DOI:10.7544/issn1000-1239.2018.20170715
55(3):551-562,2018

面向边缘计算的嵌入式FPGA卷积神经网络构建方法

卢 冶[1] 陈 瑶[2,4] 李 涛[1,3] 蔡瑞初[2] 宫晓利[1,3]
[1](南开大学计算机与控制工程学院 天津 300350)
[2](广东工业大学计算机学院 广州 510006)
[3](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
[4](新加坡高等数字科学研究中心 新加坡 138632)
(luye@nankai.edu.cn)

Convolutional Neural Network Construction Method for Embedded FPGAs Oriented Edge Computing

Lu Ye[1], Chen Yao[2,4], Li Tao[1,3], Cai Ruichu[2], and Gong Xiaoli[1,3]
[1](*College of Computer and Control Engineering, Nankai University, Tianjin* 300350)
[2](*School of Computers, Guangdong University of Technology, Guangzhou* 510006)
[3](*State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences), Beijing* 100190)
[4](*Advanced Digital Sciences Center, Singapore* 138632)

Abstract At present, applications and services with high computational consumption migrate gradually from centralized cloud computing center to embedded environment in the network edge. FPGA is widely used in the embedded systems under edge computing because of its flexibility and high efficiency. The conventional FPGA based convolutional neural network construction method has shortcomings, such as long design cycle and small optimization space, which leads to an ineffective exploration of the design space of targeted hardware accelerator, especially in network edge embedded environment. In order to overcome these issues, a high level synthesis based general method for convolutional neural network construction on embedded FPGA oriented edge computing is proposed. The highly reusable accelerator function is designed to construct the optimized convolutional neural network with a lower hardware resource consumption. Scalable design methodology, memory optimization and data flow enhancement are implemented on the accelerator core with HLS design strategy. The convolutional neural network is built on embedded FPGA platforms. The results show the advantage of performance and power when compared with Xeon E5-1620 CPU and GTX K80 GPU, and suitable for edge computing environment.

收稿日期:2017-09-25;**修回日期**:2017-12-07
基金项目:国家自然科学基金项目(61702286);天津市自然科学基金项目(14JCQNJC00700,16ICYIC15200);计算机体系结构国家重点实验室开放课题(CARCH201504,CARCH201604);天津市大数据与云计算重大专项(15ZXDSGX00020);福建省信息处理与智能控制重点实验室开放课题(MJUKF201733);天津市优秀企业科技特派员项目(17JCTPJC49500)
This work was supported by the National Natural Science Foundation of China (61702286), the Natural Science Foundation of Tianjin (14JCQNJC00700, 16ICYIC15200), the Open Project Fund of the State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences) (CARCH201504, CARCH201604), the Major Science and Technology Program of Big Data and Cloud Computing of Tianjin (15ZXDSGX00020), the Open Project Fund of Fujian Provincial Key Laboratory of Information Processing and Intelligent Control (MJUKF201733); and the Special Fund for Excellent Enterprise Technology Correspondent in Tianjin (17JCTPJC49500).
通信作者:陈瑶(yaochen_nk@hotmail.com)

Key words edge computing; convolutional neural network; FPGA; high level synthesis; accelerator core

摘 要 当前,高计算消耗的应用和服务逐渐从集中式云计算中心向网络边缘的嵌入式环境迁移,FPGA 因其灵活性和高能效特性,使其在边缘计算的嵌入式系统中得到广泛的应用.传统的 FPGA 卷积神经网络构造方法存在设计周期长和优化空间小等缺点,无法有效探索硬件加速器的设计空间,在网络边缘的的嵌入式环境下尤为明显.针对该问题,提出一种面向边缘计算的嵌入式 FPGA 平台卷积神经网络通用的构建方法.通过设计卷积神经网络函数中的网络层间可复用的加速器核心,以少量硬件资源实现性能优化的卷积神经网络硬件;通过拓展设计、缓存优化及数据流优化等技术,实现 HLS 设计优化;利用该方法在嵌入式 FPGA 平台上构建相应卷积神经网络,实验结果表明:优化后的网络模型在与 Xeon E5-1620 CPU 和 GTX Titan GPU 相比时,在功耗与性能方面具有一定优势,适合应用于边缘计算环境中.

关键词 边缘计算;卷积神经网路;FPGA;高层次综合;加速器核心

中图法分类号 TP391

随着物联网的快速发展,传统的以云计算模型为核心的集中式处理方式,其关键技术已经无法高效处理边缘设备所产生的数据[1].边缘计算的提出正是为了克服云计算固有的问题[2-3],通过将应用、数据和服务从集中式节点推向网络边缘,在靠近移动设备和数据源头,融合网络、计算、存储和应用核心能力的开放平台,就近提供智能服务,进而形成对云计算的有益补充,推动物联网的发展[2].

Gartner 预测,到 2020 年全世界将有近 260 亿的物联网设备[4],越来越多的应用服务将被部署在网络边缘.边缘计算具有低延迟、高可用、高实时[4-7]等优势,因此,高计算消耗的应用也开始向边缘计算的嵌入式环境迁移.卷积神经网络(convolutional neural network, CNN)正是一种高计算消耗的典型应用,其在图像分类、语音识别、目标检测等诸多领域优势突出[8],也被越来越多的应用于边缘计算环境中.

SRAM 工艺的进步使得 FPGA 的计算密度逐渐提高,并出现包含通用计算核心的片上系统可编程逻辑阵列器件 SoC FPGA(system-on-chip FPGA)[9-10]平台.因具备专用硬件定制的特性,且兼顾低功耗需求,逐渐成为边缘环境中计算需求较高的嵌入式应用的理想平台[11].

传统面向 FPGA 的卷积神经网络构造方式是基于寄存器传输级(register transfer level, RTL)描述语言设计的,具有流程复杂、周期较长和优化空间较小等问题[11-14].尤其针对嵌入式 FPGA,缺乏卷积神经网络实现方法的有效的特征分析,而且卷积计算对硬件资源要求较高.因此,在边缘计算环境下,构建卷积神经网络的方法设计变得尤为重要[9-10,15-16].

随着 FPGA 上应用设计复杂度的提升,基于 RTL 描述语言的设计方法,逐步成为复杂硬件逻辑设计的瓶颈.高层次综合(high level synthesis, HLS)是解决该问题的一种有效技术,它采用高级编程语言对算法进行设计,通过编译、语义转化、映射与布局布线等过程,将其转化为可用于 ASIC 设计或 FPGA 设计的 RTL 语言的跨层次设计方法[10,14].运用 HLS 设计的电路,在逻辑资源充足的情况下能够取得良好性能,但在边缘计算环境中,边缘设备种类复杂,对资源集约的情况,设计方法与理论还有待深入探索[10].

在嵌入式 FPGA 上构建 CNN 存在的挑战是:应用重构复杂、过度依赖通用处理器和板载存储器资源,不适用于边缘计算环境,具体表现为:1)基于 RTL 设计方法构建的 CNN,应用重构复杂度较高,即使有目的地针对嵌入式 SoC FPGA 进行设计,在应用于新的网络模型时也需要大量的设计重构,并且往往表现出对通用处理器依赖过重,系统更新需进行大量额外的处理器程序设计.2)基于 HLS 设计方法构建的 CNN,目标对象集中在基于 PCIe 总线的大规模 FPGA 加速器,实现方式需板载等大规模存储器支持,设计结果也表现为较高的逻辑资源消耗,在嵌入式 FPGA 上的高效性及重用性大大降低.因此,针对网络边缘环境的嵌入式 FPGA,如何高效构建 CNN 并有效发挥其性能成为新的挑战.

综上所述,本文提出一种面向边缘计算的高层次综合卷积神经网络构建方法.该方法首先对应用进行软硬件划分,将任务分解为在通用处理器和

FPGA 上执行的 2 部分.采用 C++作为编程语言,针对不同的应用,利用不同的加速器实现其函数功能;针对不同的函数功能,通过加速器参数对其进行调节;同时,加速器本身也可通过参数调整其资源占用及性能.在此基础上,通过可拓展设计结合缓存优化、数据流优化等手段,提高数据处理并行度,提升 CNN 网络性能.本方法采用模块化的方式针对嵌入式 FPGA 平台构建 CNN 应用,多个模块之间通过片内总线实现高速连接,以 MNIST 和 CiFar10 这 2 种在边缘环境中具有代表性的数据集,在嵌入式 FPGA 上构建图像识别 CNN 的过程为例,验证了方法的有效性.本文主要包括 3 个创新点:

1) 构建共享加速器,通过复用单元降低存储开销,并建立详细准确的 CNN 加速器模型,保证在 CNN 重构时,更好地复用基础功能函数和层功能函数.

2) 引入循环切割参数,对输入输出循环进行切割,实现加速器可拓展性设计.通过对循环重排,提高数据处理并行度,使其充分发挥 HLS 优势,从而进行硬件生成优化.

3) 对数据缓存的数组进行分割,增加接口数量,保障加速器数据吞吐率,实现缓存优化;利用数据流优化指令,使卷积计算与缓存数据获取并行执行,提高加速器内部并行度.

1　基本概念

本节主要介绍 CNN 的基本概念,描述利用 HLS 构建的目标网络对象.

1.1　网络模型定义

一个完整的卷积神经网络通常包含多个卷积层、池化层以及全连接层.其中,卷积层实现对输入数据的特征提取,池化层根据不同的池化(极值或均值)原则进行采样,全连接层负责输入与输出的线性映射.如图 1 所示,该模型是在手写识别领域广泛应用的 LeNet-5 卷积神经网络结构[17-20].

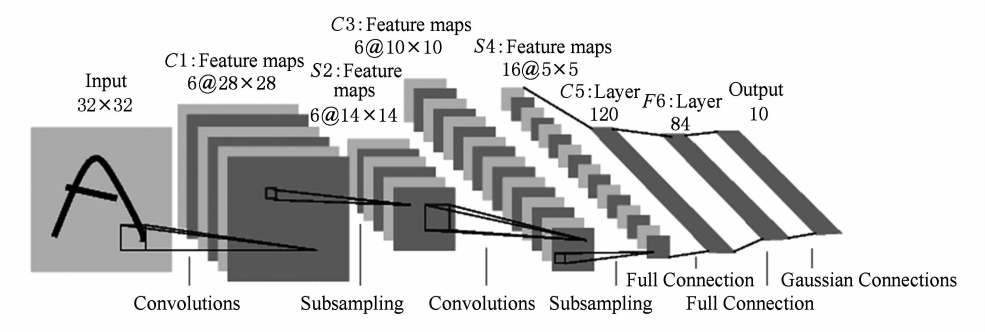

Fig. 1　LeNet-5 Hand written recognition model

图 1　LeNet-5 手写识别模型

1) 卷积层模型

卷积层通过输入 $\boldsymbol{f}_j^{\text{in}}$ 与由权重 $\boldsymbol{w}_{i,j}$ 组成的卷积核进行卷积操作,结果经偏置后获得输出,输出 $\boldsymbol{f}_i^{\text{out}}$ 即卷积的局部采样所得特征集合.卷积层的模型描述为

$$\boldsymbol{f}_i^{\text{out}} = \sum_{j=1}^{n_{\text{in}}} \boldsymbol{f}_j^{\text{in}} * \boldsymbol{w}_{i,j} + \boldsymbol{b}_i,\ 1 \leqslant i \leqslant n_{\text{out}}. \tag{1}$$

2) 池化层模型

池化层通常采用最大值采样或均值采样进行池化操作来降低输入矩阵的规模,其操作如式(2)所示.池化操作可有效降低下一层的数据处理量,同时避免特征信息丢失.

$$\begin{aligned} f_{i,j}^{\text{out}} = \max_{p\times p}(&f_{m,n'}^{\text{in}}, f_{m,n+1}^{\text{in}}, \cdots, f_{m,n+p-1}^{\text{in}}, f_{m+1,n}^{\text{in}}, \\ &f_{m+1,n+1}^{\text{in}}, \cdots, f_{m+1,n+p-1}^{\text{in}}, \cdots, f_{m+p-1,n}^{\text{in}}, \\ &f_{m+p-1,n+1}^{\text{in}}, \cdots, f_{m+p-1,n+p-1}^{\text{in}}) \end{aligned} \tag{2}$$

3) 全连接层

全连接层对输入进行线性空间转换,从而得到输出:

$$\boldsymbol{f}^{\text{out}} = \sum_{j=1}^{n} \boldsymbol{f}_j^{\text{in}} \boldsymbol{w} + \boldsymbol{b}. \tag{3}$$

4) 激活函数

激活函数实现对输入激励的非线性转换,通常在每一层后对输出结果进行处理.常用的激活函数包括冲击响应(Sigmod)、非线性(Relu)、三角函数(Tanh)等.

通过以上简介可知:1)CNN 内部的数据处理依照顺序执行,即前后相邻的 2 层之间,具有数据依赖关系.因此,边缘计算环境中,应有针对性地对该特征进行并行优化设计.2)池化层与全连接层相对于卷积层来讲,循环嵌套数目低、减少了卷积核心内部的循环操作,因此具备一定的子集关系,即针对卷积

层的优化方法具备重用性. 所以,在 FPGA 上进行 CNN 构建时,应充分考虑以上因素.

1.2 网络训练及应用

网络模型在使用前,需要先采用标定好的数据集进行训练,得到网络执行所需的权重及偏置数据. 网络的训练通常基于链式法则与反向传递原理,即通过损失函数,对一组输入所得输出与准确值进行对比评估,从而根据评估结果对网络中的权重、偏置等参数进行修正. 其中,得到网络输出的过程称为网络的前向传播,对比结果并进行网络参数修正的过程称为网络的反向传播. 对于需要预先训练的网络模型,反向传播只发生在网络训练过程中[17-18]. 在实际应用过程中,则只需要进行网络的前向传播.

边缘计算环境中,由于 FPGA 上的 CNN 侧重于面向具体应用来使用,而网络模型及权重数值可通过离线训练得出,因此研究 CNN 的构建方法时,重点关注网络的前向传播过程.

1.3 HLS 技术与设计要求

传统的 RTL 设计方法缺点明显,而 HLS 提供了将高级编程语言直接转化为 FPGA 或 ASIC 实现的硬件描述语言(Hardware Description Language, HDL)的途径. 同时,HLS 可通过在转化过程中插入编译指示指令的方法,对所生成的硬件结构进行优化,包括映射硬件寄存器、循环、接口等操作[11-14]. CNN 与应用联系紧密. 因此,在需要对其快速实现时,采用 HLS 可极大提升设计速度,并有效降低设计难度[13].

HLS 使高级编程语言直接映射到硬件结构,不同于针对特定处理器的程序设计,硬件平台无法提供比较完善的接口、存储、程序调用等支持,因此对基于 HLS 的设计过程提出了新的要求,主要包括:

1) 应用划分与重构

根据硬件平台特性,对应用程序进行任务划分与重构,使应用在硬件平台上呈现出高性能.

2) 硬件接口设置

采用 HLS 进行硬件设计时,接口性能对系统至关重要,其设计应充分考虑应用的具体特征.

3) 硬件电路优化

应用 HLS 插入编译指示指令等技术手段,剖析应用特征探索指令高效组合方式的优化实现.

2 基于 HLS 的 CNN 网络构建

本节将针对 CNN 网络模型,阐述基于 HLS 的加速器设计方法,通过可复用设计、可拓展设计和缓存优化和数据流优化,提高计算的并行度和资源利用效率,在此基础上进行 CNN 网络构建,并给出在嵌入式 FPGA 平台上的模块化网络构建具体方法.

2.1 加速器可复用设计

本文所研究卷积神经网络加速器面向网络边缘设备,即嵌入式、低功耗应用场景,此类应用场景下,应用的硬件平台普遍具有硬件资源集约的特点. 因此,在针对此类场景进行 CNN 加速器设计时,相同资源占用情况下的性能成为主要衡量标准. 根据第 1 节 CNN 理论模型可知,由于卷积计算的特性,CNN 中卷积层的计算占用其中主要的计算时间[16,21],而全连接层的计算相对卷积层的计算,减少了卷积核心内部的循环操作,因此,可同样采用卷积层加速器进行.

根据 CNN 数据处理特性,可知 CNN 内部的数据处理过程是依照不同处理层顺序进行的,即网络中前后相邻的层之间,由于具有数据依赖关系,其数据处理无法并行进行. 根据以上条件,为达到对资源的高效利用,采用各个不同网络层间共享加速器的方案,以实现对硬件资源的节约并提高单个加速器的计算效能,加速器共享如图 2 所示. 即其中卷积层及全连接层均采用同一个卷积加速器进行计算,而其中的所有池化层采用同一个池化层加速器进行处理.

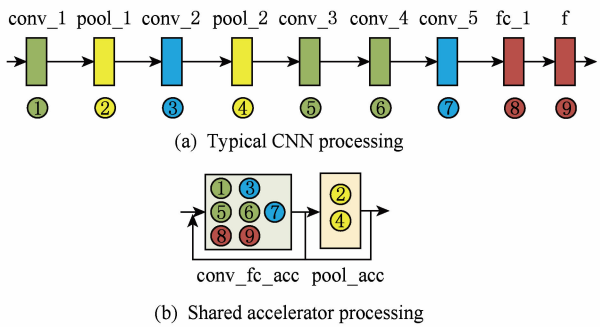

Fig. 2 Shared accelerator design

图 2 共享加速器设计

2.2 加速器可拓展性设计

常用的 CNN 中,每层网络函数所处理的输入输出数据均为 3 维数据. 采用各个网络层间共享加速器设计时,要求网络中每层的输入输出维度数据(x_in, y_in, z_in, x_out, y_out, z_out)需以函数变量形式传递给进行每一层处理的加速器. 与此同时,不同层进行数据处理时,其卷积或池化核心(*kernel*)与处理的步长(S)大小均不相同,因此,各层核心大

小 *kernel*、处理步长 S 也需做为变量进行传递.一个卷积层的处理过程描述如图 3 伪代码所示[22].

```
① I[x_in][y_in][z_in] / * input feature map * /
② O[x_out][y_out][z_out] / * output feature map * /
③ W[x_out][x_in][kernel][kernel] / * weights * /
④ for (m=0;m<x_out;m++)
⑤   for (n=0;n<x_in;n++)
⑥     for (r=0;r<y_out;r++)
⑦       for (c=0;c<z_out;c++)
⑧         for (i=0;i<kernel;i++)
⑨           for (j=0;j<kernel;j++)
⑩             O[m][r][c] +=
                W[m][n][i][j]×I[n][S×r+i][S×c+j].
```

Fig. 3　Poseudo code of a convolutional layer

图 3　卷积层伪代码

以上卷积层的 HLS 硬件实现中,由于每个循环的参数表均为变量,因此,其卷基层加速器的 HLS 实现结果如图 4 中(a)部分所示.卷积处理每次仅完成一个输出结果的计算.由于每一个循环的条件列表中均含有变量,无法采用 HLS 中所采用的循环流水线、循环展开等方式进行优化处理.为提升卷积加速器的性能,需提升其数据处理并行度——即提高其每次卷积计算所得结果数目.因此,为卷积层加速器增加 I_x,I_y,O_x,O_y 共 4 个循环切割参数,分别对输入输出维度的(x_in,x_out,y_out,z_out)的循环进行切割,并在卷积层加速器进行高层次综合前,确定其数值,即确定为固化的加速器参数,通过对卷积层循环的重排,使得其可以借助 HLS 工具所提供的循环展开及流水线优化,进行硬件生成优化.循环切割及重排后的的卷基层伪代码如图 5 所示.

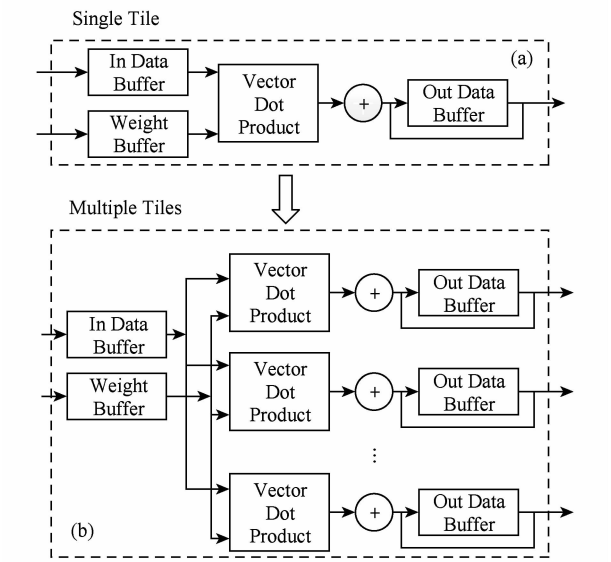

Fig. 4　Scalability design of the accelerator

图 4　加速器可拓展设计

```
① #pragma DATAFLOW
② fetch weight();
③ fetch bias();
④ fetch input data();
/ * layer loop * /
⑤ for(i=0;i<kernel;i++)
⑥   for (j=0;j<kernel;j++)
⑦     for (n=0;n<x_in;n+=Ix)
⑧       for (m=0;m<x_out;m+=Iy)
⑨         for (r=0;r<y_out;r+=Ox)
⑩           for (c=0;c<z_out;c+=Oy)
⑪             for (tr=r;tr<r+Ox;tr++)
⑫               for (tc=c;tc<c+Oy;tc++)
⑬ #pragma PIPELINE
⑭                 for(tx=n;tx<n+Ix;tx++)
⑮ #pragma UNROLL
⑯                   for(ty=m;ty<m+Ox;ty++)
⑰ #pragma UNROLL
⑱                     O[tx][tr][tc]+=W[ty][tx][i][j]×
                        I[ty][S×tr+i][S×tc+j];
⑲ write_output().
```

Fig. 5　Optimized convolutional layer

图 5　优化后的卷积层

其中,PIPELINE 及 UNROLL 表示对相应的循环层进行流水线及循环展开优化.经过图 5 所示优化后,卷积层的卷积核心根据循环展开的程度,展开为多条数据处理通道并行进行.如图 4 中(b)部分所示.

池化层的实现方式与卷积层的实现过程类似,区别在于池化层的操作不改变输入数据的第 1 维,因此,池化层计算的循环嵌套数目低于卷积层,但以上重排及高层次综合优化同样适用于池化层处理.

2.3　性能优化

1) 缓存优化

经过第 2.2 节可拓展性设计后,多数据并行处理提升了其数据处理性能.FPGA 中的数据缓存,采用片上 BRAM 存储器进行实现.因此,不同的数据输入与输出均被实例化为片上的 BRAM 区块,同一区块共享相同的输入输出接口.

由于输入输出接口数量的限制,加速器的数据吞吐速率受到缓存数据吞吐速率的影响.为保障加速器数据吞吐速率,同时进一步促进加速器内数据处理的并行性,对图 5 中进行数据缓存的数组进行分割,使其分布于多块不同的 BRAM 区块,增加其接口数量.本设计中依赖高层次综合工具,图 5 中代码的优化伪代码如图 6 所示.

经过以上缓存优化,输出数据与输入数据分别采用 I_x,I_y 个独立的 2 维数组进行数据缓存,而权重数据则采用 $I_x \times I_y$ 个独立的缓存数组.卷积层加

速器展开为 I_x 条并行计算通道,即卷积层结果计算的并行度达到 I_x. 由于卷积计算与池化计算均需大量的循环,且其数据获取及计算过程具有相似性,以上卷积加速器的优化方式同样适用于池化层加速器. 经过以上优化,卷积层及池化层核心数据处理部分可达到每个时钟周期输出 I_x 个结果.

```
① O[Ix][Ox][Oy]; /* Array partition dim=1, factor=lx */
② I[Iy][S×Ox+k][S×Oy+k]; /* dim=1, factor=ly */
③ W[Iy][Ix][k][k]. /* dim=1,2, factor=ly,lx */
```

Fig. 6 Accelerator buffer optimization

图 6 加速器缓存优化

2) 数据流优化

由于卷积层加速器的数据缓存均采用众多独立且不同的 BRAM,因此,其接口相互独立. 由于具有独立的接口,其数据获取可同时进行,从而缩短数据的获取与输出时间. 本设计中,采用高层次综合设计方法,为达到优化加速器核心数据流的目的,通过为输入数据、权重等设计独立的数据获取函数,并对所有函数进行数据流分析,同时采用数据流优化指令 DATAFLOW,使得数据的获取可以并行进行. 同时,因采用数据流优化指令,卷积计算单元的执行,可与缓存单元获取数据并行执行,从而提高了并行度.

2.4 系统软硬件划分

CNN 应用的数据处理过程由 3 个主要部分构成,即数据预处理、采用网络进行分类或识别、识别结果的整理与输出. 以典型的 CNN 手写识别为例,在 CNN 应用系统运行过程中,图像的预处理和结果输出部分与应用的前后端联系更为紧密,对计算能力的需求较低,所以该部分可划分给处理器中的软件来执行;而数据分类、识别操作计算量大,并且任务内的数据吞吐量相对较高,因此该部分可利用可重构硬件资源来构造专用的硬件加速器,借此可充分发挥 FPGA 的高计算效能优势. 目标系统的软硬件划分如图 7 所示. 为保障经过划分后的系统软件部分,可依据应用进行再编程,并保障硬件加速器的可重构性,CNN 的硬件加速器通过高性能片上总线与处理器进行连接,加速器的设计采用数据驱动形式,将输入与输出分别设置为硬件平台所支持的总线接口形式. CNN 计算过程中,所需权重数据及中间数据的规模较大,因此其数据存储采用片外动态存储器.

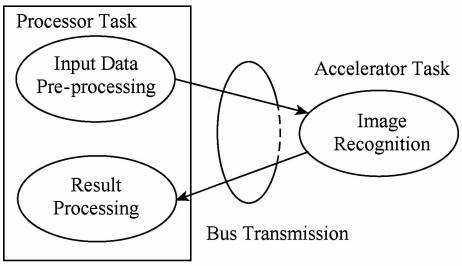

Fig. 7 System partitioning

图 7 系统分割

2.5 网络构建流程

本节采用网络层间可复用加速器设计(2.1 节),因此,网络各层之间数据的传输速率以及各层内部的处理是否形成有效的流水线层级将影响网络整体性能. 同时,各层能否高效共享存储空间,将决定网络硬件实现的资源占用率.

基于 HLS 及上述优化技术,在嵌入式 FPGA 上构建卷积神经网络需解决:1)应用中任务的软硬件划分;2)网络中不同层的任务分派;3)网络不同层间数据传输;4)生成硬件系统的验证与实现. 因此,设计实现流程如图 8 所示:

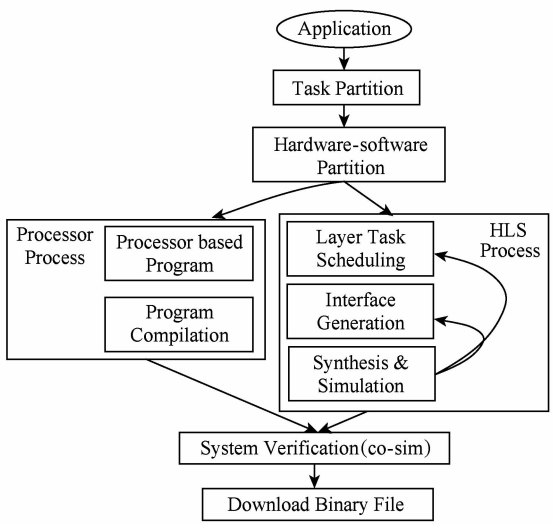

Fig. 8 System Generation Flow

图 8 系统实现流程

首先,对目标应用进行软硬件划分,将不同的应用任务分别划分到通用处理器与 FPGA 上;根据 2.1 节所述共享加速器设计,对 FPGA 上的任务进行构建,即对卷积神经网络进行构建,包括加速器的选择与初始化;对不同层在卷积层加速器及池化层加速器上进行分派;针对网络间的数据传输量,进行网络中各层间缓存的优化设计;其后进行整个网络的接口生成,将加速器组合进行封装. 然后对生成的

系统进行 C 语言到 RTL 语言的硬件综合及仿真.其中,处理器程序设计基于重构的加速器及加速器接口;完成处理器程序设计后,对整个系统进行联合仿真,验证输出结果的准确性;最后,对设计进行硬件综合及布局布线,得到可下载于 FPGA 平台上的比特流文件.针对具体的 CNN 网络如 LeNet-5,CiFarNet,网络构建和特征分析将在实验环节进行阐述.

3　实验及结果分析

本节在嵌入式 SoC FPGA 平台上,以 LeNet-5 和 CiFarNet 的网络构建具体方案为例,采用 MNIST 及 CiFar10 数据集对其进行性能测试,验证本文设计并评估设计中所采用的优化方法对设计结果的影响,并与桌面 CPU 和 GPU 性能及功耗进行对比分析.

3.1　实验方案

1) 软硬件环境

实验中采用 Xilinx Zedboard 嵌入式 FPGA 评估平台,该系统搭载一颗 xc7z020clg484SoC FPGA 芯片,片内由 ARM A9 处理器与可重构逻辑部分构成,如图 9 所示.可重构逻辑部分提供的 BRAM,DSP,FF 及 LUT 等资源数量分别为 280,220,106400 及 53200.对比实验中,采用 Intel XeonE5-1620 CPU,主频 3.50 GHz,GPU 为 nVidia GTX Geforce TiTan.实验中所采用高层次综合系统为 Xilinx Vivado_HLS 2017.1,硬件实现环境为 Xilinx Vivado 2017.1;卷积神经网络的训练与执行采用 CAFFE 1.0 版本[18].FPGA 功耗测量采用评估板电源外接功耗测量设备方式进行.

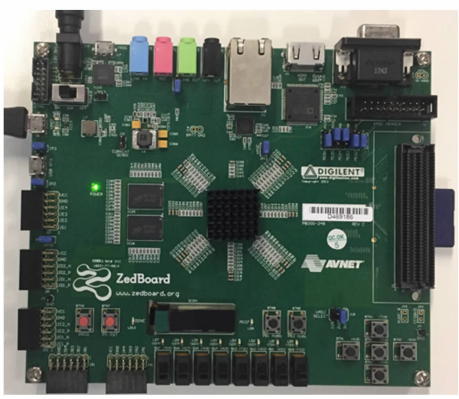

Fig 9　Experiment Board

图 9　嵌入式 SoC FPGA 实验开发板

2) 测试数据集与网络模型

MNIST 数据集包含 60 000 张手写数字图像构成的训练集及 10 000 张手写数字图像测试集.所有图片为 28×28 像素灰度图像.Lenet-5 网络模型常用来进行 MNIST 数据集内容的识别,其网络各层及参数如表 1 所示:

Table 1　Lenet-5 Network Model Parameters

表 1　Lenet-5 网络参数

Layer	Input	Kernel	Output	Activation
Conv-1	3×28×28	5×5	6×28×28	Relu
Pool-1	6×28×28	2×2	6×14×14	Max
Conv-2	6×14×14	5×5	16×10×10	Relu
Pool-2	16×10×10	2×2	16×5×5	Max
Fc-1	16×5×5	5×5	10	Relu

CiFar10 数据集包含 10 种不同物体的 60 000 张彩色图像,像素值为 32×32;其中 50 000 张用于模型训练,10 000 用于测试;Cifarnet 模型用于 CiFar10 数据集上的图像识别,其结构及各层参数如表 2 所示:

Table 2　CifarNet Network Model Paramenters

表 2　CifarNet 网络参数

Layer	Input	Kernel	Output	Activation
conv-1	3×32×32	5×5	32×32×32	Relu
pool-1	32×32×32	3×3	32×16×16	Max
Conv-2	32×16×16	5×5	32×16×16	Relu
Pool-2	32×16×16	3×3	32×8×8	Ave
Conv-3	32×8×8	5×5	64×8×8	Relu
Pool-3	64×8×8	3×3	64×4×4	Ave
Fc-1	64×4×4	4×4	10	Relu

3) 实验设计

对 4 项内容进行评估并分析其原因:

① 加速器评估.对基础功能加速器进行评估,揭示其参数设置对加速器性能、加速器硬件资源消耗的影响.由于 CNN 在进行 FPGA 实现时,DSP 为主要限制资源,因此本评估主要针对 DSP 的资源使用.

② 硬件资源消耗评估.在嵌入式 FPGA 实现中,对比优化前后的 CNN 硬件资源消耗,并对包含了已优化的加速器的系统性能、系统硬件资源消耗情况等进行评估分析.

③ 准确率评估.对所实现 Lenet-5,CiFarNet 的准确率与在桌面 PC 中采用 CAFFE 执行相同的网络结构与参数进行对比.

④ 性能功耗评估. 对以上网络在 FPGA 平台实现与在桌面 CPU,GPU 上采用 CAFFE 所构建的相同系统进行性能及功耗对比.

4) 性能评估标准

本实验中,所有时钟周期及处理时间的数据,为整个系统在评估板上实现后的实测数据. 在对硬件部分进行综合及布局布线过程中,时钟周期约束指定为 10 ns,在对设计进行优化时不对其进行改动. 嵌入式 FPGA 芯片中 ARM 处理器运行在 800 MHz 时钟频率,系统执行时间为完成整个识别任务所需的时间,对比系统运行于其默认工作时钟.

由于本设计针对嵌入式 FPGA,其性能评估针对单一输入图片的处理性能、硬件消耗与系统功耗,即单一图片的识别速度、此速度下硬件资源消耗量及功耗. 资源消耗评价指标表现为,针对可重构逻辑部分提供的 LUT,FF,DSP 及 BRAM 资源. 通过与桌面端 CPU 及 GPU 对比,考察系统的性能与功耗.

3.2 结果与分析

1) 加速器评估

① 卷积(全连接)层加速器. 卷积层加速器内部的乘加运算需采用 DSP 来进行. 如图 10 所示,由于设计中对 I_x,I_y 所代表的循环体进行了完全展开,因此卷积层加速器的 DSP 消耗量与循环切割参数 I_x,I_y 的大小相关,呈线性关系.

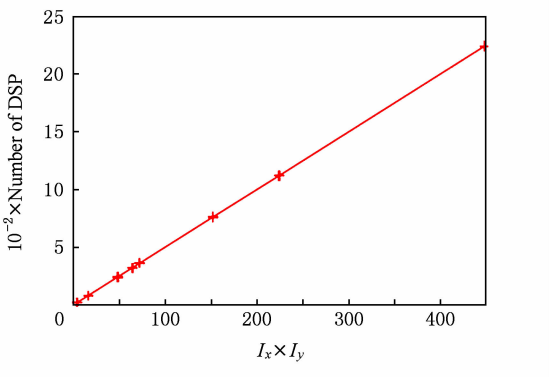

Fig. 10 DSP consumption with different parameters

图 10 不同参数设置下的 DSP 消耗

虽然个别参数点由于循环体的计算会消耗 DSP 资源,但误差在 5 个 DSP 范围内. 卷积加速器进行一次卷积计算所需的时钟周期数仅与 O_x,O_y 相关,经实验验证,其时钟周期结果符合 $O_x\times O_y\times K\times K$ 的设计. 卷积加速器采用片上 BRAM 进行数据暂存,加速器对 BRAM 的需求由 I_x,I_y,O_x,O_y 共同决定,如图 11 所示. 可知其参数设置与 BRAM 消耗呈线性关系,满足加速器的可拓展性设计要求.

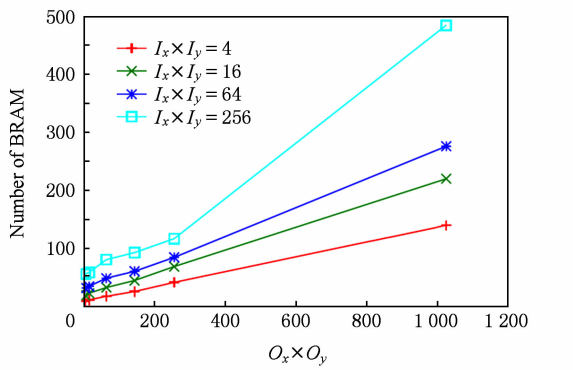

Fig. 11 BRAM consumption with different parameters

图 11 不同参数设置的 BRAM 消耗

② 池化层加速器. 由于池化层具有最大值池化和均值池化 2 种形式,不同的池化计算所需的硬件资源有所不同. 最大值池化不消耗 DSP 资源,仅需要 LUT,FF 及 BRAM 即可实现. 均值池化的资源需求类似于卷基层,但其 DSP 资源需求量远小于卷积层.

2) 硬件资源消耗评估

未优化的情况下,CNN 每一层的资源消耗量几乎相同. 网络整体占用硬件资源远远高于系统提供的硬件资源总量,硬件资源消耗的归一化如图 12 所示:

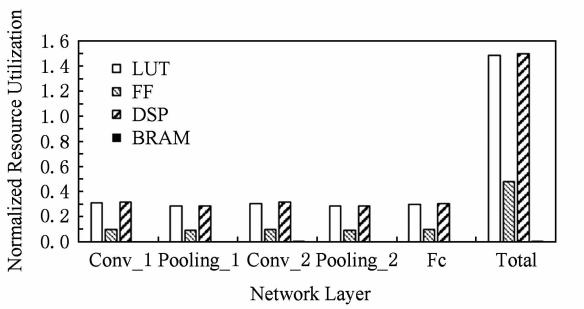

Fig. 12 Normalized resource utilization of layers with no optimization

图 12 未优化的各层硬件资源消耗归一化结果

由表 3 可见,网络整体处理时间大于各层处理时间之和,说明网络各层间存在较大的数据传输延时,且各层处理之间未能形成有效的流水线层级. 同时,网络接口也存在数据传输延时,从而导致整体性能受到影响. 通过对嵌入式 FPGA 结构及应用特征的深入分析,充分利用 HLS 技术进行网络结构精简,综合利用循环展开、流水线、数据缓存插入及数据流优化等方法,优化手写识别卷积神经网络的硬件实现. 检测网络硬件资源占用百分比、执行所需时钟周期数及执行时间,其结果如表 4 所示. 对比表 3 和表 4 可知,结构精简后的网络模型的硬件资源占

用量降低了 38.3%;利用循环展开与流水线技术可进一步提升网络性能,相比未优化也获得 23.1%的提升,但其资源占用有所升高;未进行数据缓存优化时,网络实现内部没有能够展开的数据缓存空间,循环展开和流水线优化所达到的性能受到限制;数据缓存插入可优化网络实现内部数据存储,再对插入的数据缓存进一步分割,提高了循环展开和流水线优化等级,网络性能得到进一步提升,同时 BRAM 的使用量随之升高;数据流优化使得网络的性能达到最优,同时资源消耗稍有提升,但未超过系统所提供资源上限.

Table 3　Network Layer and Network Performance Without Optimization

表 3　未优化网络各层及网络整体性能

Layer Name	# of Cycles	Clock Period/ns	Running Time/ms
conv_1	2 953 585	10	29.535 85
pool_1	99 145	10	0.991 45
conv_2	2 531 815	10	25.318 15
pool_2	33 985	10	0.339 85
fc	44 571	10	0.445 71
Network	5 669 978	10	56.699 78

Table 4　Resource Utilization and Performance

表 4　优化后资源占用及性能

Utilization	LUT	DSP	FF	BRAM	# of Cycles	Clock Period/ns	Running Time/ms
Baseline	149	149	48	10	5 669 978	10	70
Network Simplify	92	93	30	10	5 634 928	10	65
Unroll & Pipeline	107	93	32	10	4 361 114	10	50
Buffer Insert	102	94	33	11	2 726 064	10	35
Data Flow	84	94	38	11	232 504	10	30

LeNet-5 网络中,主要采用具有 Relu 激活函数的卷积层、全连接层以及采用 maxpooling 的池化层,因此,其网络构建中仅采用一个卷积加速器与一个池化加速器. CifarNet 中采用具有 Relu 激活函数的卷基层以及 Maxpooling 和 Avepooling 的池化层,因此其网络构建包含以上 3 种硬件加速器. 通过分析网络参数,各加速器参数设置及资源占用如表 5 所示,针对所选平台及网络模型,DSP 作为稀缺资源表现出较高的占用量,其他逻辑及存储资源均可满足需求.

Table 5　Network Parameter Settings and Resource Consumption

表 5　参数设置与资源消耗

Net	Acc	Params	BRAM	DSP	LUT
L-Net	Conv	8,4,28,28	193	174	23 582
	MPool	16,16,28,28	68.9%	79.1%	44.3%
C-Net	Conv	10.4.32.32	256	220	32 542
	Mpool	16,16,32,32	91.4%	100%	61.2%
	Apool	16,16,16,16			

Notes: Mpool stands for max pooling layer, Apool stands for ave pooling layer, parameters format(I_x,I_y,O_x,O_y).

3) 准确率评估

由于本设计中未涉及数据量化,因此所实现的 LeNet-5,Cifarnet 模型的准确率变化单纯由硬件实现中浮点运算的精度变化引起. 通过 MNIST,CIFAR-10 数据集实测结果可知,嵌入式 FPGA 上网络的实现未对离线训练网络的最优精度产生影响,如表 6 所示. 经过充分训练后的识别准确率分别可达 97%和 79%.

Table 6　Comparison of Network Implementation Accuracy

表 6　网络实测精度对比

Platform	Precision Accuracy/%		
	CPU	GPU	SoC FPGA
LeNet-5	97.18	97.18	98.7
Cifarnet	79.1	79.1	79

4) 性能功耗评估

选择表 5 中的性能优化配置,对其性能及功耗进行测量,并将性能和功耗与 Intel XeonE5-1620 CPU 和 GTX Titan GPU 比较,其对比结果如表 7 所示.

针对以上 2 个模型优化后的 FPGA 实现了在性能方面表现出的较高优势,得益于 FPGA 较高的并行执行能力. 由于网络模型的规模较小,与 GPU 相比并没有表现出其与 CPU 相比时所具有的优势. 不同平台的功耗对比表 8 所示. 其中,FPGA 平

台得益于其较高的计算效能,具有最低功耗.虽然对于较大规模的 Cifarnet 其计算速度相比 GPU 优势减低,但功耗较低,其效能仍远高于桌面 GPU,表现出较高的能耗优势.

Table 7 Latency per Image

表 7 单张图片处理时间

Platform	Latency/ms		
	CPU	GPU	FPGA
LeNet-5	19.983	6.7232	7.6
Cifarnet	31.248	7.1262	28.9

Table 8 Comparison of Power Consumption

表 8 网络功耗对比

Platform	Power Consumption/W		
	CPU	GPU	FPGA
LeNet-5	130	290	4.19
Cifarnet	130	290	4.21

4 相关工作

施巍松等人根据边缘式大数据处理模式提出边缘计算模型[1,23],指出可在网络边缘设备上,增加执行任务计算能力和数据分析处理能力,降低云计算中心负载,减缓网络带宽压力,提高数据处理效率.Liyanage 等人在雾计算模型基础上,提出一种面向服务的移动嵌入式平台服务框架[6],可为物联网中的边缘设备提供程序模型的部署和执行的平台.Liang 等人和 Rahman 等人分别通过融合边缘服务器与边缘设备的体系结构[24-25]有效支持请求实时响应、增强网络服务能力.以上这些工作侧重于计算模式,为边缘计算提供了架构级的指导.

边缘计算环境下,应用系统对计算速度、灵活性等需求不断提高,嵌入式加速器对特定应用具有较高的处理效能,因此也成为应用加速首选[16,21,26-29].例如夏辉等人通过设计专用的 ECC 指令执行硬件,提升了在资源受限的嵌入式环境中执行 ECC 加密算法的效能[27];Li 等人采用 HLS 方法实现了嵌入式 H264 解码器,实现了对输入 H264 数据流的实时硬件解码[16].

卷积神经网络是一种典型的应用,针对 CNN 的加速器设计与实现,诸多文献在不同层面进行了重点研究[21-22,26,29-31],例如 Qiu 等人采用 RTL 语言,针对图像处理应用中的卷积神经网络,进行了针对大规模加速器板卡和嵌入式 FPGA 的实现[21],其结果显示,基于 RTL 的设计硬件资源消耗易控制,但与本文基于 HLS 的设计相比,其系统更新或应用重构复杂度较高.Farabet 等人通过在 FPGA 平台实现了通用处理器结合单一的卷积加速器模块的结构[29],建立了卷积神经网络加速单元,利用通用处理器核心处理卷积计算以外的任务,其设计同样采用 RTL 语言,系统结构简单,对通用处理器的依赖程度较高,应用更新需进行大量的处理器程序设计.Zhang 等人、Suda 等人和 Venieris 等人分别采用 HLS 方法对基于 FPGA 的卷积神经网络进行了研究[22,30-31],分别采用了 C/C++,OpenCL,CUDA 等编程语言对卷积神经网络硬件进行实现,但其目标系统是基于 PCIe 总线的大规模 FPGA 加速器,以性能为最终优化目标,其方法不能直接应用于边缘计算模式,因此,在边缘设备上实现 CNN,需要结合边缘环境低功耗、高实时等特点进行特征分析.而本文所关注的正是面向边缘计算环境,如何满足 CNN 应用所面临的低带宽、高实时性需求,通过加速器设计提升系统性能.

5 总 结

面向边缘计算的嵌入式环境,针对 FPGA 平台提出一种基于高层次综合的卷积神经网络构建方法,通过设计高度复用的核心库函数,采用模块化方式构造网络.对 FPGA 结构及应用特征进行剖析,挖掘 HLS 技术优势,通过加速器可拓展设计、缓存优化及数据流优化等技术,实现高效的编译指示指令组合优化网络性能.本文在嵌入式 FPGA 平台上构建手写识别卷积神经网络 LeNet-5 及 CifarNet,与 CPU 和 GPU 相比,在嵌入式 FPGA 上的功耗及性能均表现出较大优势,体现出在边缘计算环境下构建卷积神经网络的高效性,实验结果验证了本文所提出的方法,在深度学习及嵌入式高性能计算领域具有较高的应用价值和推广价值.

参 考 文 献

[1] Shi Weisong, Sun Hui, Cao Jie, et al. Edge computing—An emerging computing model for the Internet of everything era [J]. Journal of Computer Research and Development, 2017, 54(5): 907-924 (in Chinese)

(施巍松,孙辉,曹杰,等. 边缘计算:万物互联时代新型计算模型[J]. 计算机研究与发展,2017,54(5):907-924)

[2] Yi Shanhe, Hao Zijiang, Zhang Qingyang, et al. LAVEA: Latency-aware video analytics on edge computing platform [C] //Proc of the 37th IEEE Int Conf on Distributed Computing Systems (ICDCS). Piscataway, NJ: IEEE, 2017: 2573-2574

[3] Shi Weisong, Dustdar S. The promise of edge computing [J]. Computer, 2016, 49(5): 78-81

[4] Aggarwal C, Srivastava K. Securing IOT devices using SDN and edge computing [C] //Proc of the 2nd Int Conf on Next Generation Computing Technologies. Piscataway, NJ: IEEE, 2017: 877-882,

[5] Sabella D, Vaillant A, Kuure P, et al. Mobile-edge computing architecture: The role of MEC in the Internet of things [J]. IEEE Consumer Electronics Magazine, 2016, 5(4): 84-91

[6] Liyanage M, Chang C, Srirama S N. mePaaS: Mobile-embedded platform as a servicefor distributing fog computing to edge nodes [C] //Proc of the 17th Int Conf on Parallel and Distributed Computing, Applications and Technologies. Piscataway, NJ: IEEE, 2016: 73-80

[7] Park H D, Min O G, Lee Y J. Scalable architecture for an automated surveillance system using edge computing [J]. Journal of Supercomputing, 2017, 73(3): 1-14

[8] 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)

[9] Baklouti M, Ammar M, Marquet P, et al. A model-driven based framework for rapid parallel SoC FPGA prototyping [C] //Proc of the 22nd IEEE Int Symp on Rapid System Prototyping. Piscataway, NJ: IEEE, 2011: 149-155

[10] Oruklu E, Hanley R, Aslan S, et al. System-on-chip design using high-level synthesis tools [J]. Circuits and Systems, 2012, 3(1): 1-9

[11] Meeus W, Beeck K V, Goedem'e T, et al. An overview of todays high-level synthesis tools [J]. Design Automation for Embedded Systems, 2012,16(3): 31-51

[12] Gajski D D, Ramachandran L. Introduction to high-level synthesis [J]. IEEE Design & Test of Computers, 1994, 11(4): 44-54

[13] Chen Yao, Gurumani S T, Liang Yun, et al FCUDA-NoC: A scalable and efficient network-on-chip implementation for the CUDA-to-FPGA flow [J]. IEEE Trans on Very Large Scale Integration Systems, 2016, 24(6): 2220-2233

[14] Chen Ying, Nguyen T, Chen Yao, et al. FCUDA-HB: Hierarchical and scalable bus architecture generation on FPGAs with the FCUDA flow [J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35(12): 2032-2045

[15] Fleming K, Lin C C, Dave N, et al. H.264 decoder: A case study in multiple design points [C] //Proc of the 6th ACM/IEEE Int Conf on Formal Methods and Models for Co-Design (MEMOCODE). New York: ACM, 2008: 165-174

[16] Li Huimin, Fan Xitian, Jiao Li, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks [C] //Proc of the 26th IEEE Int Conf on Field Programmable Logic and Applications (FPL). Piscataway, NJ: IEEE, 2016: 1-9

[17] Chetlur S, Woolley C, Vandermersch P, et al. cuDNN: Efficient primitives for deep learning [EB/OL]. [2017-09-01]. https://arxiv.org/abs/1410.0759v3, 2014-12-18/2017-09-01

[18] Jia Yangqing, Shelhamer E, Donahue J, et al. Caffe: Convolutional architecture for fast feature embedding [C] //Proc of the 22nd ACM Int Conf on Multimedia. New York: ACM, 2014: 675-678

[19] Jaderberg M, Vedaldi A, Zisserman A. Speeding up convolutional neural networks with low rank expansions[EB/OL]. [2017-09-01]. https://arxiv.org/abs/1405.3866v1, 2014-05-15/2017-09-01

[20] Ren Shaoqing, He Kaiming, Girshick R, et al. Faster R-CNN: Towards real-time object detection with region proposal networks [C] //Proc of the 28th Int Conf on Neural Information Processing Systems. Cambridge: MIT Press, 2015: 91-99

[21] Qiu Jiantao, Wang Jie, Yao Song, et al. Going deeper with embedded fpga platform for convolutional neural network [C] //Proc of the 2016 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2016: 26-35

[22] Zhang Chen, Fang Zhenman, Zhou Peipei, et al. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks [C] //Proc of Int Conf on Computer-Aided Design. New York: ACM, 2016: 12

[23] Shi Weisong, Cao Jie, Zhang Quan, et al. Edge computing: Visionand challenges [J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646

[24] Liang Tong, Li Yong, Gao Wei. A hierarchical edge cloud architecture for mobile computing [C] //Proc of the 35th Annual IEEE Int Conf on Computer Communications. Piscataway, NJ: IEEE, 2016: 1-9

[25] Rahman A, Hassanain E, Hossain M S. Towards a secure mobile edge computing framework for Hajj [J]. IEEE Access, 2017, 5(99): 11768-11781

[26] Liu Xinheng, Chen Yao, Nguyen T, et al. High level synthesis of complex applications: An H.264 video decoder [C] //Proc of the 2016 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2016: 224-233

[27] Xia Hui, Yu Jia, Qin Yao, et al. The researches on the ASIP of ECC in embedded domain [J]. Chinese Journal of Computers, 2017, 40(5): 1092-1108 (in Chinese)

(夏辉, 于佳, 秦尧, 等. 嵌入式领域 ECC 专用指令处理器的研究[J]. 计算机学报, 2017, 40(5): 1092-1108)

[28] Ma Jiuyue, Yu Zihao, Bao Yungang, et al. A programmable data plane design in computer architecture [J]. Journal of Computer Research and Development, 2017, 54(1): 123-133 (in Chinese)
(马久跃, 余子濠, 包云岗, 等. 体系结构内可编程数据平面方法[J]. 计算机研究与发展, 2017, 54(1): 123-133)

[29] Farabet C, Poulet C, Han J Y, et al. Cnp: An fpga-based processor for convolutional networks [C] //Proc of the 19th Conf on Field Programmable Logic and Applications (FPL 2009). Piscataway, NJ: IEEE, 2009: 32-37

[30] Suda N, Chandra V, Dasika G, et al. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks [C] //Proc of the 2016 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2016: 16-25

[31] Venieris S I, Bouganis C S. FPGA convnet: A framework for mapping convolutional neural networks on FPGAs [C] // Proc of the 24th Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2016: 40-47

Lu Ye, born in 1986. PhD and assistant professor in Nankai University. Member of CCF. His main research interests include embedded system, Internet of things and machine learning.

Chen Yao, born in 1987. PhD. His main research interests include high level synthesis, deep learning optimization.

Li Tao, born in 1977. PhD. Associate professor at the College of Computer and Control Engineering, Nankai University, China. Member of the ACM and the IEEE Computer Society, and senior member of CCF. His main research interests include high-performance computing, machine learning and Internet of things (litao@nankai.edu.cn).

Cai Ruichu, born in 1984. PhD and professor in Guangdong University of Technology. His main research interests include data mining, machine learning and information retrieval (cairuichu@gmail.com).

Gong Xiaoli, born in 1983. PhD and associate professor in Nankai University. Member of CCF. His main research interests include embedded system design and optimization (gongxiaoli@nankai.edu.cn).