缓存模块对G个卷积窗口进行流水线操作，具体结构如图12所示。

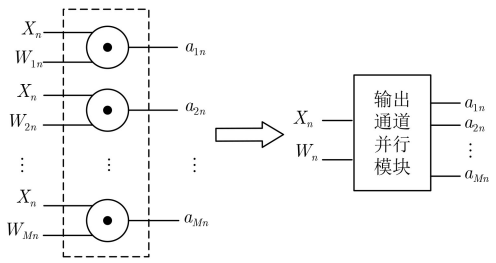

图 11 输出通道并行模块

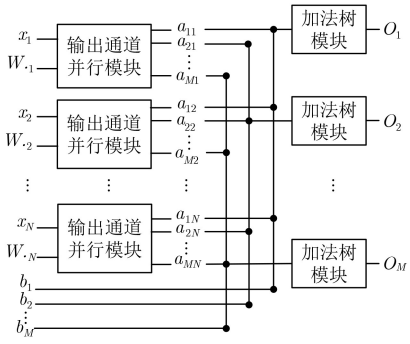

图 12 并行加速方案结构

其中$\boldsymbol{W}_{\cdot n}=\{\boldsymbol{W}_{mn}|m\in[1,M]\}$，表示所有$M$个卷积核的第$n$个通道矩阵集合。

图13为对卷积窗口进行流水线操作的示意图。

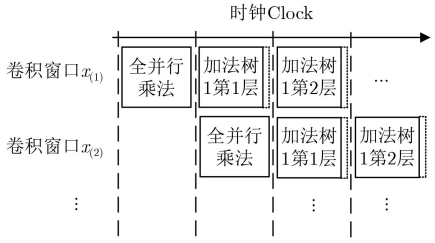

图 13 卷积窗口流水线

该方案使用N个输出通道并行模块，将图11中的输出通道并行模块中的乘法-加法树模块的加法树称为加法树1。在获得$\{a_{ij}|i\in[1,M],j\in[1,N]\}$后，由式(9)可知，为了得到第$m$个输出通道$O_m$，需对$N+1$个数做加法运算，即图12中的第$m$个加法树模块，此类加法树称为加法树2。两者都采用前面所设计的加法树结构。

4 实验验证

4.1 实验环境

本实验在GPU平台训练了多层的卷积神经网络对MNIST数据集[9]进行分类，然后在FPGA平台上实现加速前向推理。使用的FPGA平台是ALTERA公司开发的Cyclone V(5CGXFC9D6F27C7)的开发板，含有最大带宽为2 GB/s，容量为1 GB的DDR3 SDRAM。并通过PLL的倍频，将FPGA工作频率提升至100 MHz。采用的神经网络结构参数如表1所示。

此外还在CPU和GPU上进行了对比测试。其中CPU和GPU软件代码都是采用TensorFlow[10]深度学习框架，底层运算代码为C++。CPU采用的是Intel(R) Core™ i7-7700，主频为3.6 GHz；GPU采用的是NVIDIA GTX1080Ti，峰值双精度浮点运算性能为5.75TFLOPS，峰值单精度浮点运算性能为11.5 TFLOPS，显存容量为11 GB，显存带宽为484 GB/s, CUDA核心数量为3584，核心频率1.582 GHz。

表 1 卷积神经网络结构参数

层名称	层结构	参数量（个）
卷积层1	卷积核大小3×3，卷积核个数15，步长1	150
激活层1	无	0
池化层1	池化大小2×2，步长2	0
卷积层2	卷积核大小6×6，卷积核个数20，步长1	10820
激活层2	无	0
池化层2	池化大小2×2，步长2	0
全连接层	输出神经元个数10	3210

4.2 实验结果

FPGA电路板上资源消耗情况如表2所示。表2中DSP资源消耗比较多，主要是因为卷积运算时使用的乘法器比较多。

表 2 FPGA资源消耗情况

	资源	比例(%)
ALMs	89423/113560	79
Block Memory	730151/12492800	6
DSPs	342/342	100

如图14所示，当只使用一张图片进行前向推理，即批量大小为1时，FPGA比CPU快40.4倍左右，比GPU快33.52倍左右。随着批量大小的增大，FPGA依然比CPU快7.8倍以上。但当批量大小增大到64以上时，GPU的优势就突显出来，因为GPU运算的特点在于大带宽，可以满足多张图片同时并行计算。但是在嵌入式平台视频实时性检测的任务中，都是一帧一帧图片的处理即批量大小为1时，FPGA的优势非常明显，且功耗比GPU低得多，更适合在实际应用中采用。

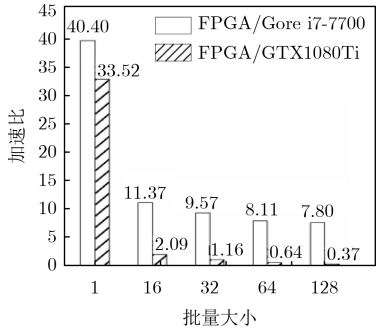

图 14 FPGA, CPU, GPU的性能对比

表3是与其他卷积神经网络FPGA实现的结果对比，由于各文献使用不同的FPGA器件和不同的网络结构，如果仅仅只采用性能作为参考指标会缺少公平，所以还增加能效比(power efficiency)参数以便全面的对比加速效果。

从表3中可以看出，本文设计的加速器能效比达到32.73 GOPS/W，优于其它3种方案，比现有文献[11]最好的实现方法提高了34%。此外其加速性能达到了317.86 GOPS，相比于文献[7]文献[11]加速效果明显，但性能相比于文献[12]还存在一些差距，其原因在于本文所采用FPGA平台的DSP资源比其它平台少很多，而DSP数量是提高加速器性能的关键因素之一。虽然受资源限制，测试平台没

表 3　与文献FPGA硬件加速对比

	文献[7]	文献[11]	文献[12]	本文方法
FPGA	Zynq XC7Z045	ZynqXC7Z045	Virtex-7 VX690T	Cyclone V 5CGXF
频率(MHz)	150	100	150	100
DSP资源	780(86.7%)	824(91.6%)	1376(38%)	342(100%)
量化策略	16 bit fixed	16 bit fixed	16 bit fixed	16 bit fixed
功耗(W)	9.630	9.400	25.000	9.711
性能(GOPS)	136.97	229.50	570.00	317.86
能效比(GOPS/W)	14.22	24.42	22.80	32.73

有充分发挥出本文设计的加速器最优性能。但现有结果已经表明本文实现的加速器具有很高的能效比及良好的加速性能。

5　结论

本文深入分析了卷积层并行加速的原理及可行性，重新设计了加法树模块，与乘法运算结合设计了通用的全并行乘法-加法树模块，之后又设计了高效的窗口缓存模块对卷积窗口进行流水线操作，并设计了输入通道并行、输出通道并行的加速方案对卷积运算加速，最后构建了一个完整的前向推理卷积神经网络FPGA加速器。通过实验分析，本文设计的加速器能效比达到32.73 GOPS/W，比现有的解决方案高了34%。表明本文实现的卷积神经网络硬件加速器具有很高的能效比，非常适合应用于嵌入式平台。

参 考 文 献

[1] LIU Weibo, WANG Zidong, LIU Xiaohui, *et al*. A survey of deep neural network architectures and their applications[J]. *Neurocomputing*, 2017, 234: 11–26. doi: 10.1016/j.neucom.2016.12.038.

[2] HAN Song, MAO Huizi, and DALLY W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding[J]. *arXiv preprint arXiv:* 1510.00149, 2015.

[3] COATES A, HUVAL B, WANG Tao, *et al*. Deep learning with COTS HPC systems[C]. Proceedings of the 30th International Conference on International Conference on Machine Learning, Atlanta, USA, 2013: III-1337–III-1345.

[4] JOUPPI N P, YOUNG C, PATIL N, *et al*. In-datacenter performance analysis of a tensor processing unit[C]. Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, Canada, 2017: 1–12. doi: 10.1145/3079856.3080246.

[5] MOTAMEDI M, GYSEL P, AKELLA V, *et al*. Design space exploration of FPGA-based deep convolutional neural networks[C]. Proceedings of the 21st Asia and South Pacific Design Automation Conference, Macau, China, 2016: 575–580. doi: 10.1109/ASPDAC.2016.7428073.

[6] ZHANG Jialiang and LI Jing. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network[C]. Proceedings of 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, USA, 2017: 25–34. doi: 10.1145/3020078.3021698.

[7] QIU Jiantao, WANG Jie, YAO Song, *et al*. Going deeper with embedded FPGA platform for convolutional neural network[C]. Proceedings of 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, USA, 2016: 26–35. doi: 10.1145/2847263.2847265.

[8] 余奇. 基于FPGA的深度学习加速器设计与实现[D]. [硕士论文], 中国科学技术大学, 2016: 30–38.
YU Qi. Deep learning accelerator design and implementation based on FPGA[D]. [Master dissertation], University of Science and Technology of China, 2016: 30–38.

[9] 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. doi: 10.1109/5.726791.

[10] ABADI M, BARHAM P, CHEN Jianmin, *et al*. Tensorflow: A system for large-scale machine learning[C]. Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, Savannah, USA, 2016: 265–283.

[11] XIAO Qingcheng, LIANG Yun, LU Liqiang, *et al*. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs[C]. Proceedings of the 54th Annual Design Automation Conference, Austin, USA, 2017: 62. doi: 10.1145/3061639.3062244.

[12] SHEN Junzhong, HUANG You, WANG Zelong, *et al*. Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA[C]. Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, USA, 2018: 97–106. doi: 10.1145/3174243.3174257.

秦华标：男，1967年生，教授，研究方向为智能信息处理、无线通信网络、嵌入式系统、FPGA设计.

曹钦平：男，1995年生，硕士生，研究方向为集成电路设计.