QingLong:一种基于常变量异步拷贝的神经网络编程模型

杜伟健[1),2),3),7)] 陈云霁[1),2),4),5),6)] 支 天[1),3),7)] 吴林阳[1),2),3),7)]
陈小兵[1),2),3),7)] 庄毅敏[1),2),3),7)]

[1)](中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)
[2)](中国科学院大学 北京 100049)
[3)](上海寒武纪信息科技有限公司 上海 201308)
[4)](张江实验室脑与智能科技研究院 上海 201308)
[5)](上海脑科学与类脑研究中心 上海 201308)
[6)](中国科学院脑科学与智能技术卓越创新中心 上海 201308)
[7)](中科寒武纪科技股份有限公司 北京 100190)

摘 要 近年来,人工神经网络的研究取得了巨大成就,在图像识别、自然语言处理等领域均有突破性的成果,同时产生了众多商业应用,方便了我们的生活,比如语音助手、辅助驾驶等.由于神经网络算法属于计算密集型和访存密集型的负载,传统CPU处理器已不能满足其大规模商业化应用的需求,因此学术界和产业界试图在GPU、FPGA和ASIC上寻求突破.其中,神经网络加速器作为一种ASIC,它提供了高性能、低功耗的硬件解决方案,相关研究也越来越多.神经网络加速器作为一种协处理器,在其计算前后需要将数据在主机与设备之间进行搬运.特别是对吞吐量要求较高的神经网络前向推理任务,需要将网络模型参数、硬件指令等常量数据和输入、输出等变量数据,分别从主机内存拷入设备内存.如果常量数据在每一份输入数据计算前都拷贝一次,就存在常量数据重复拷贝的问题,浪费了时间与存储资源.如何在神经网络开发工具软件中实现拷贝多次变量数据但只拷贝一次常量数据,如何保证指令在每次计算中都正确寻址常量和变量,如何简化用户编程,提供用户友好的接口,就成为一系列值得研究的问题.在本文中,我们提出了一种基于常变量异步拷贝的神经网络开发工具软件及其编程模型QingLong来解决上述问题.QingLong编程模型包含三个阶段:定义网络、编译网络和计算.在定义网络阶段,用户可以为神经网络的数据节点绑定常量数据;在编译网络阶段,通过REOFF数据包装法将常量数据封装为数据包;在计算网络阶段,用户拷贝一次数据包后即可多次拷入输入数据并计算输出结果.该编程模型具有编译、计算分离,常变量异步拷贝,计算和数据拷贝可切分为三级流水线等优势.实验表明,在连续计算100份输入样本时,QingLong比DLPlib有平均17.48倍的性能提升,且输入样本越多,性能提升的倍数越大.

关键词 神经网络;编程模型;常量和变量;异步拷贝;软件开发工具
中图法分类号 TP183 **DOI号** 10.11897/SP.J.1016.2020.00587

收稿日期:2019-09-09;在线出版日期:2020-02-08.本论文工作受到国家重点研发计划(2017YFA0700900,2017YFA0700902,2017YFA0700901,2017YFB1003101,2018AAA0103300)、国家自然科学基金(61432016,61532016,61672491,61602441,61602446,61732002,61702478,61732007,61732020)、北京市自然科学基金(JQ18013)"核心电子器件、高端通用芯片及基础软件产品"科技重大专项(2018ZX01031102)、中国科学院科技成果转移转化重点专项(KFJ-HGZX-013)、中国科学院前沿科学重点研究项目(QYZDB-SSW-JSC001)、中国科学院战略性先导科技专项(XDB32050200,XDC01020000)、中科院标准化研究项目(BZ201800001)、北京智源人工智能研究院以及北京市科技新星计划项目(Z191100001119093)的支持.**杜伟健**(通信作者),博士研究生,中国计算机学会(CCF)会员,主要研究领域为深度学习、神经网络加速器编程库.E-mail:duweijian@ict.ac.cn.**陈云霁**,中国科学院计算技术研究所研究员、博士生导师,国家杰出青年科学基金入选者,长江学者青年项目入选者,中国计算机学会(CCF)杰出会员,主要研究领域为计算机体系结构和计算智能.**支 天**,博士,中国计算机学会(CCF)会员,主要研究领域为可重构架构和集成电路设计.**吴林阳**,博士,中国计算机学会(CCF)会员,主要研究领域为深度学习、神经网络加速器编程库.**陈小兵**,博士研究生,中国计算机学会(CCF)会员,主要研究领域为深度学习、神经网络加速器编译器.**庄毅敏**,博士研究生,中国计算机学会(CCF)会员,主要研究领域为深度学习、神经网络加速器编译器.

QingLong: A Neural Network Programming Model Based on Asynchronous Copy of Constant and Variable

DU Wei-Jian[1),2),3),7)] CHEN Yun-Ji[1),2),4),5),6)] ZHI Tian[1),3),7)] WU Lin-Yang[1),2),3),7)]
CHEN Xiao-Bing[1),2),3),7)] ZHUANG Yi-Min[1),2),3),7)]

[1)] (*SKL of Computer Architecture, Institute of Computing Technology, CAS, Beijing* 100190)

[2)] (*University of Chinese Academy of Sciences, Beijing* 100049)

[3)] (*Cambricon Technologies, Shanghai* 201308)

[4)] (*Institute of Brain-Intelligence Technology, Zhangjiang Laboratory, Shanghai* 201308)

[5)] (*Shanghai Research Center for Brian Science and Brain-Inspired Intelligence, Shanghai* 201308)

[6)] (*CAS Center for Excellence in Brain Science and Intelligence Technology, Shanghai* 201308)

[7)] (*Cambricon Technologies, Beijing* 100190)

Abstract In recent years, the research of artificial neural network has made great achievements in image recognition, natural language processing and other fields. At the same time, it has produced many commercial applications, which is convenient for our life, such as voice assistant, assisted driving and so on. Because the neural network algorithm belongs to the computing intensive and memory intensive application, the traditional CPU processor is not suitable for large-scale commercial applications, so the academia and industry try to seek a breakthrough in GPU, FPGA and ASIC. Neural network accelerator is a kind of ASIC. It provides high-performance, low-power hardware solutions, which has many related research. As a kind of coprocessor, neural network accelerator needs to copy data between the host memory and device memory before and after its calculation. Especially for the neural network inference task with high throughput requirements, constant data such as network model parameters, hardware instructions and variable data such as input and output are copied into device memory from host memory. If constant data is copied once before each input data calculation, there is a problem of repeated copying of constant data, which wastes time and storage resources. There are a series of problems worth studying. How to copy multiple variable data but only one constant data in the neural network development tool software? How to ensure that the instructions address constants and variables correctly in each calculation? How to simplify user programming and provide user-friendly interface? In this paper, we propose neural network development tool based on asynchronous copy of constant and variable and its programming model QingLong to solve the above problems. QingLong programming model consists of three stages: network definition, compilation and computation. In network definition stage, users can bind constant data for data nodes of neural network. In network compilation stage, constant data is packaged into data package by REOFF method. In network computation stage, users can copy input data and calculate output results many times after one data package is copied. The programming model has the advantages of compiling and computing separation, asynchronous copy of constant and variable, calculation and data copy can be cut into three stage pipelines. The experiments show that QingLong has an average performance improvement of 17.48x over DLPlib when calculating 100 input samples continuously. And the more input samples, the greater the performance improvement.

Keywords neural network; programming model; constant and variable; asynchronous copy; software development kit

1 引　言

近年来,人工神经网络的研究取得了显著的成就,在计算机视觉[1]、计算机图形学[2]、自然语言处理[3]等领域均有突破性的成果,也有众多案例成功实现了商用,服务于我们每个人,例如苹果公司的Siri、百度的自动驾驶汽车、公安视频监控等.神经网络包括反向训练和前向推理两种计算任务.在实际应用中,训练任务具有数量少、耗时长的特点,比如网络训练一次可能需要数小时乃至几天的时间[1],才能得到一个较好的模型;而推理任务则相反,数量大,耗时短,以图像分类为例,识别一张图片的时间只有几十、几百毫秒[4],却可以为无数张图片做分类.而人工智能服务提供商对用户提供的服务以推理任务为主,因此推理计算的时延直接影响了服务质量和用户体验,减少时间开销变得十分重要.

然而神经网络算法包含大量矩阵乘法、向量矩阵乘法、向量内积等操作,具有很高的计算密集度和访存密集度,CPU通用处理器无法满足高吞吐、低时延的需求.因此学术界和产业界将目光转向了GPU[5]、FPGA[6]、神经网络加速器[4]等,通过这些专用处理器实现对神经网络计算任务的加速.实践证明,神经网络加速器能获得比通用处理器数百倍的性能提升和数十倍的功耗降低[1,5].

神经网络加速器作为一种协处理器,与CPU和内存的连接关系如图1所示.整个计算过程的数据流向是:①CPU将主机内存上的输入数据搬移到设备内存,②加速器从设备内存读入输入数据,按照指令完成计算,③加速器将计算结果写回设备内存,④CPU将设备内存中的输出数据搬移到主机内存.

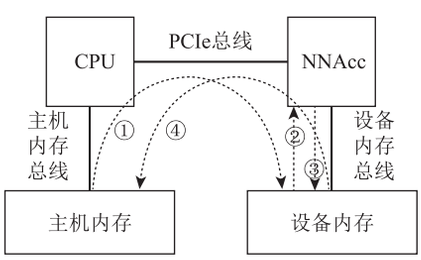

图1　神经网络加速器与CPU和内存的连接关系及其计算过程的数据流向

对于神经网络推理计算而言,其所需的输入数据包括输入样本数据(比如图片、语音等)、网络模型参数(例如权值、偏置等)、加速器硬件指令等,需要的输出数据包括推理结果(例如词向量、分类概率等).其中,硬件指令在网络拓扑结构和硬件参数确定后就可以编译生成,且不会变更,网络模型参数在训练稳定之后也不会频繁改动,因此它们可以被视为常量数据.而输入样本和推理结果在每次推理计算时是不同的,它们可以被视为变量数据.如图2所示,一份训练完成的权值和偏置等模型参数,可以参与多个输入样本的推理计算.

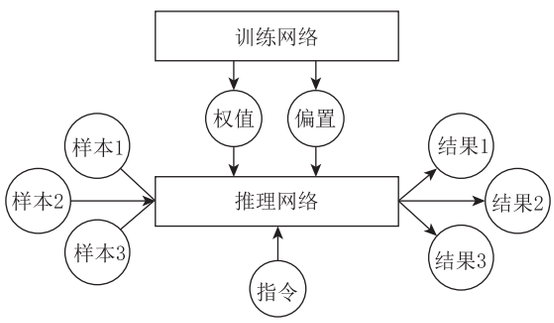

图2　推理网络常量、变量数据示例

对应图1的计算过程,变量数据的拷入拷出是必不可少的,因为每次推理计算的输入样本不同.但如果每次都拷入相同的常量数据,就会造成不必要的时间和资源浪费.解决推理过程中常量数据重复拷贝问题的一种方案是:拷入一次常量数据,拷贝多次变量数据,即可完成同一网络模型的多次推理计算,我们称之为"常变量异步拷贝".

然而,用户一般并不直接使用神经网络加速器的底层指令集,而是通过芯片设计者提供的更高层开发工具软件来开发和计算自己的神经网络.例如NVIDIA GPU使用的cuDNN①,寒武纪MLU使用的CNML②,AMD GPU使用的MIOpen③等,它们都是各公司针对自己的硬件产品而对外提供的编程库,方便用户使用,后文称之为神经网络开发工具软件(Neural network Development Tools,NDT).

这就产生了四个具有挑战性的问题:第一,如何在神经网络开发工具软件中实现常变量异步拷贝;第二,如何组织和管理神经网络中的众多常量数据,以实现常量数据的一次拷贝;第三,如何在神经网络加速器中解决指令对常量、变量不同数据的寻址问

① cuDNN: NVIDIA GPU deep learning library, https://developer.nvidia.com/cuDNN 2020,2,6

② CNML: Cambricon NeuWare Machine Learning Library, http://www.cambricon.com/index.php?m=content&c=index&a=lists&catid=71 2020,1,20

③ MIOpen: AMD Machine Intelligence Library, https://github.com/ROCmSoftwarePlatform/MIOpen 2019,4,10

题，防止取错数据；第四，提供什么样的编程模型可以简化用户编程，屏蔽实现细节.

在本文中，我们提出了一种基于常变量异步拷贝的神经网络编程模型QingLong及其开发工具软件ACCV-NDT（Neural network Development Toolswith Asynchronous Copy of Constant and Variable），来解决上述四个问题. 具体来说，我们的主要工作有：

（1）我们针对Cambricon-X硬件平台[7]设计并实现了“三段式”的ACCV-NDT，即包含了三个阶段的功能. 第一阶段是定义网络阶段，为用户提供一系列接口使其可以定义网络的拓扑结构和数据、操作的连接关系，并且实现网络数据节点绑定常量数据的功能. 第二阶段是编译网络阶段，完成硬件指令的生成和常量数据的打包等功能. 第三阶段是计算网络阶段，实现拷贝常量数据包、拷贝变量数据和计算网络的功能.

（2）我们设计了存储常量数据和各数据块空间大小的数据包（Data Package），在编译网络阶段按照“REOFF数据包装法”生成数据包，在计算网络阶段解析数据包，并完成常量数据从主机内存到设备内存的拷贝.

（3）我们为常量和变量设计了两种不同的寻址方式，“段直接寻址”和“段间接寻址”，便于指令读取常量和读写变量.

（4）我们抽象出基于常变量异步拷贝的神经网络编程模型QingLong，对应上述三个阶段，用户在第一阶段定义网络结构并且绑定常量数据，第二阶段调用一个接口编译网络，第三阶段拷贝一次常量数据，完成多个样本的计算.

（5）我们在Cambricon-X硬件平台上将QingLong与DLPlib[8]做了一系列对比实验，结果表明，在连续计算100份输入样本时，QingLong比DLPlib有平均17.48倍的性能提升，且输入样本越多，性能提升的倍数越大.

本文后续章节的内容包括，第二章介绍了神经网络加速器的相关工作，第三章给出了ACCV-NDT的设计方案及关键方法的细节，第四章抽象出QingLong编程模型，第五章做了一系列对比实验，第六章进行了总结，并规划了后续工作.

2 相关工作

2.1 神经网络加速器

近年来的学术研究和商业应用表明，人工神经网络的硬件计算器件主要有CPU、GPU、FPGA和ASIC这四类. CPU和GPU适用的领域最广，支持的神经网络算子最多，具有丰富的软件生态，Caffe[9]、TensorFlow[10]、MXNet[11]等主流神经网络编程框架软件都支持CPU和GPU计算，因此非常适合神经网络算法的研究和快速验证. FPGA和ASIC可以针对神经网络算法而特别设计，因此适合神经网络专用体系结构的研究和开发. 神经网络加速器作为一种ASIC，比其他三者具有更快的速度、更低的功耗、更小的芯片面积和更低的使用成本[4]，是理想的商业应用产品，然而其软件生态和易用性仍需拓展.

Lee S. Y. 等人[12]在1987年即提出了一种由乘法器和加法器组成的二维卷积处理阵列，加速图像处理任务，但不支持其他神经网络算子. Kamp等人[13]在1989年设计了一个可计算二维卷积的芯片，满足了实时视频处理业务，不过该芯片只适用于7*7的卷积核，无法灵活扩展. Chakradhar等人[14]在2010年设计了一种可动态配置的支持不同规模的卷积神经网络协处理器，可以做每秒30帧的视频处理，可用于视频监控、人脸识别等场景，具备了一定的通用性.

当然，最著名的神经网络加速器是DianNao系列. Tianshi Chen等人[4]在2014年提出了DianNao加速器，其依据“存储靠近计算”的思想，设计了三种片上缓存，实现访存量和计算量的平衡. 随后以DiaoNao为基础设计了多核神经网络加速器DaDianNao[15]，其包含16个DianNao单核，扩大了片上存储容量，相比GPU有450倍的性能提升. 2015年，机器学习加速器PuDianNao[16]通过增加运算单元和流水线级数，支持了包括神经网络、SVM、k-means在内的7种机器学习算法，扩大了神经网络加速器芯片的适用范围，相比于GPU，其功耗降低了128倍，而芯片面积仅有3.51 mm^2. 同年，ShiDianNao[17]依据“传感器靠近计算”的思想，将神经网络加速器嵌入图像信号处理器，可直接处理摄像头采集的实时图像，获得比GPU快30倍的性能提升.

DianNao系列加速器目前最新的Cambricon-X由Shijin Zhang等人[7]在2016年提出，它加入了稀疏运算单元，既支持稀疏神经网络，也支持稠密神经网络，性能比DianNao提升7.23倍. 我们后面的实验就选择Cambricon-X作为硬件平台.

2.2　神经网络开发工具软件及编程模型

根据是否和处理器硬件相关,我们将神经网络开发工具软件分为两个层次:神经网络开发框架和神经网络编程库.

神经网络编程库是根据处理器硬件结构与指令集对神经网络算法进行了特殊的优化,使神经网络算法在特定硬件上具有较高的性能和计算效率,通常由芯片设计公司随硬件一起推出,例如NVIDIA针对自家的GPU推出的cnDNN和TensorRT①,AMD的GPU编程所使用的MIOpen,Intel也为自己的"至强"系列CPU开发了神经网络编程库MKL-DNN②.

神经网络开发框架并不针对具体硬件而设计,而是偏重于算法层面的优化,通常支持一种或多种硬件,也支持多硬件平台的任务协作,为用户设计和运行自己的神经网络提供方便,例如Caffe、TensorFlow、MXNet等,它们通过调用处理器的神经网络编程库接口来使用对应的硬件.当然,开发框架一般也会自己编写CPU平台的代码,而不使用Intel的编程库.

用户、开发框架、编程库和硬件的层次关系如图3所示.

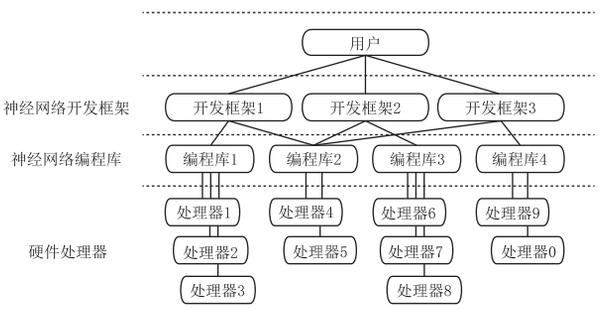

图3　神经网络开发工具层次关系

用户处于最高层次,可在众多神经网络开发框架中选择一个进行开发.开发框架可调用多个的神经网络编程库,从而支持在不同的硬件上计算.每个神经网络编程库通常支持自家的一系列处理器.本文后续部分我们设计的ACCV-NDT则属于神经网络编程库这一层次,但也会借鉴开发框架的一些编程模型.

根据编译和运行是否分离,我们将神经网络编程模型分为两种:编译运行耦合和编译运行解耦.前者是指编译和运行一起完成,编程模型只有"定义—计算"两个主要环节,cuDNN、PyTorch③和DLPlib就属于这一类,使用其接口描述神经网络结构之后,调用一个计算函数即可完成编译和运行两步操作.后者编译运行解耦是指编译和运行分多步完成,编译完成后通常会生成一个运算模型,在运行时按照运算模型的指导完成计算,代表工具是TensorRT、Caffe和TensorFlow等,其编程模型则包含"定义—编译—计算"三个主要环节.

编译运行解耦的特点是支持一次编译、多次运行,因此,相对于编译运行耦合在一起,前者更适合大批量计算神经网络推理任务,避免重复编译.

3　ACCV-NDT设计

在设计NDT之初,我们考虑到编程模型要分为用户输入网络和硬件计算网络两个阶段,而后者把编译和计算耦合在一起会使得每次计算都要重复编译,造成资源浪费,所以我们把编译、计算再拆分为两个阶段,最后的架构设计对应三个阶段:定义、编译、计算.这样可以做到定义网络一次,编译一次,计算多次.

在多次计算时,为了做到只拷贝一次网络中的所有常量数据,拷贝多次变量数据,我们计划在定义网络阶段由用户为数据节点绑定常量数据,在编译网络阶段将所有常量数据封装为一个数据包,在计算网络阶段实现一次拷入数据包,而无需用户逐一拷入每个常量.

ACCV-NDT的软件架构如图4所示,定义网络阶段实现三个数据结构,即数据节点、操作节点和神经网络.神经网络由数据节点和操作节点连接而成.数据节点允许绑定常量数据.编译网络阶段实现两个组件:指令编译器将神经网络编译为具体硬件的指令(使用的硬件平台架构为Cambricon-X);数据打包器将常量数据和记录各数据块空间大小的标签封装在一起.计算网络阶段包含三个单元:内存控制器负责申请、释放设备内存空间和控制主机内存与设备内存之间的双向数据拷贝;设备控制器负责开启、关闭加速器设备,启动计算,接收中断等;数据包解析器负责拆解数据包,提取常量数据和标签.

---

① TensorRT: NVIDIA neural network inference engine, https://docs. nvidia. com/deeplearning/sdk/tensorrt-developer-guide/index. html 2019,4,11

② MKL-DNN: Intel Math Kernel Library for Deep Neural Networks, https://software. intel. com/zh-cn/articles/intel-mkl-dnn-part-1-library-overview-and-installation? language=fr 2019,4,11

③ PyTorch: An open source deep learning platform. https://pytorch. org 2019,4,11

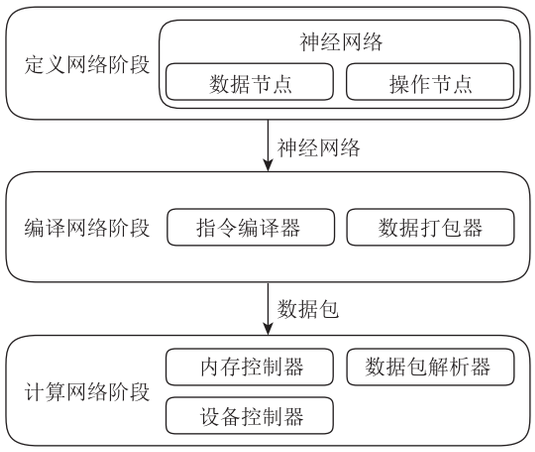

图4 ACCV-NDT软件架构

3.1 定义网络阶段

3.1.1 积木式网络组件

神经网络可以看做是一个积木成品，数据节点和操作节点可看成搭起积木的组件，因此我们把这三者统称为积木式网络组件. 对应到代码上，它们则是三个数据结构，其基本元素和操作如表1所示.

除了两种节点外，网络的搭建离不开边的连接，我们将边保存在节点指针中，通过一个节点可以找到其上下两层的节点. 为了简化用户编程，我们将建立边的过程隐藏在操作节点的创建函数中，用户只需创建数据节点和操作节点即可. 例如图5中conv+scale两层网络的结构可按照过程1的代码来描述. 图中的四个白色方块代表常量数据.

表1 三种积木式网络组件

组件名	基本元素	基本操作
数据节点 DtNode	数据精度、维度、大小等属性，生产者操作节点指针，消费者操作节点指针，常量数据指针	创建、销毁、设置数据属性、绑定常量数据
操作节点 OpNode	操作类型及参数(例如卷积操作的窗口大小、滑动步长等)，输入数据节点指针，输出数据节点指针	创建、销毁、设置操作参数
神经网络 NeuNet	网络操作节点，变量输入数据节点，变量输出数据节点，数据包	创建、销毁、编译、拷贝常量数据、参数化计算

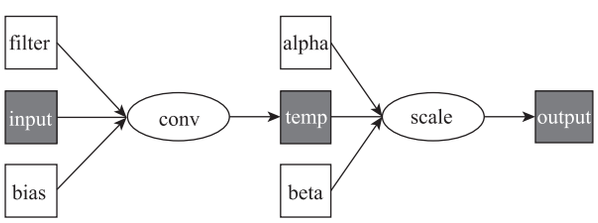

图5 conv+scale两层网络示例

过程1. conv+scale两层网络的代码描述.

```
DtNode input = new DtNode();
DtNode filter = new DtNode();
// bias, temp等数据节点略
ConvNode conv = new ConvNode(input, filter, bias, temp);
ScaleNode scale = new ScaleNode(temp, alpha, beta, output);
```

由于神经网络中既有数据节点也有操作节点，为了简化用户编程，在创建神经网络时，只需填入操作节点(因为操作节点数量比数据节点更少)，NDT根据操作节点的前后链接即可找到数据节点，如过程2所示. 为了保证后续计算时指令寻址不会出错，创建函数中还需要指明变量输入、输出节点及其顺序. 常量输入数据和网络中间隐藏层数据temp，我们会在后面的编译阶段统一处理.

过程2. 创建conv+scale网络的代码.

```
NeuNetnn = new NeuNet({conv, scale}, {input}, {output});
```

至此，一个简单的两层神经网络搭建完成.

3.1.2 绑定常量数据

对于数据节点，我们特别增加了绑定常量数据的操作，用于将推理过程中用到的模型参数记录在数据节点里，等待编译网络阶段的数据打包器使用. 如过程3所示，将保存常量数据的内存空间指针记录在数据节点中.

过程3. 数据节点绑定常量数据示例.

```
void* dataPtr = malloc(4096);
filter.bindConstant(dataPtr);
```

当然，不是所有权值、偏置等都必须绑定常量数据，而是根据实际需求，未绑定的都被视为变量数据节点.

3.2 编译网络阶段

本阶段完成硬件平台指令生成和常量数据打包的功能，是性能优化和实现常变量异步拷贝的关键步骤，对外提供的接口只有一个，如过程4所示.

过程4. 编译网络接口.

```
nn.compileNetwork();
```

其内部包含两个功能组件，指令编译器和数据

打包器,按顺序依次执行.在介绍两个组件前,先介绍数据包的结构设计.

3.2.1　数据包结构与设备内存数据排布

数据包的主体是常量数据,包含了指令和众多网络模型参数,但也包括记录数据块大小的标签.例如,在拷贝常量数据前,需要知道常量数据总大小来申请设备内存空间,因此我们把记录常量数据大小的标签也保存在数据包里.

还有网络中间隐藏层数据(例如图5中的temp),我们为了减轻用户负担,不再让用户为它们申请设备内存空间,所以数据包中也需要记录隐藏层数据大小的标签,但不需要保存数据内容,因为它们也是变量,会随输入的更换而变化.

再有,不同神经网络的输入、输出数量不同,不一定是单输入单输出,且它们的设备内存空间由用户申请,因此数据包不需要记录IO数据的大小,但为了指令能正确寻址,我们需要把IO的地址写在设备内存上,这就需要记录IO地址保存区的大小,我们将其记录在第三个标签中.

一个完整的数据包结构分为标签区和常量数据区,以图5网络为例,其数据包形式如图6所示.IO地址区16字节是因为该网络输入、输出数据一共两个,每个数据地址占8字节.

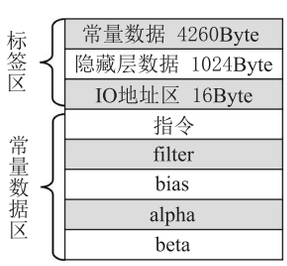

图6　数据包结构示例

数据包由数据打包器生成,保存在神经网络数据结构中,在计算网络阶段的常量拷贝函数中解析并使用,后文会介绍.

数据包是存储在主机内存中的,那么在设备内存中的数据排布也需要对应着数据包做精心规划.如图7所示,我们将设备内存分为"数据段"和"数据块"两级逻辑结构.数据包中的三个标签对应三个数据段:常量数据、隐藏层数据和IO地址区.每个段内包含若干数据块,例如常量数据区中的filter数据块、alpha数据块等.网络输入、输出数据空间是用户申请的,也作为数据块,但其地址填在IO地址区内.另外还需要一个段基址区记录三个数据段的起始地址,这样就具备了数据寻址的基本条件,把基址区作为入口,就能找到任意位置的数据.

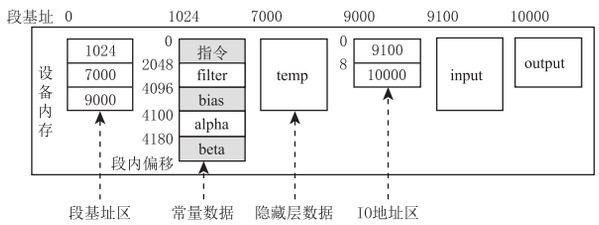

图7　设备内存数据排布示例

3.2.2　指令编译器与两种寻址方法

指令编译器依据硬件指令集,将网络算法映射为指令,这一过程不再详述.其中需要解决的一个重要问题是数据寻址.通过前面的设备内存排布设计可以发现,常量数据和隐藏层数据通过段基址加段内偏移就可以找到一个数据块,而IO数据通过段基址加段内偏移只能取到一个地址,再跳转到对应位置才能取到数据.因此我们为指令设计了两种寻址方法来匹配这两种情况,"段直接寻址"和"段间接寻址".

"段直接寻址"的目标地址=段基址+段内偏移+块内偏移,例如寻址filter第4字节,那么目标地址就是1024+2048+4=3076.该方法适用于常量数据和隐藏层数据的寻址.在指令编译时,块内偏移4会被固化在指令中;段内偏移2048会在数据打包器封装数据包后才能确定,再回填到指令中;段基址1024是在计算网络阶段由常量拷贝函数动态申请的,再填入基址区.

"段间接寻址"的目标地址计算分两步,第一步计算指针=段基址+段内偏移,从指针中取到块地址,第二步算目标地址=块地址+块内偏移,例如寻址output第6字节,那么目标地址就是9000+8=9008->10000->10000+6=10006.该方法适用于网络输入、输出数据的寻址.在指令编译时,块内偏移6会被固化在指令中;块地址10000是在计算网络阶段由用户申请,经参数化计算函数填入IO地址区内;段内偏移8会在数据打包器封装数据包后才能确定,再回填到指令中;段基址9000是在计算网络阶段由常量拷贝函数动态申请的,再填入基址区.

3.2.3　数据打包器与REOFF数据包装法

数据打包器的主要作用是将指令编译器生成的指令、用户绑定的常量数据和记录各数据段大小的标签有序组织在一起,并计算各数据块所在数据段内的偏移,再将段内偏移写入指令中.其详细过程

如图8所示，我们将这三个步骤称为“REOFF数据包装法”.

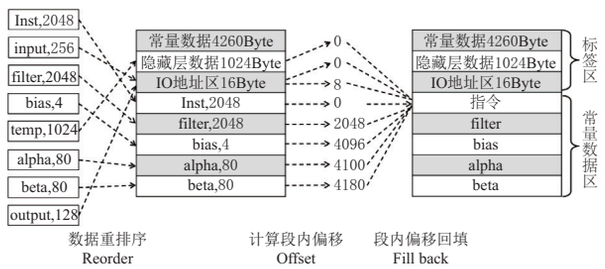

图8 REOFF数据包装法

第一步数据重排序，在全部数据节点中先找出常量数据，按照先指令后数据的顺序组成数据包的常量数据区，同时保存着各数据块大小，便于后续使用；再计算常量数据总大小，记录在标签中；然后找出所有隐藏层数据节点，计算它们的总大小，记录在标签中，最后统计输入节点数量，按照每节点8字节计算IO地址区所需大小，记录在标签中. 此时形成一个初级数据包.

第二步计算段内偏移，三个数据段独立计算各数据块在自己段内的偏移，数据块大小累加即可，IO地址区按8字节累加. 段内偏移记录在各数据节点中.

第三步段内偏移回填. 指令编译器在编译过程中已将需要填入偏移量的指令位置记录在各数据节点中，因此本步根据节点记录的位置填写上一步得出的偏移量即可. 至此一个完整的数据包就生成完毕，存入神经网络的数据结构中.

3.3 计算网络阶段

本阶段包括三个组件：内存控制器、设备控制器、数据包解析器，但对外功能有两类，常量拷贝和参数化计算，它们是由三个组件的部分功能拼接而成.

3.3.1 常量拷贝

常量拷贝功能即对数据包内常量和标签向设备内存做了复制，其对外接口如过程5所示.

过程5. 常量拷贝接口

```
nn.copyConstant();
```

函数内部的实现原理如图9所示. ①～③步调用内存控制器，依据标签记录的数据段大小，在设备内存上分别申请常量数据、隐藏层数据、IO地址区的存放空间. 第④步将常量数据区整体拷入设备内存. 第⑤步将三个数据段的基址，填入段基址区，此区域相对固定，由内存控制器保证不会分配做其他用途，硬件处理器也会默认从此区域取段基址.

函数执行完后，隐藏层数据段和IO地址区还是空的，前者在硬件计算过程中被写入和读出，后者在神经网络计算函数中被填入.

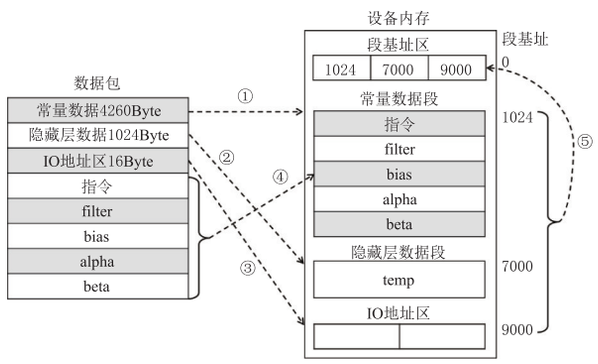

图9 常量拷贝函数原理

3.3.2 参数化计算

参数化计算是一系列函数的统称，为用户提供的功能包括申请、释放设备内存空间，主机内存与设备内存之间的双向数据拷贝，神经网络计算等. 在计算时的这些函数的调用流程如过程6所示. 用户先申请输入、输出数据的设备内存空间，然后从主机内存拷入样本输入数据，计算神经网络，再将输出结果拷出至主机内存，最后释放设备内存空间. 其中“拷输入—计算—拷输出”是可以循环多次做，实现了拷入一次常量数据，拷贝多次变量数据，计算多次的目标.

过程6. 参数化计算系列函数调用流程

```
void* inputPtr = devMalloc(input.size());
void* outputPtr = devMalloc(output.size());
devMemcpy(inputPtr, inputData, HOST_TO_DEV);
// inputData是主机内存指针，保存了样本输入数据
nn.computeNetwork({inputPtr}, {outputPtr});
devMemcpy (outputData, outputPtr, DEV_
TO_HOST);
// outputData是主机内存指针，用于接收输出结果
devFree(inputPtr); devFree(outputPtr);
```

在这一系列函数中，最具特色的是神经网络计算函数，它的参数列表中接收了两个地址，这也是参数化计算名称的由来. 该地址的填入顺序需保持和过程2中创建网络时的输入、输出数据节点的顺序一致，才能保证寻址的正确性. 该函数执行时先将接收的地址填入IO地址区，再使用设备控制器启动硬件计算，等待计算完成的中断返回.

整个参数化计算过程的设备内存变化情况如图10所示.

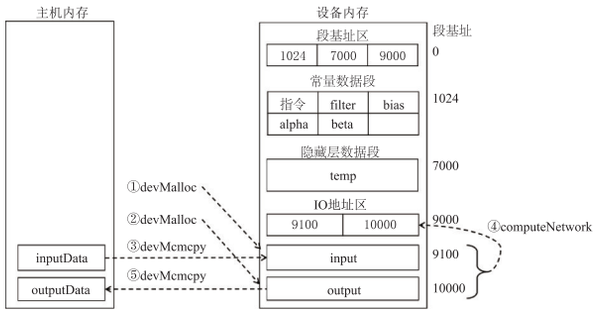

图10　参数化计算过程中设备内存的变化

在参数化计算环节,我们将网络输入、输出数据的设备内存申请和拷贝功能留给用户做的考虑是,用户可以自己做“拷输入—计算—拷输出”的三级流水线,从而将变量数据拷贝的时间“隐藏”在计算时间内,对于大数据量的推理计算而言,进一步压缩了总处理时间.做流水计算时,用户会申请多份输入、输出空间,在网络计算函数中填入不同的地址即可.流水计算原理如图11所示.

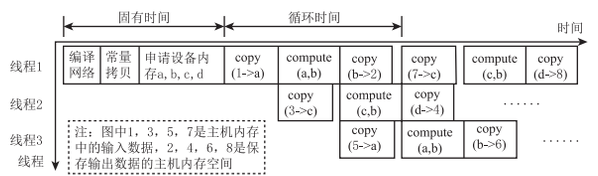

图11　参数化计算环节的三级流水计算原理

4　QingLong编程模型

根据前一章ACCV-NDT的架构和接口,我们抽象出QingLong编程模型.该模型同样分为三个阶段.

在定义网络阶段,用户使用积木式网络组件定义网络的拓扑结构,并可以给数据节点绑定常量数据.

在编译网络阶段,用户调用编译接口完成指令生成和常量数据打包.

在计算网络阶段,用户拷贝一次常量数据,为输入、输出申请设备内存空间,然后就可以多次拷入样本输入、计算网络、拷出结果.

编程模型流程如图12所示.

该模型具有三个优势:

(1)网络一次编译,多次计算,避免重复编译;

(2)计算阶段实现常量、变量异步拷贝,即拷入一次常量,拷贝多次输入、输出变量,避免常量数据的重复拷贝;

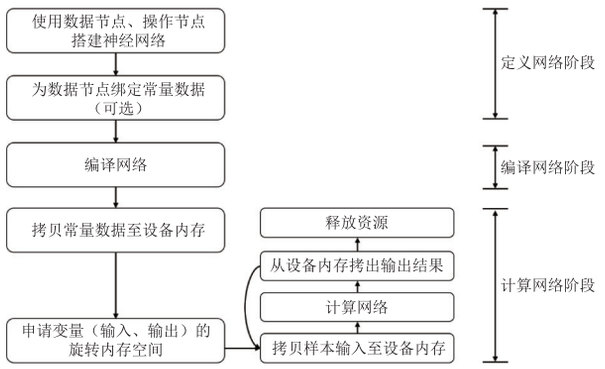

图12　QingLong编程模型

(3)计算阶段的“拷输入—计算—拷输出”循环,可以做成三级流水线,在多批次数据处理时,可将变量数据拷贝的时间“隐藏”在计算时间内,进一步压缩总处理时间.

5　实　验

本章我们在Cambricon-X模拟器上做了一系列对比实验,对比的对象是DLPlib和QingLong的三个版本.四个对象的区别如表2所示.

表2　四个实验对象及其区别

对象	是否支持一次编译多次计算	是否支持常变量异步拷贝	是否支持三级流水线
DLPlib	否	否	否
QingLong1	是	否	否
QingLong2	是	是	否
QingLong3	是	是	是

我们之所以把QingLong编程模型拆分为三个子版本,一是因为我们想清楚地知道每一项优化方法会带来多少性能的提升,二是因为用户不一定使用全部的优化方法.比如在资源受限的终端计算平台上,可能无法实现多线程的三级流水,那么可以以QingLong2为参考.所以我们根据cuDNN、TensorRT、MXNet等当前主流神经网络开发工具软件的编程模型特性,划分了三个子特性:是否支持一次编译多次计算,是否支持常变量异步拷贝和是否支持三级流水线.QingLong三个子版本的划分就依据这三个特性的有无.

在前面的相关工作中,我们介绍了编译运行耦合和编译运行解耦两种神经网络编程模型,DLPlib属于前者.由于Cambricon-X平台目前还没有编译运行解耦的编程模型与编程库可做为对比,因此我们选用DLPlib作为对照.

5.1 实验平台

我们用Verilog语言编写了一个Cambricon-X的模拟器，使用VCS（Synopsys Verilog Compiler Simulator）编译并模拟该架构. 模拟器具备16个处理单元（processing element，PE），每个PE具有16个半精度浮点数乘法器和一个半精度浮点加法树，也就是可以一次计算16个半精度浮点数的向量点乘. 半精度浮点数的格式有1位符号位，5位指数位，10位尾数位，一共16bit. 每个PE包含一个2KB的权值缓存SB（synapse buffer），所有16个PE共用一个8KB的输入神经元缓存NBin（neuron buffer）和一个8KB的输出神经元缓存NBout. 该模拟器的频率是1GHz. 设备内存8GB，主机内存8GB，主机CPU是core i7 8700K，主频3.7GHz.

5.2 测试集

我们选择8个单层网络和2个多层神经网络进行测试，10个测试用例都用四个实验对象的编程模型实现. 多层网络包括两个做图像分类的推理网络AlexNet和VGG16，单层网络选择的是四类常用的带有常量数据的操作：卷积（Conv）、全连接（FC，同MLP）、缩放（Scale）、批规范化（batch norm，BN），每类操作有两组测试参数，如表3所示.

表3 实验测试集（单层网络）

操作	参数
Conv1	input C, H, W: 96, 112, 112; kernel number, C, H, W: 384, 96, 3, 3; stride H, W: 1, 1; pad H, W: 1, 1
Conv2	input C, H, W: 96, 112, 112; kernel number, C, H, W: 256, 96, 3, 3; stride H, W: 1, 1; pad H, W: 1, 1
FC1	input size: 4096; output size: 4096
FC2	input size: 4096; output size: 1000
Scale1	input C, H, W: 96, 112, 112; alpha size: 96; beta size: 96
Scale2	input C, H, W: 256, 112, 112; alpha size: 256; beta size: 256
BN1	input C, H, W: 96, 112, 112; alpha size: 96; beta size: 96
BN2	input C, H, W: 256, 112, 112; alpha size: 256; beta size: 256

我们对这10个测试用例都输入100组输入数据，分别统计四个实验对象从开始编译到100组数据全部计算完成并拷出到主机内存的时间，连续做三次取平均值.

然后我们对两个多层神经网络再测试200组输入、300组输入至1000组输入的处理时间，模拟实际使用场景. 这里的“处理时间”同样是指编译加计算总时间.

我们选择编译加计算总时间，而不选择纯计算时间的原因，一方面是不同的编程模型对编译、计算的耦合情况不同，对于编译计算耦合的编程模型，我们无法将纯计算时间剥离出来，另一方面在实际推理业务场景中，编译时间也是算在服务总时延内的，因此我们选择编译加计算总时间作为实验对比量.

5.3 实验结果与分析

对于10个用例100组输入的测试，QingLong相对DLPlib的加速比如图13所示.

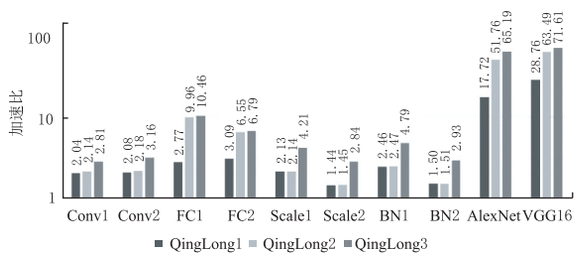

图13 100组输入的处理时间相对于DLPlib的加速比

可以发现，同时支持一次编译、多次计算，常变量异步拷贝和三级流水线这三种优化的QingLong3加速效果最好，前8个单层网络相比于DLPlib有平均4.75倍的提升，后两个多层网络则有平均68.4倍的提升，10个测试用例整体平均加速17.48倍.

对比图13中的QingLong2和QingLong1两组数据，可以发现Scale和BN的QingLong2和QingLong1的加速比非常接近，原因是Scale和BN的常量数据量小于输入输出等变量数据量，常变量异步拷贝所发挥的作用较小，提升倍数还不到1%. 而FC的权值等常量数据量多于输入输出等变量数据量，QingLong2平均比QingLong1提升了186%. Conv则介于两者之间，提升倍数适中，平均提升了约5%.

由此可以得出结论：网络中的权值、偏置等常量数据越多，常变量异步拷贝的加速比越大.

需要特别指出的是：(1)对于Scale和BN等常量数据较少的算子，甚至ReLU、LRN等没有常量数据的算子而言，常变量异步拷贝的性能提升效果不明显，甚至没有提升，但实际网络中一般都存在conv、mlp等包含常量数据的算子，因此会有性能提升；(2)由于编译阶段增加了常量数据打包的过程，QingLong2的单次编译时间会大于QingLong1，特别是在只有一组输入的时候，QingLong2的编译加计算总时间大于QingLong1，没有性能收益，但在实际推理业务中，一般不会出现一组输入的情况，通常

输入组数较多,单次编译增加的时间“平摊”到多次计算上,性能会有提升.

此外,对比图13中的QingLong3和QingLong2两组数据,还可以发现FC的QingLong3平均比QingLong2提升了4.3%,小于Conv的平均38%和Scale和BN的平均95%,这是因为FC的计算时间是输入输出拷贝时间的平均18倍,在“拷输入—计算—拷输出”的三级流水中,计算的时间占了主要部分,因此流水线的提升倍数不如其他单层网络的大.

在实验中还可以发现,QingLong对实际多层网络的提升倍数要高于单层网络,单层网络平均4.75倍,多层网络平均68.4倍.这主要是因为多层网络所含的常量数据量多于单层网络,常变量异步拷贝可以做到只拷贝一次常量数据,这就减少了常量数据的拷贝量,因此性能收益更明显.

对于两个多层网络的100至1000组输入测试,QingLong相对DLPlib的加速比如图14和图15所示.

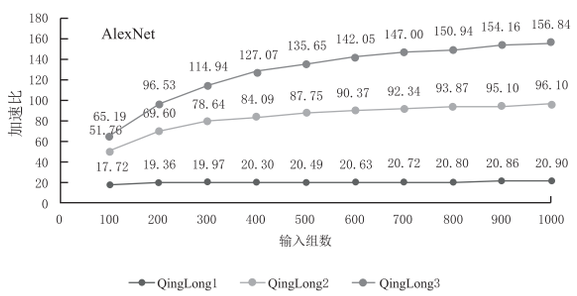

图14　AlexNet在输入递增时QingLong相对于DLPlib的加速比

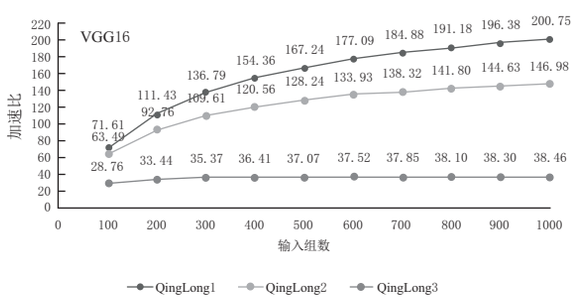

图15　VGG16在输入递增时QingLong相对于DLPlib的加速比

可以发现,QingLong2的性能比QingLong1平均提升3.73倍,且输入样本越多,加速比越大.这一趋势与前面单层网络的测试结果一致,因为网络中的权值、偏置等常量数据只拷贝了一次,输入组数越多,“节省”的常量拷贝次数就越多,性能提升倍数越大.

另外,QingLong3在三者当中的提升倍数依然是最高的,性能比QingLong2平均提升1.4倍,且输入样本越多,加速比越大,在1000组输入时,比DLPlib平均有178.8倍的提升.这种趋势与单层网络的测试结果一致,因为三级流水相当于将变量数据拷贝的时间“隐藏”在了计算时间内,输入组数越多,“隐藏”的变量数据拷贝时间就越多,性能提升倍数越大.

由此得出结论,QingLong编程模型在三种优化全部开启时处理速度的提升倍数最高,且输入样本越多,速度提升倍数越高.

6　结　论

本文先设计了基于常变量异步拷贝的神经网络开发工具软件ACCV-NDT,解决了推理过程中常量数据的重复拷贝问题,在其中还定义了存储常量数据的数据包,提出了REOFF数据包装法,针对常量和变量的特点设计了“段直接寻址”和“段间接寻址”两种不同的寻址方式;然后抽象出基于常变量异步拷贝的神经网络编程模型QingLong,既简化了用户编程,又实现了编译计算分离、常变量异步拷贝、流水计算等功能;最后通过实验测得,在100组输入数据时,QingLong比DLPlib有平均17.48倍的性能提升,对于1000组输入数据的实际网络则有平均178.8倍的性能提升.

我们未来的工作会倾向于适配更多的操作,运行更多的网络,探究多核神经网络加速器的常变量异步拷贝方法,改造QingLong,使其兼容单核和多核平台等.

参　考　文　献

[1] Krizhevsky A, Sutskever I, Hinton G E, et al. ImageNet Classification with Deep Convolutional Neural Networks. Neural information processing systems, 2012, 25(2): 1097-1105

[2] Gregor K, Danihelka I, Graves A, et al. DRAW: A Recurrent Neural Network for Image Generation// Proceedings of the 32nd International Conference on Machine Learning. Lille, France, 2015: 1462-1471

[3] Alahi A, Goel K, Ramanathan V, et al. Social LSTM: Human Trajectory Prediction in Crowded Spaces// Proceedings of the Computer Vision and Pattern Recognition. Las Vegas, USA, 2016: 961-971

[4] Tianshi Chen, Zidong Du, Ninghui Sun, et al. DianNao: a

small-footprint high-throughput accelerator for ubiquitous machine learning// Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA, 2014: 269-284

[5] Dan Claudiu Cireşan, Meier U, Gambardella L M, et al. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. Neural Computation, 2010, 22(12):3207-3220

[6] Gupta S, Agrawal A, Gopalakrishnan K, et al. Deep Learning with Limited Numerical Precision// Proceedings of the 32nd International Conference on Machine Learning. Lille, France, 2015: 1737 - 1746

[7] Zhang S, Du Z, Zhang L, et al. Cambricon-X: an accelerator for sparse neural networks// Proceedings of the International Symposium on Microarchitecture. Taipei, China, 2016: 1-12

[8] Lan H Y, Wu L Y, Zhang X, et al. DLPlib: A Library for Deep Learning Processor. Journal of Computer Science and Technology, 2017, 32(2):286-296

[9] Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional Architecture for Fast Feature Embedding// Proceedings of the ACM Multimedia. Orlando, USA, 2014: 675-678

[10] Abadi M, Barham P, Chen J, et al. TensorFlow: A system for large-scale machine learning// Proceedings of the Operating Systems Design and Implementation. Savannah, USA, 2016: 265-283

[11] Chen T., Li M., Li Y., et al. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems// Proceedings of the Neural Information Processing Systems, Workshop on Machine Learning Systems. Montreal, Canada, 2015

[12] Lee S Y, Aggarwal J K. Parallel 2-D Convolution on a Mesh Connected Array Processor. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1987, 9(4): 590-594

[13] Kamp W, Kunemund R, Soldner H, et al. Programmable 2D linear filter for video applications. IEEE Journal of Solid-State Circuits, 1990, 25(3):735-740

[14] Chakradhar S, Sankaradas M, Jakkula V, et al. A dynamically configurable coprocessor for convolutional neural networks// Proceedings of the International Symposium on Computer Architecture. Saint-Malo, France, 2010: 247-257

[15] Chen Y, Luo T, Liu S, et al. DaDianNao: A Machine-Learning Supercomputer// Proceedings of the International Symposium on Microarchitecture. Cambridge, UK, 2014: 609-622

[16] Liu D, Chen T, Liu S, et al. PuDianNao: A Polyvalent Machine Learning Accelerator// Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems. Istanbul, Turkey, 2015: 369-381

[17] Du Z, Fasthuber R, Chen T, et al. ShiDianNao: shifting vision processing closer to the sensor// Proceedings of the International Symposium on Computer Architecture. Portland, USA, 2015: 92-104

Du Wei-Jian, Ph. D. His research interests include deep learning and programming library of neural network accelerator.

Chen Yun-Ji, Ph. D. supervisor. His research interests include computer architecture and intelligent computation.

Zhi Tian, Ph. D. Her research interests include reconfigurable architecture and integrated circuit design.

Wu Lin-Yang, Ph. D. His research interests include deep learning and programming library of neural network accelerator.

Chen Xiao-Bing, Ph. D. His research interests include deep learning and compiler of neural network accelerator.

Zhuang Yi-Min, Ph. D. His research interests include deep learning and compiler of neural network accelerator.

Background

Neural network accelerator has become a research hotspot in recent years because of its high performance and low power in artificial intelligence applications. The role of its coprocessor determines that it needs to move data between host memory and device memory before and after computing. In the high throughput flow-based neural network inference service, this data movement will extend the total processing time and affect the user experience. In these data, constant data such as network model parameters and hardware instructions are the same in each inference calculation, and they are copied repeatedly in each inference calculation, which wastes bandwidth resources and processing time. Therefore, we can realize the function of copying constant data once and copying input and output variables multiple times in the development tool software of neural network accelerator. This research of constant variable asynchronous copy is relatively

scarce. This paper presents a neural network development tool software based on asynchronous copy of constant and variable and its programming model QingLong to solve the problem of repeated copy of constant data. QingLong programming model integrates the function of asynchronous copy of constant and variable into three stages: defining network, compiling network and computing network. It provides simple programming interface for users and shields internal implementation details. Experiments show that QingLong has an average performance improvement of 17.48x over DLPlib when calculating 100 input samples continuously, and the more input samples, the greater the performance improvement. The main research direction of our group is the programming library of the neural network accelerator. The related work has been published in journals, such as Chinese Journal of Computers and High Technology Letters.

This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFA0700900, 2017YFA0700902, 2017YFA0700901, 2017YFB1003101, 2018AAA0103300), the NSF of China (under Grants 61432016, 61532016, 61672491, 61602441, 61602446, 61732002, 61702478, 61732007 and 61732020), Beijing Natural Science Foundation (JQ18013), National Science and Technology Major Project (2018ZX01031102), the Transfor-mation and Transfer of Scientific and Technological Achieve-ments of Chinese Academy of Sciences (KFJ-HGZX-013), Key Research Projects in Frontier Science of Chinese Academy of Sciences (QYZDB-SSW-JSC001), Strategic Priority Research Program of Chinese Academy of Science (XDB32050200, XDC01020000), Standardization Research Project of Chinese Academy of Sciences (BZ201800001), Beijing Academy of Artificial Intelligence (BAAI) and Beijing Nova Program of Science and Technology (Z191100001119093).