软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(9):2654−2677 [doi: 10.13328/j.cnki.jos.005946]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向实时应用的深度学习研究综述*

张政馗, 庞为光, 谢文静, 吕鸣松, 王 义

(东北大学 计算机科学与工程学院 智慧系统实验室,辽宁 沈阳 110819)

通讯作者: 张政馗, E-mail: zhangzhengkui@cse.neu.edu.cn

摘 要: 深度学习算法和 GPU 算力的不断进步,正促进着人工智能技术在包括计算机视觉、语音识别、自然语言处理等领域得到广泛应用.与此同时,深度学习已经开始应用于以自动驾驶为代表的安全攸关领域.但是,近两年接连发生了几起严重的交通事故表明,深度学习技术的成熟度还远未达到安全攸关应用的要求.因此,对可信人工智能系统的研究已经成为了一个热点方向.对现有的面向实时应用的深度学习领域的研究工作进行了综述,首先介绍了深度学习技术应用于实时嵌入式系统所面临的关键设计问题;然后,从深层神经网络的轻量化设计、GPU 时间分析与任务调度、CPU+GPU SoC 异构平台的资源管理、深层神经网络与网络加速器的协同设计等多个方面对现有的研究工作进行了分析和总结;最后展望了面向实时应用的深度学习领域进一步的研究方向.

关键词: 深度学习;深层神经网络;实时系统;时间分析;实时调度;共享资源冲突

中图法分类号: TP181

中文引用格式: 张政馗,庞为光,谢文静,吕鸣松,王义.面向实时应用的深度学习研究综述.软件学报,2020,31(9):2654−2677. http://www.jos.org.cn/1000-9825/5946.htm

英文引用格式: Zhang ZK, Pang WG, Xie WJ, Lü MS, Wang Y. Deep learning for real-time applications: A survey. Ruan Jian Xue Bao/Journal of Software, 2020,31(9):2654−2677 (in Chinese). http://www.jos.org.cn/1000-9825/5946.htm

Deep Learning for Real-time Applications: A Survey

ZHANG Zheng-Kui, PANG Wei-Guang, XIE Wen-Jing, LÜ Ming-Song, WANG Yi

(Smart System Laboratory, School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: The persistent advance of deep learning algorithms and GPU computing power have promoted artificial intelligence in various fields including but not limited to compute vision, speech recognition, and natural language processing. Meanwhile, deep learning already began exploiting its usage in safety-critical areas exemplified by self-driving vehicles. Unfortunately, the successive severe traffic accidents in the past two years manifest that deep learning technology is still far from mature to fulfill safety-critical standards, and consequently the trustworthy artificial intelligence starts to attract a lot of research interests worldwide. This article conveys a state-of-the-art survey of the research on deep learning for real-time applications. It first introduces the main problems and challenges when deploying deep learning on the real-time embedded systems. Then, a detailed review covering various topics is provided, such as deep neural network lightweight design, GPU timing analysis and workload scheduling, shared resource management on the CPU+GPU SoC platform, deep neural network and network accelerator co-design. Finally, open issues and research directions are identified to conclude the survey.

Key words: deep learninig; deep neural network; real-time systems; timing analysis; real-time scheduling; shared-resource interference

* 基金项目: 国家自然科学基金(61532007, 61772123); 装备预研教育部联合基金青年人才基金(6141A020333)

Foundation item: National Natural Science Foundation of China (61532007, 61772123); Ministry of Education Joint Foundation for Equipment Pre-Research (6141A020333)

本文由“智能嵌入式系统”专题特约编辑王泉教授、吴中海教授、陈仪香教授、苗启广教授推荐.

收稿时间: 2019-07-08; 修改时间: 2019-08-18; 采用时间: 2019-11-02;jos 在线出版时间: 2019-12-05

CNKI 网络优先出版: 2019-12-05 14:55:12, http://kns.cnki.net/kcms/detail/11.2560.TP.20191205.1454.006.html

深度学习(deep learning,简称 DL)[1]是机器学习(machine learning,简称 ML)[2]的一个分支,都是能够让计算机根据数据进行预测,并且改进其预测或行为的一组方法.其核心思想是:在训练阶段,以最小化损失函数(loss function)为引导,通过梯度下降算法(gradient descent)来调整计算模型的权重(weight)和偏置(bias)两种参数;在推理阶段,则使用输入数据和训练好的模型参数来计算预测值.深度学习的主要特征是采用了分层的深层神经网络(deep neural network,简称 DNN)模型:每一层抽象出不同的简单特征,不同层的简单特征可以叠加组合成更复杂的特征,并使用这些组合的复杂特征解决问题[3].相比而言,传统机器学习算法很难抽象出足够有效的特征.现在的 DNN 模型已经有百万个人工神经元,深度达到了几十层.例如 2015 年,ImageNet 图像分类比赛(ImageNet large scale visual recognition challenge,简称 ILSVRC)的冠军是 ResNet,其最大深度为 152 层,属于超深神经网络.

随着大数据、深度学习算法与高性能计算技术的不断发展,深度学习在人工智能(artificial intelligence,简称 AI)领域一些最困难的问题上取得了重大突破,并且被应用于图像识别、语音识别、自然语言处理、自动驾驶、生物信息处理、电脑游戏、搜索引擎、广告推荐、医学自动诊断和金融分析等各大领域.现如今,物联网、可穿戴设备正在从概念变成现实,5G 网络建设已全面起步,这些全新的信息技术必将促进人工智能技术从工作站和服务器端迁移到各种嵌入式终端.这其中也包括了安全攸关系统,典型案例是全球互联网巨头和汽车巨头正如火如荼地研究以深度学习为基础的自动驾驶系统.特斯拉公司在 2015 年已经将自动驾驶技术(autopilot)投入了商用.然而,近两年接连发生的几起关于自动驾驶汽车的严重交通事故表明,深度学习技术的成熟度还远未达到安全攸关应用的要求.事实上,无论从功能层面还是非功能层面,深度学习赋能的安全攸关系统亟待有效的验证技术来保障整个系统计算逻辑的正确性和响应外部事件的实时性[4].

在功能层面保证 DNN 的正确性会面临 3 大困难[5,6]:(1) DNN 具有鲁棒性缺陷,对抗样本的轻微扰动就可能导致 DNN 给出错误的判断,这对安全攸关系统是一个致命的问题;(2) DNN 不具备可解释性,这导致无法对 DNN 模型进行(精确的)形式化建模和属性描述;(3) DNN 无穷的输入空间、大量的神经元、非线性激活函数会导致验证时发生状态空间爆炸.针对这些挑战,学术界在 DNN 的可解释性(interpretability)[2]、建模(modeling)[7]、测试(testing,把神经网络看作一个组件进行白盒测试,力求让测试案例最大化地覆盖整个网络结构,包括神经元、条件和分支等)[8]、证伪(falsification,又称半形式化验证(semi-formal verification),其目的是产生让整个系统违反设计规范的极端测试案例(corner test case))[9,10]、验证(verification)[11−13]等方面展开了大量的研究.此外,欧美政府也开始重视可信人工智能方向的研究.2017 年,美国国防部高级研究计划局(Defense Advanced Research Projects Agency,简称 DARPA)发起了“XAI 计划”,旨在研究如何更好地理解 AI 系统的预测逻辑.2018 年,欧盟颁布的 GDPR(general data protection regulation)要求 AI 算法具有透明性和可解释性.宾夕法尼亚大学成立了安全 AI 研究中心(PRECISE center for safe AI)[14],并致力于研究安全 AI 系统的设计和形式化验证技术.

在非功能层面保证 DNN 的实时性是指:在实时嵌入式系统中,DNN 任务(特别说明:本文中的深度学习和 DNN 任务都是指深度学习的推理阶段(inference phase),即用训练好的 DNN 模型和输入数据进行预测)的运行时间不能超过该任务所指定的时限.而实现这一目标同样面临 3 个方面的挑战:第一,嵌入式平台的计算资源和内存资源是非常有限的,电源供给也受电池容量的限制,散热条件也远比不上配备了大型冷却设备的数据中心;第二,DNN 任务已经被普遍部署在多核 CPU+GPU SoC 异构嵌入式平台(例如 NVIDIA Jetson TX2[15], NVIDIA DRIVE PX2[16]),但是学术界对调度异构多核嵌入式系统的理论研究还不够成熟;第三,目前为了保证实时性,一般让 GPU 只负责运行某种特定的 DNN 任务,例如图像识别,但在未来,需要在 GPU 上并行运行多个混合关键任务,例如未来的自动驾驶汽车上,GPU 可能同时并行处理多个 DNN 推理任务(图像识别、语音识别、自然语言处理)、DNN 在线训练任务[17]、容错备份任务等关键任务[18],同时还需要执行仪表盘的信息显示任务、多媒体或游戏系统的 2D/3D 渲染任务等非关键任务,这种多任务需求更增加了实时嵌入式操作系统实施任务调度和资源管理的复杂度.

近两年,实时系统学术界开始关注深度学习所带来的新问题.其中,德克萨斯大学 Liu 等学者在 DNN 网络的实时优化设计和 DNN 网络的能耗分析方面做了开创性的研究[19,20],北卡罗莱纳大学教堂山分校 Smith 教授对 NIVIDA GPU 的调度机制进行了深入细致的探究[15,21].但是,目前还缺乏对实时深度学习推理系统的主要设计

难点以及分析的方法进行系统且深入阐述的文献资料.针对上述问题,本综述介绍了深度学习推理过程应用于实时嵌入式系统所面临的关键设计问题,并比较了现有主要解决方案的优缺点.针对深层神经网络的轻量化设计、GPU 时间分析与任务调度、CPU+GPU SoC 异构平台的资源管理、深层神经网络与网络加速器的协同设计等多个方面,对现有的研究工作进行了分析和总结,旨在为相关领域的研究者提供参考.

1 背景知识

本节简要介绍深度学习的发展史、基本概念、深层神经网络 DNN、DNN 运行框架以及运行 DNN 实时任务的嵌入式硬件平台,包括嵌入式 GPU 和网络加速器.

1.1 深度学习简史

众所周知,深度学习已经成为了当下最流行的技术词汇,似乎是一项才刚刚发明的新技术.但是事实上,有关深度学习的研究历史(见表 1)可以追溯到 1943 年由 McCulloch 教授和 Pitts 教授提出的 McCulloch-Pitts Neuron 单神经元计算结构[22].该结构会通过 N 个权重与 N 个输入来计算加权和作为输出,是现代神经元结构的雏形.1958 年,Rosenblatt 教授提出的感知器(perceptron)模型[23]是首个可以根据样本数据来学习特征权重的模型,对现代机器学习的发展产生了巨大影响.1969 年,Minsky 教授在文献[24]中证明:感知器模型只能解决线性可分问题,无法解决异或问题,并且给出了“基于感知器的研究注定将失败”的结论.这导致了神经网络的第一次重大低潮期,在之后的 10 多年内,神经网络的研究几乎处于停滞状态.

Table 1 Scientific milestones and big news on deep learning history[3,25]

表 1 深度学习发展史上的重大科学及新闻事件[3,25]

时间	阶段	事件
1943	第 1 阶段	单神经元计算结构
1958		感知器模型
1969		神经网络第一次重大低潮
80 年代末	第 2 阶段	分布式知识表达与反向传播算法
90 年代初		CNN,RNN,LSTM 得到很好发展
2012	第 3 阶段	AlexNet 获得 ILSVRC 冠军
2013		深度学习被 MIT 评为“年度十大科技突破之一”
2015		特斯拉将自动驾驶功能投入商用
2016		AlphaGo 击败世界围棋冠军李世石
2017		AlphaGo 击败世界排名第一的围棋高手柯洁
2019		AlphaStar 战胜《星际争霸》职业选手

这期间,人们在认知生物学领域取得了重要进展,发现了认知的两个重要机制:一个是抽象,另一个是迭代.从原始信号,做低层抽象,逐渐向高层抽象迭代,在迭代中抽象出更高层的模式[25].到了 20 世纪 80 年代末,分布式知识表达(distributed representation)[26]和反向传播算法(back propagation)[27]的提出,开启了神经网络研究的第二阶段的序幕.分布式知识表达是深度学习的重要性质,基本思想是:先让每个神经元表达简单的特征,再把神经元组合起来用于描述复杂的特征.相比本地式知识表达(即一个特征对应一个神经元),分布式表达描述相同大小的特征空间需要的神经数量少很多,极大节约了存储空间.1986 年,Rumelhart, Hinton 和 Williams 这 3 位教授提出的反向传播算法则大幅降低了训练神经网络所需要的时间.时至今日,反向传播算法仍是训练深层神经网络的主要方法.同期,计算机的飞速发展也促进了卷积神经网络(convolutional neural network,简称 CNN)、递归神经网络(recurrent neural network,简称 RNN)、长短期记忆模型(long short-term memory,简称 LSTM)等模型的发展.

到了 2010 年左右,计算机性能的极大提升和大数据互联网+的发展,使得曾经阻碍神经网络发展的计算力和训练样本量问题得到了解决,从此深度学习的发展一日千里.2012 年,ImageNet 举办的 ILSVRC 图像分类竞赛中,由 Krizhevsky 教授实现的深度学习系统 AlexNet[28]赢得了冠军.此后,深度学习算法的性能在图像识别领域已经完全碾压了传统机器学习算法[29](如支持向量机 SVM 等),并且在 2013 年之后的 ILSVRC 中基本就只有深

度学习算法参赛了.2013 年,深度学习被麻省理工(MIT)评为“年度十大科技突破之一”.2015 年 10 月,特斯拉在 Model 系列车型上开启了自动驾驶功能(autopilot),标志着基于深度学习的自动驾驶技术已开始进入了商用阶段.2016 年和 2017 年,Google 的子公司 DeepMind 基于深度学习研发的 AlphaGo 击败了围棋大师李世石和柯洁,一度引发了全世界对人工智能的恐慌.2019 年 1 月,DeepMind 研发的 AlphaStar 击败了经典战略游戏《星际争霸》的职业电竞选手,标志着人工智能在部分信息博弈中已经可以战胜人类了.

1.2 深层神经网络

深度学习是“一类通过多层非线性变换对高复杂性数据建模算法的合集”,深层神经网络(DNN)是实现“多层非线性变换”的最常用的一种方式,两者互为代名词[3].DNN 的两个非常重要的特征是多层和非线性[30]:多层是为了符合分布式知识表达(第 1.1 节)的要求,非线性是为了解决更加复杂的问题.因为在现实世界中,绝大部分的问题都是无法线性分割的,而任何线性模型的组合仍然还是线性的.为 DNN 提供非线性表达能力的是激活函数(activation function).图 1 展示了一个神经元的输出是输入数据加权和与偏置加和之后经过激活函数非线性变换得到的结果.激活函数的特点是可微分并且单调.常用的激活函数有 Sign,Sigmoid,Tanh,ReLU,P-ReLU,Leaky-ReLU,ELU,Maxout 等.损失函数(loss function)是用于度量 DNN 输出结果向量与样本期望向量之间差距的函数.常用的损失函数有交叉熵(cross entropy)、均方差(mean square error,简称 MSE)、Log、L1 Loss、L2 Loss、Elastic Net 等.构造一个深层神经网络就是确定网络的 3 个组成部分:DNN 的架构(或称为拓扑结构)、激活函数与损失函数、训练 DNN 的算法.DNN 的使用一般分为训练和推理两个阶段.训练 DNN 即为网络中的神经元找到最优权值配置,主流的训练算法是梯度下降算法[1]和反向传播算法(第 1.1 节).训练得到的网络也称为推理网络(inference network),可以用于对测试数据集或实际数据的推理.

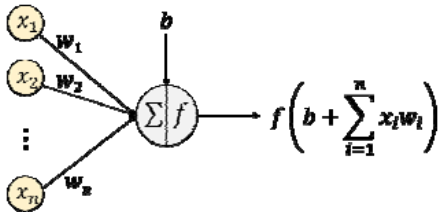

Fig.1 Neuron with activation function

图 1 含有激活函数的神经元

神经网络主要分为两种类型[1]:(1) 前馈神经网络(feedforward neural networks),即有向无环图,信号只能沿着最终输出的那个方向传播;(2) 反馈神经网络(feedback neural networks),即网络中有回路,输出层的信号经过一步时移(shift)再接入到输入层.常用的 DNN 模型如卷积神经网络(CNN)属于前馈神经网络,递归神经网络(RNN)和长短期记忆模型(LSTM)都属于反馈神经网络.CNN 可以有效地降低传统前馈神经网络的复杂性,并广泛应用于图像分类和物体识别等场景.CNN 网络的架构可以用公式[3]来表示:输入层→(卷积层+→池化层?)+→全连接层+,其中,“卷积层+”表示一层或多层卷积层(CONV),“池化层?”表示没有或一层池化层(POOL).卷积层的核心是卷积核,尺寸一般为 3×3,5×5,7×7.相比全连接方式,卷积核的参数非常少,各层通过卷积核共享机制可以极大减少训练阶段需要优化的总参数量.池化层可以非常有效地缩减矩阵的尺寸(主要用于减小矩阵的长和宽),从而减少最后全连接层中的参数,并有防止过拟合的作用.常用的池化策略有最大池化和平均池化等.RNN 引入了“记忆”的概念,主要用途是处理输出内容和历史内容有关联的场景,比如时间序列分析、语音识别、机器翻译、自然语言处理等.RNN 实现递归的结构非常简单,一般是若干个激活函数层叠加在一起组成隐藏层(DNN 可以划分成输入层、隐藏层、输出层.隐藏层就是输入层和输出层之间的多层神经元结构),循环元素在隐藏层上执行相同的任务,但是输出依赖于当前的输入和历史记忆.RNN 的推理过程可以用两个公式[1]表示:$S_t=f(UX_t+WS_{t-1})$,$O_t=Softmax(VS_t)$.其中:S_t 代表时刻 t 的隐藏层状态;O_t 代表时刻 t 的输出;X_t 是时刻 t 的输入;f 是激活函数;U 表示输入层到隐藏层的权重;V 表示隐藏层到输出层的权重;W 表示隐藏层到隐藏层的权重,负责 RNN 的记忆调

度.LSTM 是一种特殊的 RNN,用来解决 RNN 存在的长期依赖问题,即相关信息与预测位置的间隔很大而导致 RNN 无法学习连接信息.LSTM 的隐藏层比传统 RNN 复杂,是一种拥有 3 种“门”的特殊网络结构,从而更有效地保存长期记忆.其中:“遗忘门”的作用是让循环神经网络“忘记”之前没有用的信息,“输入门”决定哪些部分记忆进入当前时刻的状态,“输出门”决定当前时刻的输出.

当前主流的 DNN 开发及运行框架包括 TensorFlow(Google)[31],PyTorch(Facebook)[32],Caffe(Berkeley 大学)[33],其他 DNN 框架如 Theano(Montreal 大学),Keras(Keras-Team),MXNet(Amazon),CNTK(Microsoft)的用户基础远比不上前 3 种.DNN 框架的运行原理(以 TensorFlow 为例[34]):首先,将用户基于应用层 API 编写的、以神经网络算法为代表训练或推理过程表达为数据流图计算;在运行时,把数据流图转换成 C++核心层的细粒度、抽象化运行时状态,进而在计算设备(CPU 或 GPU)上以一致而有效的方式调度执行.主流的 DNN 框架能够支持在个人电脑、大型数据中心的服务器、嵌入式设备等多种平台上运行.

1.3 嵌入式GPU

图像处理器(graphics processing unit,简称 GPU)最初是计算机系统中加速图形图像运算的专用芯片,如今的 GPU 因集成了大量的处理器核心,具有了非常强大的并行处理能力,并且已经被广泛应用于通用计算领域,例如游戏开发、图像处理、视频处理、科学计算、大数据、深度学习等领域.当前,国际主流的 GPU 厂商主要有通用计算领域的 NVIDIA,AMD(ATI),Intel(CPU 内置显示核心)以及移动计算领域的 ARM,Qualcomm,PowerVR 等.本节选择最为广泛的 NVIDIA GPU 系列,先介绍普通 PC 平台 GPU,了解 GPU 的一般架构和工作原理.在此基础上,再介绍嵌入式 GPU 的架构和工作特点.

PC 平台的 GPU 计算卡一般通过高速的 PCI-E(peripheral communications interconnect express)总线与主板上的北桥芯片相连.PCI-E 是连接 GPU 卡与 CPU 的全双工高速总线.PCI-E 2.0 的传输速率为 5GB/s,PCI-E 3.0 提升到了 8GB/s.PCI-E 总线能够为每块 GPU 卡提供确定的读写带宽,而不受所挂载的 GPU 卡数量的影响.图 2 显示了一块 NVIDIA GPU 芯片的架构模块示意图,包括 3 种关键模块[35]:流处理器簇(streaming multiprocessor,简称 SM)、流处理器(streaming processor,简称 SP)、(全局、常量、共享)内存.“流”是一个 GPU 操作队列,该队列中的操作(可能由多个主机线程发出)将以添加到流中的先后顺序而依次执行.可以将一个流看作是 GPU 上的一个任务,不同流里的任务可以并行执行.所以一个流对应于并发的概念,多个流对应并行的概念.GPU 是由多个 SM 组成的 SM 阵列,每个 SM 包含 8N 个 SP(SP 也称 CUDA(compute unified device architecture)核,G80/GT200 中有 8 个 SP,RTX2080 中有 64 个 SP),每个 SP 都是一个设计简单的处理器.全局内存(global memory)就是 GPU 卡的显存.纹理内存(texture memory)和常量内存(constant memory)都是针对全局内存的一个特殊视图.其中,纹理内存是在显示 2D 或 3D 图像时存储插值计算所需要的数据,常量内存用于存储只读数据.每个 SM 通过总线可以独立访问 3 种内存.每个 SM 内部都有共享内存(shared memory).与 CPU 不同,它没有自动完成数据替换的硬件逻辑,而是完全由程序员控制,所以它是一种程序可控的高速缓存.

主流的 GPU 通用编程接口包括 NVIDIA 公司开发的 CUDA 运算平台、苹果公司持有但是保持开放的 OpenCL(open computing language)标准、微软公司开发的 Direct(direct compute)标准.CUDA[36]是 C 语言的一种扩展,支持基于 PTX(parallel thread execution)虚拟指令集的运行时编译,即 CUDA 代码先被编译成并行线程执行(parallel thread execution,简称 PTX)这种中间形式,然后再次编译为原生的 GPU 微码;并且向前兼容(forwards compatibility),即无论 GPU 的硬件结构如何改变,为早期 CUDA 设备编写的程序依然能够运行在最新的 CUDA 设备上.CUDA 编程模型是一种异构模型.CUDA 程序可以在 CPU 和 GPU 上并行运行,在 CPU 上运行的代码段叫 Host code,在 GPU 上运行的代码段叫 Device code,其中包含若干 kernel 函数.每一个 kernel 在 GPU 上执行时会启动很多线程运行同样的代码指令,即单指令多线程(single-instruction multiple-thread,简称 SIMT)的并行计算模型.Kernel 线程的数量取决于程序员为 kernel 函数指定的参数 num_blocks(线程块数)和 num_threads(每个线程块内执行 kernel 函数的线程数).这两个参数的配置会影响 kernel 函数的执行性能.每个 SM 最多能处理的线程数是有上界,也就间接影响每个 SM 最多能容纳的线程块(thread block)的数量.另外,每 32 个线程组成一个线程束(wrap),线程束是 GPU 调度和执行的基本单元.一个线程块所管理的线程束等于 num_threads 除以 32 并

向上取整,所以应该把 num_threads 设置成 32 的整数倍,否则会造成最后一个线程束中有部分线程被闲置.CUDA 主程序启动后,会将操作指令和数据提供给线程块,线程块以锁步(lock-step)的方式广播到每个线程束所占用的 SP 中;线程块一旦被调度到 GPU 的某个 SM,就必须从开始执行到结束.执行期间,线程束之间的数据交换也由 CUDA 主程序负责管理.

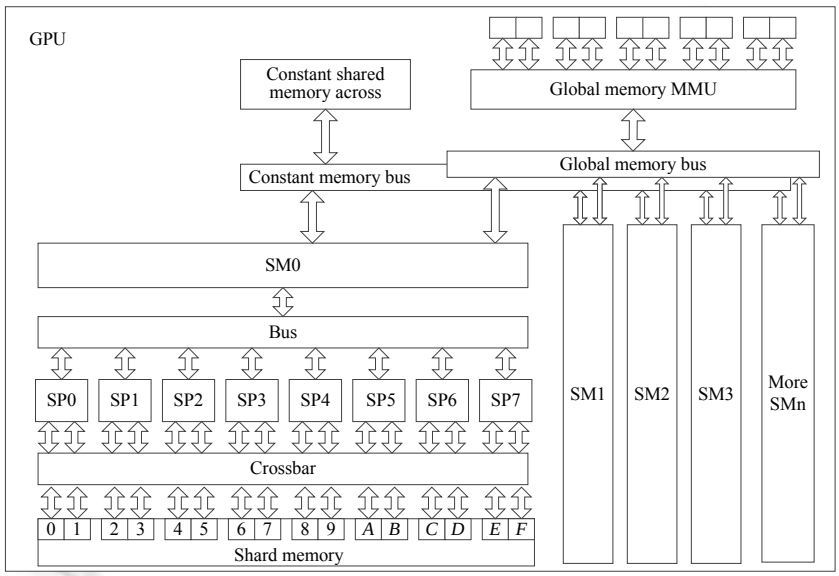

Fig.2 Block diagram of a GPU (G80/GT200) card

图 2 GPU(G80/GT200)卡的组成模块图

以上是 PC 平台 GPU 的基本结构、CUDA 程序的基本概念和基本运行规则.下面以 NVIDIA Jetson TX2[15,37] 为例,介绍嵌入式 CPU+GPU SoC 的结构特点.NVIDIA Jetson TX2 SoC 计算卡(图 3)集成了一枚 64 位四核 ARMv8@2.0 GHz 的 A57 微处理器、一枚双核超标量 ARMv8@2.0GHz 的 Denver 微处理器和一枚嵌入式的 Pascal 架构 GPU.两枚 CPU 各自拥有 2MB L2 cache.GPU 内包含执行引擎(execution engine,简称 EE)和拷贝引擎(copy engine,简称 CE),其中,EE 由两个流处理器簇(SM)构成,每个 SM 有 128 个 SP 核@1.3GHz,并共享 512 KB 的 L2 cache.所有的 CPU 与 GPU 共享 8GB DRAM@1.866GHz.

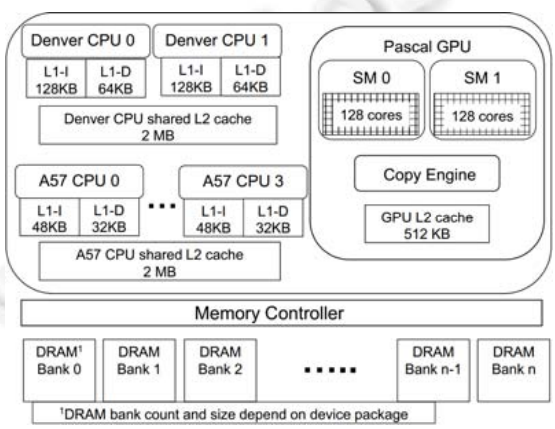

Fig.3 Block diagram of a NVIDIA Jetson TX2 card[15]

图 3 NVIDIA Jetson TX2 卡的组成模块图[15]

表 2 比较了 NVIDIA 的 RTX 2080 GPU(PC 平台)与 Jetson TX2 中嵌入式 GPU(SoC 平台)在计算资源、计算性能、显存、功耗等方面的差别.RTX 2080 拥有 46 个 SM 和 8GB 独立显存,每个 SM 包含 64 个 SP(即 CUDA 核)、8 个 Tensor 核用于深度学习加速、1 个 RT 核用于光线处理加速.相比而言,TX2 SoC 平台中 GPU 的计算资源要少得多,只有 256 个 SP 核,并且没有独立显存.由于 RTX 2080 的 SP 的数量比 Jetson TX2 高出一个数量级,所以其全精度计算性能也高出后者一个数量级,达到 10.1 TFLOPs.不过,TX2 中 GPU 的功率为 7.5~15 瓦,远小于 RTX 2080 GPU 的功率(215 瓦).

Table 2 Comparing RTX 2080 GPU and Jetson TX2 embedded GPU

表 2 比较 RTX 2080 GPU 与 Jetson TX2 嵌入式 GPU

	RTX 2080	JETSON TX2
架构	Turing 104	Pascal
SM	46	2
CUDA 核	2944(64×46)	256(128×2)
TENSOR 核	368(8×46)	无
RT 核	46(1×46)	无
计算性能(FP32)	10.1 TFLOPs	1.3 TFLOPs
显存容量	8GB 256-bit DDR6 独立显存	8GB 128-bit DDR4 共享显存
访存带宽	448 GB/s	59.7GB/s
功耗	215W	7.5W/15W
价格	¥5000	¥3500

1.4 神经网络加速器

近 10 年以来,深度学习的飞速发展得益于 CPU 和 GPU 算力的巨大进步.反过来,DNN 计算时的高负载特性,也促使 CPU 和 GPU 改进其体系结构以进一步提高计算效率.例如,Intel 最新的 Xeon Phi Knights Mill 处理器增加了对浮点运算可变精度的支持,NIVIDA 最新的 Volte 架构增加了专门的 Tensor Core 用于快速处理 DNN 任务中的被密集调用的矩阵乘与累加(multiply and accumulate,简称 MAC)操作.不过,这两种通用计算芯片所包含的部分功能模块(如多级缓存、分支处理、内存管理、线程调度等)在处理 DNN 任务时并不会被用到,反而占用芯片的面积,这就限制了在处理 DNN 任务时每单位芯片面积计算性能与能效比的提升[38].于是,人们研发了专用于处理深度学习任务的神经网络加速器,包括基于 DSP 或 FPGA 两种通用型芯片改进而来的加速器,还有 TPU(tensor processing unit),NPU(neural network processing unit)等采用为深度学习任务而定制体系架构的专用神经网络加速器.这里仅概述各加速器的主要特性和优缺点,详细信息可以参考文献[38].

DSP(digial signal processor)即数字信号处理芯片,具有强大的浮点运算能力.DSP 芯片通过在架构中加入神经网络处理核心成为了一种实现神经网络加速的可编程可扩展的有效平台.主流的 DSP 加速产品包括 EV6x(Synopsys),VIP8000(VeriSilicon)等,它们都支持当前主流的网络模型和框架.FPGA(field programmable gate array)即现场可编程门阵列芯片,能够实现快速的硬件功能验证和评估,从而加快设计的迭代速度.FPGA 凭借可编程阵列的特性,利用 VHDL 或 Verilog 语言可以设计新的硬件结构,更好地匹配神经网络的计算特点,比如针对 CNN 卷积层硬件实现各类优化的卷积方法[39,40].FPGA 在能耗方面更优于 GPU,但浮点运算性能逊于 GPU,因为目前的 DNN 计算还是重度依赖密集的浮点矩阵乘法(general matrix multiplication,简称 GEMM),而 GEMM 更利于映射到 GPU 上(常规并行性).

相比通用芯片,专用神经网络加速芯片(TPU,NPU 等)在处理 DNN 计算时能耗更低,性能更高(芯片的面积和功耗仅为 GPU 的百分之一量级).专用加速器片内主要包括针对 DNN 计算特点而特别设计的运算单元和存储单元,并且一般会设计专用的复杂指令集(complex instruction set,简称 CISC)来处理复杂的神经元操作,提升了速度,同时减少了功耗.运算单元有两种主流的结构:(1) 树状结构,包括 DianNao/DaDianNao/PuDianNao NPU(寒武纪)等;(2) 脉动阵列结构,包括 TPU(Google)、Scaledeep(普渡大学)、Eyeriss(MIT)、ShiDianNao(寒武纪)、昇腾 Atlas NPU(华为海思)等.存储单元用来存储神经网络每一层的输入值、权重和输出值等参数.随着神经网络规模变大,传统 2D 存储结构的 DDR 技术已经不能适应高带宽的要求,研究者已开始把最新的 3D 存储

技术引入到加速器设计中[41,42].同时,为了最小化数据搬移,缓解带宽瓶颈,运算器和存储器正朝着一体化的方向发展(忆阻器[43]).除此以外,专用神经网络加速器未来还将支持更多的神经网络框架,并采用非冯·诺依曼体系架构来大幅度提高整体运算性能、降低功耗.

2 DNN 任务在实时嵌入式系统中面临的挑战

嵌入式系统的共性特征是“实时性”,从应用的角度,实时性是指“不仅要保证运算的逻辑正确性,还必须保证运算在规定的时间内完成,否则将产生灾难性后果”.DNN 任务(主要是推理过程)在实时嵌入式系统上的成功部署与运行,既要在功能层面保证 DNN 推理结果的正确性和精确度,又要在非功能层面确保满足系统的实时性、资源和功耗的要求.本节将从 DNN 模型、深度学习框架以及硬件计算平台等方面总结和剖析 DNN 任务应用于实时嵌入式系统带来的问题和挑战.首先从 DNN 模型性能层面讨论在资源受限的实时嵌入式系统上部署复杂 DNN 任务所面临的问题,第 3 节针对这些问题从网络的性能分析、网络轻量化设计、实时神经网络方面做了详细的调研;然后讨论了计算平台(主要是 GPU、SoC、操作系统)保障 DNN 任务的实时性方面所面临的挑战,第 4 节针对这些问题从 GPU 时间分析、任务调度、资源管理策略方面做了详细的调研.

2.1 DNN模型带来的关键问题

(1) 传统 DNN 在实时嵌入式系统中具有局限性

当今,嵌入式信息物理融合系统正与深度神经网络领域互相融合,并且未来有朝着增强自主性方向发展的趋势.DNN 应用部署在高性能硬件平台(例如 GPU 集群)上,并且具有良好的可扩展性[19].如果把现有的网络模型部署在资源受限的实时嵌入式系统中,这将很难满足时序要求[21].其主要原因是受到 DNN 模型复杂度、计算和内存资源的巨大需求等约束,DNN 在低功耗的终端设备上运行,无法满足系统的实时性和可预测性.因此,在计算资源与时间方面,DNN 任务本身的需求与实际目标硬件平台的能力存在着较大的差异.在硬件层面,由于硬件的发展和更新速度明显落后于软件方面,仅仅通过提升硬件性能来解决该问题仍然具有局限性.在软件层面,探索轻量化网络的设计和优化也是一种有效的解决方案,即:在 DNN 网络的精度与时间、资源等方面进行权衡,综合考虑系统的功能层面要求与非功能层面要求.

(2) 通用深度学习框架的设计并未考虑嵌入式平台实时性要求

当前流行的深度学习框架有 TensorFlow,Caffe,CNTK,Theano,Torch 等,这些框架各具特色,基于这些深度学习框架可以较为容易地实现各种前沿的深度学习算法.深度学习框架的设计面向高性能计算,是为了提高任务吞吐量和系统可扩展性.然而,深度学习框架所实现的数据或任务并行(data or task parallelism)加速并未充分考虑到底层硬件,并未考虑嵌入式平台实时性要求[44](如资源分配时间有上界等).另一方面,不同的深度学习框架之间存在着性能差异,并且同一网络在不同框架上实现也存在着明显的性能差异,所以从深度学习框架角度优化网络运行时间,对于确保深度学习任务的实时性具有较大的提升空间.Kim 等学者[45]通过实验的手段分析了 AlexNet 在上述 5 种流行的框架上性能差异,只是通过实验的手段简单地揭示了不同框架之间存在性能差异的现象,但却没有从本质上揭露造成性能差异的原因.

(3) DNN 模型结构更新后不能保证系统的实时性

深度学习模型是基于设计的网络结构,通过大规模数据进行训练得到的网络参数,且满足样本分类和预测的精度.为了保证网络模型的预测精度和系统性能,网络模型往往需要优化,需要对扩充后的样本数据重新训练,更新后的网络模型会修改模型参数,甚至会更新网络结构.然而,当网络模型结构发生变化后,任务重新部署在资源受限的实时嵌入式系统上,为了满足系统实时性,其网络模型更新过程需要满足一系列的约束[46].系统更新时,既要保证新的网络模型不会干扰现有的系统,又要确保非功能的正确性.换句话说,计算平台要有足够的计算资源来运行新应用程序,而不会违反任何时间限制[47].对于实时嵌入式系统的有限资源下,任务模型更新后需要重新验证与分析系统的可信性.

2.2 硬件计算平台所面临的关键问题

实时系统要求在系统实际执行之前对时间行为进行分析,以确保在运行时系统的时间约束能够得到满足.一般通过任务级时间行为分析[48],即最坏执行时间(worst-case execution time,简称 WCET)分析,与系统级实时调度与调度分析[49]加以保证.WCET 分析的主要功能是分析程序的执行路径信息以及程序对硬件的访问行为,从而求得最坏执行时间.实时调度分析的目标是:在给定每个任务的 WCET 以及系统的实时调度算法的情况下,利用数学手段分析系统中的所有任务是否能在截止期之前完成.在 GPU 目标硬件平台上部署实时 DNN 任务需要解决以下 6 个主要问题:

(1) 针对 GPU 的 WCET 分析和实时调度分析尚不成熟

CPU 和 GPU 在体系结构上存在本质不同,导致 CPU 程序与 GPU 程序的执行特性、访存特性差别很大,这对程序的时间行为有巨大影响[50].现有的实时系统研究大多针对于 CPU,相关技术不能直接用于 GPU 程序的分析.现有的 GPU 性能分析与优化相关研究主要考虑系统的平均性能,而非实时性能(即最坏情况下的性能)与可调度性,因此也无法满足实时系统的要求.相对于 GPU 硬件的快速发展,面向 GPU 的实时系统时间分析与优化已严重滞后,这成为了 GPU 面向实时嵌入式系统应用的主要障碍.

(2) GPU 调度机制的细节信息不公开

文献[51]指出:当前主流的 GPU 生产厂商(例如 NVIDIA)出于商用机密的原因不公开 GPU 内部调度器的调度逻辑,客观上对实时 GPU 任务负载的可调度性分析造成了障碍.而且厂商有可能在没有任何通知的情况下,在新版本的 GPU 上修改调度逻辑.此外,由于 GPU 厂商的主要市场是高性能计算领域,提高任务的吞吐量和减小执行时间延迟是 GPU 调度器要追求的目标,这种调度策略必然不适合实时系统中任务的执行时间需要具有可预测性的要求.毫无疑问,嵌入式 GPU 调度策略的透明性给自动驾驶的安全性埋下了隐患.

(3) GPU 上执行混合关键任务会发生任务间干涉

GPU 上运行单个任务时,该任务独享 GPU 上的所有硬件资源,包括流处理器簇(SM)、cache、寄存器文件、内存、PCI-E 总线等.当 GPU 上执行混合关键任务时,由于需要共享资源,一个任务的执行时间将受到其他并行任务的干涉[52].干涉会增加系统执行时间的不确定性,如果设计不当,会造成系统整体执行性能的严重下降.同时,干涉会导致任务的状态空间随并行任务的增加呈现指数爆炸,这为任务级时间分析带来了巨大的挑战.还需要注意一点:一个 GPU 程序往往包含若干个 kernel 函数,在运行时会为每个 kernel 函数分配线程块(第 1.3 节).线程块是 GPU 资源分配的单位,线程束是 GPU 调度和执行的单位.这是比 GPU 任务更加细粒度的执行元素,增加了分析与调度的难度.

(4) GPU 不支持对混合关键任务的抢占

早期的 NIVDIA GPU(如 Tesla,Fermi,Kepler,Maxwell 架构)不提供抢占机制.Pascal 架构开始为图像任务和计算任务提供像素级(pixel-level)抢占和线程级(thread-level)抢占支持[53],但这是针对单 GPU 程序使用场景,目的是提高单任务的实时性.但是 GPU 调度器所提供的抢占粒度太小,不足以在任务级实现支持混合关键任务的动态优先级调度如 EDF(earliest-deadline first,最早时限优先)等.

(5) CPU+GPU SoC 平台上存在 CPU 和 GPU 之间协同的不确定性

在 NVIDIA Jetson TX2 嵌入式 SoC 平台中,CPU 和 GPU 共享了 8GB 的 DDR 内存(图 3).文献[54]指出,CPU 与 GPU 在访问共享内存可能会发生冲突造成 CPU 任务和 GPU 任务访存延迟的不确定性.由于 GPU 没有中断线路,所以响应 CPU 通知事件的唯一方式是通过 CUDA 的同步机制.文献[51]指出:CPU 与 GPU 之间的同步操作存在时间不确定性,同步操作完成的时间取决于在 GPU 上同时运行的其他任务,从而造成 GPU 任务的响应时间无确定上界.除了 CUDA 的显式同步操作(如 cudaDeviceSynchronize,cudaStreamSynchronize)完成时间的不确定性,一些共享内存访问的操作也会引发隐式同步行为,更增加了时间不确定性和分析的难度.

(6) GPU 工作温度过高会自动触发 GPU 降频运行

NVIDIA 开发者社区的一篇技术文章[55]指出:在不做任何设置的情况下,NVIDIA GPU 的工作频率会随着 GPU 的工作温度变化、能耗管理策略自动调整 GPU 工作频率,并呈现频繁的波动变化,导致同样的 GPU 程序

在不同时间段运行所测量的运行时间会有较显著的差异.通过 GPU 驱动所提供的接口可锁定 GPU 的运行频率为 GPU BIOS(basic input output system)中的预设频率,从而让 GPU 程序的运行时间保持恒定.但是当 GPU 温度超过最高工作温度阈值时,GPU 的硬件保护机制仍然会自动强制 GPU 降低工作频率.因此,当在 GPU 上运行实时任务时,需要在初始化时锁定 GPU 的工作频率.在散热能力有限的情况下,要特别注意 GPU 任务负载的调度,避免 GPU 长时间处于高负荷工作状态而产出高热,并引发 GPU 自动降频.GPU 的自动降频可能会导致实时系统调度严重失效,引发灾难性后果.

3 DNN 任务性能分析与优化

DNN 任务性能分析与优化的重点在于分析 DNN 任务的性能特性,基于系统瓶颈提出相关优化策略.现有研究工作旨在提升网络执行的平均性能.但遗憾的是,当前研究工作并没有把网络实时性纳入考虑范畴.在实时系统中仅考虑平均性能而不考虑实时性能,是无法满足安全攸关实时系统的要求的,这是实时系统中亟待解决的问题,也是未来研究的重点和难点.本节就 DNN 任务平均性能方面,首先总结前人在 DNN 任务性能分析方面的主要工作,综合阐述了 DNN 任务性能分析方法以及现有系统的瓶颈;然后,从轻量化网络角度阐述了 DNN 网络的性能优化工作;最后,在 DNN 任务实时性能方面,整理了当前实时神经网络的最新进展与成果.

3.1 DNN任务的性能分析

程序的性能分析(performance analysis)指的是通过静态/动态程序分析方法计算/收集程序运行时的性能数据,用于分析程序的负载特性和性能瓶颈,用户(程序开发者、编译器或者硬件计算平台)根据性能分析提供的反馈信息综合优化系统整体性能.GPU 作为主要的硬件加速设备,已经被广泛的应用于深度学习领域.DNN 任务在 GPU 上运行时会遇到各种性能障碍,例如低效的 DNN 算法、低精度的结果、并行的开销、负载的不均衡、内存缓存和带宽的低效使用、计算延迟超出响应时间要求等.为了识别 DNN 任务在 GPU 上执行时的负载特性和性能瓶颈,需要进行性能分析.目前,对基于 GPU 的 DNN 任务进行性能分析可以概括为 3 个发展阶段:

性能分析的第 1 阶段是借助性能分析工具,基于实验测试的方式获取 DNN 任务的性能数据.基于特定的 GPU 目标硬件平台,选取特定的深度学习框架或 DNN 算法,采用实验的手段,结合 NVIDIA 官方提供的性能分析工具(例如 nvprof,NVVP,Nsight)读取 GPU 上有限的硬件计数器获取程序运行时间、资源利用等性能数据.Kim 等学者[45]通过实验的手段分析了同一种 DNN 网络模型在 5 种流行的深度学习框架(Tensorflow,Caffe,Torch,CNTK,Theano)上的性能差异,他们基于 DNN 模型的每一层(layer)的前向/后向传播进行“端到端”的测量,获取每一层操作的执行时间和资源占用情况,识别每种框架下的 DNN 任务的“Hot Layer”(指计算量大、耗时长、消耗资源多的层).此外,Shams 与 Platania 等学者在文献[56]中,通过对不同的高性能计算硬件环境(NVLink,Knsight Landing)进行测量评估,比较了 Caffe,TensorFlow 与 Apache SINGA 在运行速度与可扩展性等方面的性能差异.虽然该研究工作揭示了不同深度学习框架或 DNN 算法之间存在着性能差异的客观事实,但是却没有揭示导致差异的本质原因.为探究 DNN 任务在特定 GPU 硬件平台上的工作性能,Mojumder 等学者基于 NVIDIA DGX-1 多 GPU 系统开展了一些研究工作[57].他们选取了 GoogleNet,AlexNet 等图像分类应用中 5 种典型的网络模型,在 DGX-1 系统中进行训练,对网络训练过程中的各个阶段的训练时间进行获取和分析,探究 DNN 模型在训练阶段的时间行为特性.该阶段的研究工作旨在通过基础的实验测量手段获取性能数据,用于识别 DNN 的负载特性和性能瓶颈,辅助程序开发人员或者 DNN 模型的设计人员设计出高效的 DNN 的训练和推理方法.但由于实验手段与性能工具的局限性,获取得到的性能数据和性能分析结果也存在着一定的局限性.

性能分析的第 2 阶段是采用程序插桩的方法获取细粒度的性能数据.基于 GPU 硬件平台,结合编译器和程序插桩方法,获取细粒度的性能数据,并基于分析结果进行 DNN 任务的优化工作.Stephenson 等学者在文献[58]中提出了一种基于后台编译的插桩工具 SASSI,该工具允许用户指定用于插桩的指令代码,自定义插桩位置和插桩内容,相对于其他插桩工具更加灵活.SASSI 内置于 NVIDIA GPU 的编译器中,对 PTX 代码进行插桩,实现对待检测程序的控制流行为、访存行为以及程序执行过程中的寄存器状态进行性能数据获取和分析.Du Shen 等学者提出了 CUDAAdvisor 性能分析框架[59],旨在对现代 GPU 上的代码优化提供指导作用.该工具利用

LLVM 框架对 CUDA 程序代码进行插桩,在程序的执行过程中收集性能数据,包括 CPU 与 GPU 两者之间的交互作用,基于性能数据进行访存行为分析、控制流分析以及代码调试.与其他工具不同的是,CUDAAdvisor 具有较好的扩展性,能够支持不同版本的 CUDA 与不同架构的 GPU.Farooqui 等学者[60]针对于 GPU 并行平台的性能检测和分析问题,提出并开发了动态插桩系统 Lynx,基于 GPU Ocelot 平台,使用 JIT 编译器对待检测程序在 PTX 代码级别进行转化、插桩与优化操作,具有较高的效率,能够获取 kernel 函数的执行时间、branch divergence 情况以及不同粒度(例如 kernel,thread block 等)的性能数据.虽然针对 GPU 的性能分析问题,插桩是一种有效的技术手段,可以在程序源码级别、中间代码级别以及目标代码级别实施插桩,通过插桩代码来检测程序运行时的性能数据,但是该方法会引入额外的负载,需要对引入的负载进行评估.

性能分析的第 3 阶段是结合 DNN 任务特性和 GPU 目标硬件平台特性构建性能模型,然后采用数学的方法分析和评估 DNN 任务的性能,识别性能瓶颈.Qi 等学者在文献[61]中提出了一种可分析的性能模型 PALEO,可以对简单的、具有特定声明规范的神经网络架构进行性能评估.PALEO 从指定的架构中提取需求并将其映射到软件、硬件和通信策略等设计空间中,将神经网络建模成一个有向图,利用数学方法计算网络的执行时间.由于 DNN 任务与 GPU 硬件两者的复杂性,构建性能模型的方法存在较大的难度.目前,相关的研究工作还未取得突破性进展,是未来进一步的研究方向.此外,还有一些学者进行了其他方面的探索.Dong Shi 在文献[62]中探究了卷积神经网络的微结构对 DNN 任务在 GPU 上执行所造成的影响,他们使用微架构设计逐层分析 CNN 模型的性能影响,并在典型的 GPU 存储架构中描述访存行为特性以及硬件资源的利用情况,识别单个 GPU 的潜在性能瓶颈.Madougou 等学者在文献[63]中提出了一个 GPU 应用程序的性能分析工具,该工具综合采用了随机森林的统计方法、GPU 硬件计数器以及机器学习的方法,构建了一个用于性能预测的随机森林模型.

利用上述的分析方法或工具,可以对 GPU 上 DNN 模型的训练和推理过程进行性能分析,并得出以下结论.

(1) 不同的深度学习框架会导致较大的性能差异.在相同的硬件配置条件下,同一个 Alexnet 模型在主流深度学习框架下的训练时间存在着较大的差异:在 CNTK 中训练最快,在 Theano 中训练最慢.不同的深度学习框架实现机制与 CUDA 驱动的融合程度不同,所以其性能具有较大差异.因此在实际的应用中,针对指定的 DNN 模型,可以选择更加匹配的深度学习框架,提升模型训练和推理的性能;

(2) 卷积层运算涉及到大量的密集型计算,需要大量的计算资源,占据网络模型全部训练/推理时间的大部分.一个完整的 DNN 网络模型包含多个层,例如数据层、卷积层、池化层、激励层、全连接层,每一层都执行特定的操作.其中,卷积层对输入数据进行卷积运算,并映射到特征空间中,是 DNN 模型中“Hot Layer”,提高卷积层的性能,可以大幅度的提高系统性能;

(3) 在实际应用中,DNN 模型包含大量的神经元,具有数千万的训练参数,计算量大,存在两个潜在的性能瓶颈:计算瓶颈和通信瓶颈.一方面,由于 GPU 等硬件加速平台计算资源是有限的,DNN 模型训练以及推理过程中,大量的数据计算就容易在硬件平台上造成性能瓶颈;另一方面,随着 DNN 网络规模的不断增大,为了加快网络的训练速度,经常需要在分布式 CPU/GPU 集群中完成整个训练.在主流的基于数据并行的分布式深度学习中,各个计算单元并发训练一个 DNN 模型,各个单元的训练数据不同,每一次迭代结束后,各个计算单元需要同步 DNN 参数或梯度,更新参数服务器上的模型,之后再将最新模型推送到各个数据并行单元,用于下一轮计算.因此,数据并行中参数的交换容易成为整体性能瓶颈,如何解决这种通信瓶颈以提高性能,成为并行方法设计的研究重点.

3.2 轻量化神经网络

本节陈述的 DNN 网络层性能优化相关研究工作主要是提高网络的平均性能,而非实时性能.基于上述 DNN 性能分析结论,DNN 任务性能优化方向是减少模型中冗余参数,优化卷积核的计算结构.设计轻量化的神经网络方法包括人工手动优化 DNN 模型和自动化设计方式优化 DNN 模型:手动优化 DNN 模型包括深度学习模型压缩、设计轻量化网络、优化卷积计算方式和网络中卷积层池化层搭配的网络结构设计;自动化设计方式是指采用神经网络架构搜索方式,基于多目标优化约束自动化设计网络.

(1) 深度学习模型压缩技术

将现有神经网络模型部署在资源受限的实时嵌入式系统的主要挑战之一是同时满足严格的运行时限要求和受限的 SWaP(size,weight,and power)要求[64],该问题尚未得到有效解决.近年来,许多研究机构从软件层面研究复杂深度学习网络的精简问题,提出了一些轻量化的神经网络模型.DNN 轻量化技术是指采用模型的近似化技术保证网络输出精度不损失或损失在可接受范围内的情况下,使得网络结构和网络参数尽可能的精简,实现系统模型的轻量化[65].网络模型近似操作常采用的 3 种方式是:模型压缩、剪枝、矩阵分解.经过模型近似技术处理过的模型称为近似模型,该模型在许多实际环境中已被证明表现出足够的精度[66,67].

模型近似操作的目标是为了探索更高效的基础架构.已有的工作聚焦于剪枝、压缩以及低比特方法来表示基本的网络架构[68−71].Denil 等学者[72]证明了深度学习模型的参数存在明显的冗余,并且参数值可以通过预测得到.在 MNIST 数据集上测试,该方法在最好情况下可以预测出多层神经网络的 95%的参数,同时验证了网络模型参数的冗余性.Han 等学者[17]提出使用 10%的精度关键数据快速训练 DNN,可以实现网络 98%的精度.这个结论再次证明了 DNN 网络存在一定的冗余性,说明了 DNN 模型压缩技术具有一定的可行性.Han 等学者[73,74]尝试将训练好的模型通过剪枝和权重共享增加权重稀疏性、降低存储量的方法,使得神经网络的存储需求减少 35 倍~49 倍却不损失网络的准确性.Jaderberg 等学者[69]尝试使用降秩技术来加速卷积神经网络,并且在场景字符分类数据集上训练的 4 层 CNN 精度下降不到 1%,却获得了 4.5 倍的加速.Zhang 等学者[70]提出了一种多层逼近时减小累积误差的算法,将非线性响应的重构误差降至最小,并采用低秩约束来降低了滤波器的复杂度. ImageNet 数据集上的标准测试结果表明:通过该算法压缩 SPPNet 得到的加速模型,比原 SPPNet 的推理速度快了 4 倍.虽然加速模型的 top-5 误差率比原模型增加了 0.9%,但是其推理精度相比 AlexNet 还是提高了 4.7%.

在 DNN 网络参数的线性量化方面,二值化神经网络(binary neural network,简称 BNN)[75]具有高模型压缩率和极快计算速度的优点,近几年格外受到重视,成为深度学习的热门研究方向.二值化网络方法通过将单精度浮点型权重矩阵二值化,其中一个权重值只用一个比特来表示.二值化方法根据权重值和激活函数值的符号决定在{+1,–1}取值:数值大于等于 0 则取+1,否则取为–1.对于原来 32 位浮点型数的乘加运算,二值化之后的权重值和激活函数值可以通过一次异或运算(xnor)和一次 POPCNT 位运算指令实现.二值化方法在网络模型的内存消耗理论上能减少 32 倍,可以看出,二值化神经网络在模型压缩上具有很大的优势.研究二值化神经网络对解决当前浮点型神经网络应用到实时嵌入式系统中存在的模型过大、计算密度过高等问题,具有很重大的意义.

(2) 神经网络轻量化设计

然而,上述在已训练模型上进行因式分解和稀疏卷积技术没有从根本上实现网络的高效轻量化.研究者们从网络结构设计出发,设计出了适用于移动设备的轻量级神经网络.表 3 中,最早于 2016 年 2 月,加州大学伯克利分校和斯坦福大学提出轻量化网络 SqueezeNet[65]模型.该模型具有新的网络结构,通过降低大型卷积核的输入通道数量,新的网络结构实现了不同大小卷积核的通道连接来特征提取.同年 10 月,Google 提出了 Xception 模型.2017 年 4 月,Google 提出的 MobileNet[76]模型具有一种新颖的卷积结构,其特点是在保证特征的非线性表示情况下,深度分离卷积式的卷积结构,充分解耦了传统卷积模型,将卷积分为深度卷积和逐点卷积.这种卷积改进方式可以极大降低计算量和参数量,并且适配于移动设备.2018 年,Google 和 Face++分别相继发布 MobileNetV2 和 ShuffleNetV2,两个轻量化网络均是在之前模型的改进模型.轻量化网络极大地降低了深度学习模型的参数量和计算量,同时,在很大程度上保证了特征的高度抽象提取,在一定程度上保证了模型的精确度.

Table 3 Lightweight network development timeline

表 3 轻量化网络发展时间轴

时间	提出单位	轻量化模型
2016.02	UC Berkeley & Stanford University	SqueezeNet[65]
2016.10	Google	Xception[77]
2017.04	Google	MobileNet[76]
2017.07	Face++	ShuffleNet[78]
2018.01	Google	MobileNet V2[79]
2018.07	Face++	ShuffleNet V2[80]

实现轻量化网络的技术方法主要包括:利用卷积核分解方法使用 1×N 网络和 N×1 网络代替 N×N 卷积核;使用深度压缩(deep compression)方法,包括网络剪枝、量化、哈弗曼编码、奇异值分解、硬件加速器等方法.因此,以 SqueezeNet 为例,其设计使用以下 3 个策略来减少参数:

(a) 使用 1×1 卷积代替 3×3 卷积,该设计使得计算参数减少为原来的 1/9;

(b) 减少输入通道数量,该部分使用 squeeze 层来实现;

(c) 在网络中,将欠采样(downsample)操作延后,可以给卷积层提供更大的激活图(activation maps).因为更大的激活图保留了更多的信息,可以提供更高的分类准确率.

其中,策略(a)和策略(b)可以显著减少参数数量,策略(c)可以在参数数量受限的情况下提高准确率.轻量化网络与传统 DNN 网络相比具有诸多优点[65],网络模型的精简也有助于整个系统的优化.比如:在分布式训练中,轻量化网络与服务器通讯需求较小,由于网络模型的参数少,从云端下载模型也更快.

当然,在使用轻量化技术缩短网络推理时间,提升系统性能的同时,人们还必须保证网络的预测精度足够高.优秀的代表性网络包括 SqueezeNet 和 SqueezeNext 等.SqueezeNet 模型和 Alexnet 模型在 ImageNet 数据集上的推理精度相当,但是前者比后者的参数数量减少了 50 个,并且 SqueezeNet 模型的尺寸小于 0.5MB.随后,Gholami 等学者总结了已有轻量化网络的结构优点,并根据神经网络加速器上的仿真结果作为指导,提出新的神经网络体系结构 SqueezeNext[81].它采用滤波器降秩(low rank filters)的方法进一步减少权值参数,并且采用瓶颈模块(bottleneck module)减少全连接层参数.有别于上述的单从软件层对网络模型进行优化,SqueezeNext 是从网络设计和硬件实现两个角度综合优化网络,其思想很值得借鉴.与 MobileNet 相比,SqueezeNext 在获得相似的 top-5 的分类精度下,其参数减少了 1.3 倍.

(3) 自动机器学习

为嵌入式系统设计深度学习网络,是一项具有挑战性的工作,因为嵌入式平台要求网络模型不仅体积小、运行速度快,还要确保网络预测的准确率.尽管研究者已经做了许多轻量化模型的设计和改进工作,例如上述 SqueezeNet,MobileNet 系列,但是手动设计高效模型仍然是一项挑战,因为网络设计要考虑的因素太多.AutoML (automated machine learning)[82]和神经架构搜索(neural architecture search,简称 NAS)[83]的发展,促进了深度学习模型的自动化设计.AutoML 是模型选择、特征抽取和超参数调优等一系列自动化方法,可以实现自动训练有价值的模型.机器学习最耗费人力的部分主要是数据清洗和模型调参,而这部分过程如果采用自动化方式实现,将会加快网络模型的开发过程.NAS 实现了用神经网络设计神经网络,代表了机器学习发展的未来方向.NAS 是 AutoML 的子领域,在超参数优化和元学习(meta learning)等领域高度重叠.与以往自动化搜索方法不同,MnasNet[84]的设计综合权衡了网络输出精度和实时性的多目标优化问题.它是 Google 提出探索使用架构搜索和强化学习设计模型的一种方法,并且在模型准确率和运算速率上均有突破.MnasNet 模型的运行速度比 MobileNet V2[79]快 1.5 倍、比 NASNet[83]快 2.4 倍,同时达到同样的 ImageNet top-1 的准确率.

3.3 实时神经网络

对于包含了人工智能应用的实时嵌入式系统,实时系统的设计除了考虑网络推理任务的时间可预测性以及硬实时性,还要考虑网络模型的输出精度.如果能够对深度学习网络的性能进行建模,就能分析出推理网络的执行时间,得出任务执行时间的上界,最终设计出调度策略来保证实时任务在截止期之前正确执行完.深度学习网络的输出结果本质上是基于一定概率的输出,因此,如何权衡网络的预测精度与网络的执行时间,即在牺牲可接受的网络精度来换取网络推理过程的实时性,引起了实时领域研究者的关注.Bateni 等学者[19]提出近似意识的实时神经网络 ApNet,对 AlexNet 模型以 layer 为单位应用降秩(low rank)近似方法,实验数据显示,以损失 5%的精度,换取网络层任务执行时间减少 50%~80%.时间可预测运行时系统 ApNet 通过逐层设计有效的近似压缩,确保了 DNN 任务的硬截止期.ApNet 是建立在多层 DNN 端到端理论分析的调度框架,基于每一层基础上有效近似技术来满足层任务的子截止期.基于权衡网络计算精度和运行时间的理论分析,在 NVIDIA Jetson TX2 平台上,设计并实现了一个运行时系统,提高了运行时多任务性能.

Zhou 等学者针对 GPU 加速的实时 DNN 任务提出监督流调度方案 S^3DNN.S^3DNN 综合考虑 DNN 任务的

实时性能、系统的吞吐量和资源利用率等问题[20].通过在数据采集处理的输入端进行并行设计,S^{3}DNN 提出将多传感器的数据进行融合,合成数据后,减少了 DNN 任务实例数量.同时还提出 GPU 的 kernel 调度与并行,通过将 kernel 切分为 3 部分,基于最小 slack 策略,设计 CUDA 流调度机制实现 kernel 并行执行.Yang 等学者从 CNN 实际智能驾驶应用提出了共享 CNN 网络的策略,将非关键级的摄像机 4 帧图像压缩合并为一张图片共享一个 Tiny YOLO 网络,相当于 4 路 CNN 网络并行;同时还提出对网络进行分段处理,在多段过程中增加任务流的并行性[16].该方案在既不增加 DNN 任务处理延迟又不降低网络识别精度情况下,还增加了网络吞吐量.虽然图像压缩后网络精度有一定损失,但对于非关键级别任务,其精度在可接受范围,并且经过再次训练后精度有所提升.

除此之外,在 DNN 的运行框架层的实时性优化方面,Hang 等学者[85]对 Caffe 框架进行了修改,增加了内存管理和垃圾回收机制,提升了处理速度.优化后的 Caffe 框架使得机器学习程序具有时间可预测性.面向实时应用的 DNN 网络方面的工作目前比较少,在人工智能全面应用的背景下,如何确保具有深度学习网络功能的实时系统的可靠性,是当下的研究热点和难点.深度学习网络应用的实时系统分析挑战一方面来自网络建模的复杂性,另一方面来自底层硬件加速平台的透明性.不同于只执行传统任务的实时系统,执行 DNN 任务的实时系统不仅要满足任务的实时性约束,还必须满足 DNN 网络模型预测精度的约束.总之,由于 DNN 模型的存在,该系统的实时性和可靠性将面临来自系统内部和外部环境的双重不确定性挑战.

4 GPU 时间分析与调度管理技术

随着 GPU 在嵌入式领域的应用愈加广泛,特别是深度学习(推理阶段)在 CPU+GPU SoC 上的应用,对 GPU 的时间分析与调度管理策略的研究已经越来越受到学术界的关注.但当前,GPU 软硬件设计的目标仍然是尽可能提高软件的并发度,最大可能榨取 GPU 的运算能力,但这并不能确保系统满足实时约束,例如,一个关键的实时任务可能频繁受到其他非关键任务执行的干扰,导致其时间行为不可预测.这些客观事实给 GPU 实时分析与调度的研究带来了挑战.学术界对 GPU 实时应用的研究主要包括:(1) 对 GPU 的 WCET 分析;(2) GPU 的调度策略;(3) 对 GPU 的计算和内存资源管理.

4.1 GPU WCET分析

在传统实时系统研究中,时间分析通常分为两步:首先分析每个程序的 WCET;然后把所有程序的 WCET 作为输入,结合实时调度算法,分析系统的可调度性(即每个程序是否能够在截止期前完成).所以,WCET 分析是可调度性分析的基础.起初,学术界对 GPU 上执行的 kernel 代码段的 WCET 分析也沿用了类似传统实时系统的研究思路:不考虑调度和数据传输的影响,并假设数据已经存在于 GPU 上,然后让 WCET 模型的分析结果尽量逼近 kernel 代码的实际执行时间.

Berezovskyi 等学者早在 2012 年之前就开始了 GPU 上的 WCET 分析,并提出了一个静态分析模型[86,87].该模型根据一个 SM 上的可分配 Wrap 数量、kernel 程序的线程数、kernel 程序指令数以及指令集的时钟周期表,在一些简化假设下,把 WCET 求解转化成一个整数线性规划(integer linear programming,简称 ILP)问题.该方法可以在几个小时内分析出一个 kernel 程序的 WCET,具有可接受的分析效率.但是该研究存在一些问题:首先,该方法假设所有的数据已经在 GPU 上,忽略了数据在系统中的搬运过程;其次,该方法只能分析一个 SM 上的一个 GPU kernel 程序,而显然,实际系统中一个 SM 上会运行多个 kernel,多个 SM 上的大量 kernel 也是并行运行的.

此后,Berezovskyi 等学者转向用基于测量的动态方法[88]分析 GPU 程序的 WCET.动态方法本质上要实际执行被分析的 kernel 程序,因此该方法能够考虑程序对各级存储结构的访问延时,并适用于有多 SM 的 GPU.实际上,在只有单个 kernel 任务执行的情况下,动态分析方法获得的 kernel 代码执行时间与静态分析出的 WCET 偏差并不大.原因是:其一,kernel 程序遵循 SIMT 模型,其各个线程的访存模式和代码执行模式是非常规律的;其二,GPU 调度器为每个 SM 预分配足够的资源以后,SM 在执行时是完全隔离的.但是,动态方法的问题仍然不够安全,无法用于硬实时系统.此外,动态方法难以深入剖析程序内部的行为特征,因此难以得到有效信息,指导程序性能优化.

Betts 与 Donaldson 提出一种混合分析技术[89]来分析 GPU kernel 程序的 WCET.该方法利用控制流程图

(control flow graph,简称 CFG)来建模 kernel 程序的行为,对 GPU 的硬件调度器分配线程块的行为进行建模,利用 kernel 的执行路径(execution traces)来帮助分析多个 wrap 之间的干涉,并估算干涉对执行时间产生的影响.干涉的开销会累积并沿着执行路径传导,最终获得 SM 上最后一个 wrap 的执行时间.遗憾的是,实验表明:这种干涉分析方法非常不精确,分析得到的 WCET 过于悲观.Punniyamurthy 等学者提出了基于混合模型模拟的 GPU kernel 运行时间分析技术[90],并基于开源 GPU 模拟器 GPGPU-Sim[91,92]开发了抽象时间仿真工具 GATSim.GATSim 使用功能模型(function model)与时间模型(time model)来快速模拟 GPU 程序在 SM 上的动态运行情况,并报告较准确的预测运行时间.其中,功能模型描述了 wrap 对应的指令序列块和每条指令的执行周期.时间模型描述 GPU 对负载的映射与调度、SM 上 wrap 的并行与干涉,从而实现对指令序列块动态运行状况的模拟.实验表明:GATSim 的平均预测准确率达 96%,平均运算速度高于 80MIPS(million instructions per second).这个速度远高于被学术界广泛使用的 GPGPU-Sim(2MIPS).

Zhang 等学者对 GPU WCET 分析的研究主要于 GPU 的片上存储器(L1/L2 cache 等),研究目标是提高 GPU 程序对片上存储器访问的时间和可预测性.文献[93]研究了把缓存锁定(cache locking)与缓存分区(cache partitioning)技术应用于 CPU/GPU 共享缓存管理.他们研究了 64KB~1024KB 不同大小末级缓存上锁定和分区技术下的程序性能,结果表明,这两种技术对于提高 GPU 程序性能的作用甚微.在文献[94]中,Zhang 等学者研究了对 GPU 片上 L2 缓存使用锁定技术的效果.实验结果表明:对于部分程序,L2 缓存锁定能够显著提高程序性能.实际上,这一研究结果隐含的另一层结论是:如果多个 wrap 的线程在 L2 缓存上大量冲突,可能严重降低执行性能.此外,Zhang 等学者研究了通过调整 wrap 线程对内存的读写操作顺序来提高程序的时间可预测性与总体性能[95,96].

当前,GPU WCET 分析框架所遇到的瓶颈与多核 CPU 时间分析遇到的问题[97]相似.当 GPU 上运行多个 kernel 程序时,一个程序的执行时间势必受到并发执行的其他程序的严重干扰.为了保证 WCET 分析的安全性,通常假设最坏的干扰情况发生,其结果是 WCET 分析过于悲观,造成系统运行资源的过量预留和严重浪费.而在 GPU 上,由于程序执行的并行度要远远高于多核 CPU 系统,上述问题会更加突出.如果能够获取并行 kernel 程序执行交叠信息,则可以大大提高干涉分析的准确性.但是问题的关键在于:并行 kernel 程序的执行交叠取决于任务调度,而 WCET 的结果又是任务调度的输入(即程序级和系统级时间分析存在耦合关系).因此,学术界亟待在 WCET 分析上突破原有的两级分析框架,把程序级 WCET 与可调度分析结合在一个分析框架中,从而提高时间分析的精确性.

4.2 GPU实时调度策略

GPU 实时调度需要解决的问题是调度不同优先级且相互竞争的作业完成数据移动任务和 GPU 计算任务,保证每个作业能在截止期之前完成.该方向的研究主要聚焦在为 GPU 设计实时调度算法和框架,并建立用于可调度性分析的数学模型.由于 GPU 的硬件调度器不提供抢占机制,所以早期研究的 GPU 调度算法都是固定优先级的.有限的优化措施是把大的 kernel 任务切分成小的任务,增加调度的灵活性.近几年,有学者提出了基于软件机制实现 kernel 任务抢占,从而有可能实现动态优先级调度.

GPU 实时调度领域较早期的工作是 Kato 等学者于 2011 年提出的 TimeGraph 调度器[98].TimeGraph 可被内嵌到开源的 GPU 驱动中,其所提供的 PRT(predictable response time)调度算法支持固定优先级调度,通过预分配给每个任务固定的执行预算以及监视每个任务已使用的执行预算来调整资源分配.TimeGraph 下,任务有可能运行超时,所以它本质上支持的是软实时调度.此后,Kato 等学者又提出了 RGEM[99],一个 GPGPU 实时调度器,支持固定优先级调度,主要贡献是在固定优先级调度分析中考虑了 DMA 操作的时间延迟.此外,RGEM 把 GPU 程序的执行分为了逻辑执行和数据拷贝两个阶段,这符合 GPU 程序的执行特点.Basaran 和 Kang 等学者研究了如何把一个运行时间较长且不可抢占的 GPU 程序进行切分,从而提高系统的可调度性[100].但是该工作中,GPU kernel 的切分需要用户参与而非自动化进行.此后,Zhong 和 He 等学者提出了 Kernelet 框架[101],该框架能够自动化分析 kernel 程序代码,但是代码优化是在运行阶段进行的,因此无法支持静态分析,也难以用于硬实时系统.

Verner 等学者研究了如何把 sporadic 任务模型下的多个作业(运行在 GPU 上)综合成一个大的作业,并让这

个大的作业按照一个四级流水线的方式工作,以约束访存行为[102104].所提出的调度算法能够用于硬实时系统.但是该工作的局限是只能够处理 GPU 程序,而无法分析 CPU+GPU SoC 系统计算的应用.CPU 与 GPU 在协同过程中会发生相互等待,Kim 等学者指出:如果存在任务自我等待(self-suspension)的行为,则传统的速率单调调度分析就不再成立[105],还研究了如何把大的任务切分成小的任务,从而降低由于任务相互等待造成的 GPU 时间浪费.但是,如何为这些切分的任务分配固定优先级仍是一个主要挑战,这个问题的复杂度是 NP-hard.

鉴于 NVIDIA 公司不公开 GPU 调度逻辑的细节,在文献[15]中,Smith 等学者通过在 NVIDIA Jetson TX2 上开展广泛深入的“黑盒实验”,揭示了该款嵌入式 GPU 调度器的 25 条隐含知识,并被归为 8 个大类:通用调度规则、非抢占执行条件、线程资源约束、GPU 片内共享内存约束、寄存器资源约束、拷贝操作条件、流(stream)规则、其他注意事项.其中,CUDA 流表示一个 GPU 操作队列,该队列中的操作(可能由多个主机线程发出)将以添加到流中的先后顺序而依次执行.可以将一个流看作是 GPU 上的一个任务,不同流里的任务可以并行执行.所以一个流对应于并发的概念,多个流对应并行的概念.这些在 NVIDIA 官方开发手册上找不到的调度细节,对 CUDA 程序的优化以及对 GPU 调度器的建模都非常有帮助.另一方面,GPU 厂商提供的硬件调度算法尽管能将计算任务尽可能快速地分配给 SM 处理器完成计算任务,但是并没有提供实时系统动态优先级调度算法所必须的抢占机制.由于硬件条件的限制,基于软件机制实现 GPU 的 kernel 抢占的有效调度策略较为困难,传统的方法是对 kernel 进一步进行划分,增加调度的灵活性.Chen 等学者在文献[106]中提出 EffiSha 抢占调度框架,使用了一种新颖的以 SM 为中心的转换方法,对 kernel 的形式进行转换,将 kernel 内部用于计算 ID 的 blockIdx.x 用 taskId 进行替换,解除了 block task 与 thread block 的绑定关系,实现了 block 级别的抢占调度.但是该框架目前只支持单个 stream 的情况,在扩展性和效率等方面有所欠缺.Wang 等学者在文献[107]中研究了一种动态的 thread block 发射策略,实现了对 thread block 的动态分配.他们对基础的 GPU 架构进行了扩展,增加了必要的部件和数据结构用于跟踪 GPU 上的 thread block 的分配和执行情况,从而根据当前的执行状态进行动态的调整和分配.该方法虽然实现了有效的动态资源分配和调度,但是需要改进 GPU 的基础架构,并且引入了较大的额外负载.

2017 年,随着 NVIDIA Pascal 架构 GPU 引入了像素级与线程级抢占,一些研究者就如何在 GPU 上实现可预测的实时动态优先级调度展开了研究.由于得到了 NVIDIA 公司的支持,Capodieci 等学者在 NVIDIA Tegra SoC 的驱动程序中实现了首个 EDF 实时调度算法[53].该调度策略作为 NVIDIA 虚拟机管理程序上层的软件分区运行,利用新一代 GPU 支持的线程级抢占特性,实现了 GPU 任务的 EDF 调度.不过,这项研究成果还未走出 NVIDIA 的实验室,所以市场上销售的 GPU 卡并不支持 GPU 任务的 EDF 调度.针对第 2.2 节问题(6)提出的 GPU 工作温度过高会影响任务的时间确定性问题,文献[108]提出了基于热量感知的 GPU 调度策略.其思路是:在 GPU 任务时间模型里加入热量模型作为约束,然后通过调度分析确定任务周期性占用/释放 GPU 的时间长度,理论上保证了在 GPU 散热系统工作正常的情况下,GPU 任务的实时性,并且 GPU 温度不会超过阀值.最后,通过真实 SoC 系统测试检验了该模型的有效性.

4.3 GPU资源管理策略

调度策略的实现离不开资源管理策略的配合.通过管理 kernel 任务访问和使用 GPU 的资源,能够很好地提高多任务并行系统的时间确定性.多核 CPU 系统研究经验也表明:如果能够更多地管控程序和系统的行为,可以大大降低分析层面的挑战[54].GPU 资源的管控的主要目的是协调并行程序对各类资源访问阶段,有效降低对资源的访问冲突,提高程序对硬件访问的时间可预测性,可以分为“空间隔离”和“时间隔离”:空间隔离机制针对的是系统内存、GPU 内存、GPU 片上的各级共享缓存,时间隔离机制针对的是程序的“访存阶段”和“运算阶段”.

在传统实时系统研究中,缓存层面的空间隔离技术包括缓存锁定、缓存旁路、缓存划分.目前,大多数 GPU 不支持缓存划分和缓存锁定,而 NVIDIA 公司的 CUDA 平台提供了在编译阶段旁路(bypass)L1 cache 和共享内存的能力,但是还不支持旁路 L2 cache.文献[109]提出一种分析 GPU 程序缓存复用距离(cache reuse distance,简称 CRD)的技术,如果一个程序的 CRD 大于某个阈值,则缓存对于该程序的加速效果将基本不存在,因此可以旁路掉.文献[110]提出了优化并行 kernel 任务的线程级并行化(thread-level parallelism,简称 TLP)来提高 GPU 资源利用率.kernel 任务对 GPU 资源的需求是多维度的(寄存器、共享内存、线程和线程块),如果放任所有的 kernel

任务运行尽可能多的线程,则必将导致严重的资源竞争冲突而降低 GPU 性能.更合理的做法是:根据 kernel 任务申请资源的互补性,调整各个任务分配的线程块数,从而最小化对资源竞争的冲突.该文还提出 kernel 级动态缓存旁路技术来调和并行 kernel 任务对 L1 cache 的竞争,即,只让部分 kernel 任务使用 L1 cache 并旁路掉其他 kernel 任务使用 L1 cache.不过,该技术需要 GPU 硬件中增加一个位向量,用于标记应该被旁路掉的 kernel 任务.最后,利用 GPGPU-Sim 仿真实验展示了两种调度优化技术能够带来平均 1.42 倍的 GPU 性能提升和 1.33 倍的能效比提升.Park 等学者在文献[111]中提出一种多任务 GPU 上的动态资源管理方法,GPU 是由多个 SM 组成,任务负载在运行的过程中,在每个 SM 上会有不同的分配情况.该方法先对任务的每次运行情况进行性能的监听和测量,根据不同设置产生的测试结果,选择最佳的资源划分方案.

美国北卡罗莱纳大学的 Elliott 与 Anderson 等学者围绕 GPU 实时调度开展了近 5 年的研究工作,提出了 GPUSync 这一实时调度框架[112].GPUSync 框架的主要创新是把 GPU 当作一个共享资源对待,则 GPU 上的实时调度问题转化为带有资源共享的实时调度问题.该研究团队在 GPU 实时调度领域还开展了大量实时层面的工作,相关的实时调度技术已经集成到了该团队开发的 LITMUS^{RT} 实时操作系统中.Anderson 团队的研究成果目前是 GPU 实时调度领域最领先的工作.2011 年,Pellizzoni 等学者提出了用于多核 CPU 访存隔离的可预测执行模型(predictable execution model,简称 PREM)[113,114].PREM 的思想是:把程序划分为冲突敏感的访存阶段(contension-sensitive memory phase)和无冲突的计算阶段(contension-free computation phase),并在任务调度时保证两个阶段不会发生重叠,从而避免了访存冲突.受 PREM 启发,瑞士苏黎世联邦理工学院的 Forsberg 等学者提出了 GPUguard 软件框架[54]来控制 SoC 平台中 CPU 与 GPU 的访存冲突.GPUguard 提供了 CPU/GPU 对共享内存的分时访问策略、内存带宽控制机制、访存与计算阶段的同步机制.如图 4 所示,GPUguard 为程序的访存阶段和运算阶段设置同步点(sync),并保障 CPU 访存阶段和 GPU 访存阶段在相邻同步分区中交替执行,从而消除了 SoC 上因 CPU/GPU 对共享内存的无序访问造成的冲突,提高了程序访存的时间确定性.Forsberg 的研究团队把 GPUguard 框架实现为一个 Linux 内核模块和一个 CUDA 应用模块,并在 NVIDIA Tegra TX1 SoC 平台上得到了良好的实验结果.Ali 等学者在文献[64]中提出了一种软件框架 BWLOCK++实现对 SoC 内存带宽的控制,确保实时任务能够不受非实时且访存密集型任务的干扰.该方法是一种软件机制,利用任务的访存情况与 GPU 的内存带宽,在 kernel 级别对任务进行访存带宽的限制,从而实现优先确保实时任务的执行.

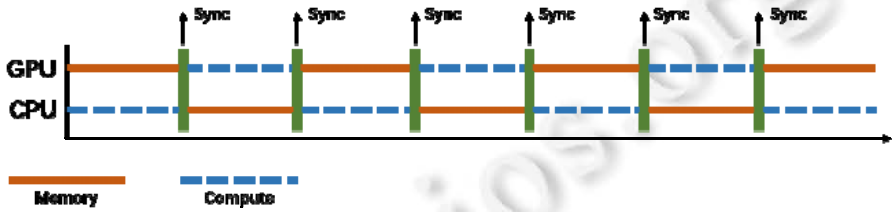

Fig.4 Execution flow of GPUguard

图 4 GPUguard 运行流图

5 DNN 与加速器的协同设计技术

深度学习与网络加速器以两个各自独立的阵营在快速发展,于是有学者指出:这种不匹配的独立发展的现状会导致模型的设计目标难免会有未充分考虑最新硬件的情况;反过来,网络加速器的设计也存在未充分考虑最新深度网络特征的情况[115,116].网络模型与网络加速器的协同设计(co-design),则能够充分发掘硬件潜力,提高网络的计算性能(速度、精度、能效比等),从而避免过度设计而导致的成本增加.协同设计可以从两个方面进行:(1) 以网络为固定标的,优化加速器的架构;(2) 以加速器为固定标的,优化网络的结构.

Kwon 等学者展示了如何从这两个方面进行 SqueezeNet 网络和加速器(基于硬件模拟器)之间协同设计[116].加速器的架构设计常用的执行流包括:权重固定流(weight stationary,简称 WS)和输出固定流(output stationary,简称 OS).针对方向(1),Kwon 通过实验 6 种 DNN 模型发现:WS 流和 OS 流对于不同大小的卷积操作,速度有明

显的不同(在 1×1 卷积中,WS 流比 OS 流快 1.4 倍~7 倍;在 Depthwise 卷积中,OS 流比 WS 流快 19 倍~96 倍).于是,为 SqueezeNet 定制了可变分层执行流架构的加速器 Squeezelerator,并获得了很好的加速效果(比单一 OS 流加速了 26%,比单一 WS 流加速了 106%).针对方向(2),Kwon 又以 Squeezelerator 加速器为基础,对 SqueezeNet 进行优化,得到 SqueezeNext[81,116]网络.优化方向包括:把第一个卷积层的卷积核从 7×7 缩小到 5×5,减少了推理时间;减少 SqueezeNet 前面阶段的层数并增加后面阶段的层数,因为前面阶段的硬件利用率低,而后面阶段硬件利用率高.SqueezeNext 获得了 2.59 倍的加速和 2.25 倍的能效比提升,并且比原 SqueezeNet 模型在图像分类基准测试上中的准确率高出 2%.

Yang 等学者也从两个方向上进行了协同设计[117].先从原 ShuffleNetV2 模型出发提出了优化模型 DiracDeltaNet,其采用了 4 项“激进”的优化策略:(1) 把所有的 3×3 卷积操作替换成移位(shift)运算[80]加 1×1 卷积操作;(2) 最大池化核从 3×3 缩小为 2×2;(3) 修改了通道交换(channel shuffle)的顺序;(4) 采用量化方法(quantization)把浮点型参数压缩成整型.这些模型层面的优化策略在硬件层面得到了来自协同设计的 FPGA 加速器 Synetgy 的支持:由 FPGA 负责执行“1×1 Conv-Pooling-Shift-Shuffle”操作.该协同设计方案在 ImageNet 图像分类测试中达到 88.2%的精度(top-5),推理速度达到了 96.5 FPS(frames per second),这刷新了所有 FPGA 分类器的最好成绩.Abdelouahab 等学者综述了通过 FPGA 加速 CNN 推理网的相关研究[115],分析了 CNN 计算负载、并行性和内存访问情况,阐述了卷积层和全连接层的优化问题、近似计算和数据路径优化方法.

Gao 等学者提出了通过优化数据流模型来实现可扩展的 NPU 加速器阵列设计[118].如果把 Eyeriss NPU 以瓦片架构(tiled architecture)加以连接,可以获得更大的硬件计算能力,从而可以计算更复杂的 DNN 任务.然而,随着 NPU 阵列的增大,数据冗余增大,数据移动和内存访问的开销也会随之增大.该研究团队通过使用两种技术——Buffer sharing dataflow 和 alternate layer loop ordering 优化了数据流模型的层内并发性(intra-layer parallelism)和层间流水线(inter-layer pipelining).基准测试实验显示该协同设计方案实现了 2 倍的加速比,并减少了 45%的能耗.

6 总结与展望

随着深度学习算法、嵌入式计算硬件、5G、物联网的不断发展,以深度学习为主要手段的人工智能技术必将在嵌入式应用领域得到更加广泛的应用,这其中也包括了安全攸关系统.因此,如何构建可信人工智能系统,已经成为了学术界和工业界的一个研究热点.本文对现有的面向实时应用的深度学习研究工作进行了综述,介绍了深度学习技术(主要是推理过程)应用于实时嵌入式系统所面临的挑战,并从深层神经网络的轻量化设计、GPU 时间分析与任务调度、CPU+GPU SoC 异构平台的资源管理、深层神经网络与网络加速器的协同设计等 4 个方面综述了近 5 年来的研究进展.虽然学术界在以上 4 个方取得了一定的成果,但仍然存在一些问题需要进一步研究和完善.下面将总结这些具体问题并展望进一步的研究方向.

DNN 轻量化设计问题:当前,对 DNN 的性能分析和优化以实验为主要手段,缺少基于形式化的 DNN 任务建模与分析框架.在 DNN 轻量化的研究方面,缺乏描述精度和实时性之间权衡关系的量化表达方法,也没有建立 DNN 轻量化技术的方法论.在 DNN 运行框架方面,现有的主流框架,如 TensorFlow, Caffe 等,尚无法满足实时系统对 DNN 任务运行时间确定性的要求.针对这些问题,需要进一步研究 DNN 任务性能分析模型、DNN 轻量化设计方法论、DNN 实时运行框架.其中,实时框架的开发涉及硬件、驱动、库、框架等多个层次,是个复杂的系统工程;

GPU 的实时分析问题:学者们大多延续了传统单核 CPU 实时系统的研究路线,无法应对 GPU 这种复杂的大规模并行器件,表现为两方面.

(1) 研究方法重分析轻设计,强调分析技术(WCET 分析、可调度性分析)在保障系统实时性中的作用.但近几年多核 CPU 分析遇到的困境表明:如果能够管控 GPU 程序的行为,提高时间的可预测性,则可以大大降低分析层面的复杂度;

(2) 现有的时间分析框架中,程序级分析和可调度性分析是独立的,没有考虑到 GPU 并行任务之间的干涉,

导致分析不精确.新的分析框架应该融合程序级分析与系统级分析,提高时间分析的精确性;

GPU 调度和资源管理问题:科技巨头 NVIDIA 公司不公开其 GPU 调度逻辑的详细资料,阻碍了对 NVIDIA GPU 开展实时调度研究和实验.虽然通过黑盒实验的方法可以获得很多不公开的调度规则,但是并不能确定这份调度规则清单是否足够完备,在新架构的 GPU 上是否依然有效.对 SoC 平台上 CPU 和 GPU 访问共享内存的分时隔离技术的研究已经取得了很大的进展,但是 CPU 与 GPU 之间显式或隐式的同步仍然会导致时间不确定性问题.由于 AMD 公司对其 GPU 技术细节的曝露要开放许多,并提供开源驱动 GPU Open[119],因此,一个可行的研究方向是以 AMD GPU+OpenCL[120]为平台来研究 GPU 实时调度[121]和资源管理技术,并研发用于实时 DNN 计算的基础软件.此外,前面综述过的调度或资源管理优化的研究工作存在技术路线不够系统化的问题,可以从 GPU 程序建模分析出发,结合系统的调度和资源分配,综合研究实时性能优化技术;

面向实时系统的网络加速器协同设计问题:无论是通用还是专用网络加速器仅能在一定程度上改善网络性能,并难以设计普适性的网络加速器结构.DNN 和网络加速器的协同设计可以提高两者的契合度,从而设计出性能特征高度匹配的网络与网络加速器整体解决方案,并降低硬件成本.不过,这个方向的研究主要集中在提高系统的平均性能,还未建立起满足实时系统要求的协同设计理论、性能建模与分析方法.考虑到神经网络加速器在未来必将广泛应用于安全攸关领域,面向实时应用的协同设计理论是一个非常有意义的研究方向;

智能实时嵌入式系统可更新问题:传统的实时嵌入式系统基于量体裁衣的方式设计程序,而很少考虑应用或系统更新之后可能会带来违背时间约束的问题(第 2.1 节中关键问题(3)).Wang 教授在文献[46]中指出了 CPS 安全攸关系统安全可更新问题和解决该问题的必要性、理论方向以及技术路线,同理,在 AI 赋能的实时嵌入式系统中,DNN 模型也会不断地更新迭代,那么如何保证模型更新之后的人工智能应用仍然能够满足原初设计的实时约束,将会是一个更具挑战性的理论问题,解决该问题无疑将大大促进人工智能实时嵌入式系统的发展.

References:

[1] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press, 2016.

[2] Molnar C. Interpretable machine learning: A guide for making black box models explainable. 2019. https://christophm.github.io/

[3] Zheng ZY, Gu SY. TensorFlow: A Practical Google Deep Learning Framework. Beijing: Publishing House of Electronic Industry, 2017 (in Chinese).

[4] Aceto L, Ingólfsdóttir A, Larsen KG, Srba J. Reactive Systems: Modelling, Specification and Verification. New York: Cambridge University Press, 2007.

[5] Seshia SA, Sadigh D. Towards verified artificial intelligence. CoRR, 2016, abs/1606.0.

[6] Sun X, Khedr H, Shoukry Y. Formal verification of neural network controlled autonomous systems. In: Proc. of the 22nd Int'l Conf. on Hybrid Systems: Computation and Control (HSCC). ACM, 2019. 147−156.

[7] Seshia SA, Desai A, Dreossi T, Fremont DJ, Ghosh S, Kim E, Shivakumar S, Vazquez-Chanlatte M, Yue X. Formal specification for deep neural networks. In: Proc. of the 16th Int'l Symp. on Automated Technology for Verification and Analysis (ATVA). Springer-Verlag, 2018. 20−34.

[8] Tuncali CE, Fainekos G, Ito H, Kapinski J. Simulation-Based adversarial test generation for autonomous vehicles with machine learning components. In: Proc. of the 2018 IEEE Intelligent Vehicles Symp. IEEE, 2018. 1555−1562.

[9] Dreossi T, Donzé A, Seshia SA. Compositional falsification of cyber-physical systems with machine learning components. In: Proc. of the 9th Int'l Symp. on NASA Formal Methods (NFM). Springer-Verlag, 2017: 357−372.

[10] Dreossi T, Ghosh S, Yue X, Keutzer K, Sangiovanni-Vincentelli AL, Seshia SA. Counterexample-Guided data augmentation. In: Proc. of the 27th Int'l Joint Conf. on Artificial Intelligence (IJCAI). 2018. 2071−2078.

[11] Pulina L, Tacchella A. An abstraction-refinement approach to verification of artificial neural networks. In: Proc. of the 22nd Int'l Conf. on Computer Aided Verification (CAV). Springer-Verlag, 2010. 243−257.

[12] Katz G, Barrett CW, Dill DL, Julian K, Kochenderfer MJ. Reluplex: An efficient smt solver for verifying deep neural networks. In: Proc. of the 29th Int'l Conf. on Computer Aided Verification (CAV). Springer-Verlag, 2017. 97−117.

[13] Singh G, Gehr T, Mirman M, Püschel M, Vechev MT. Fast and effective robustness certification. In: Proc. of the Advances in Neural Information Processing Systems 31: Annual Conf. on Neural Information Processing Systems (NeurIPS). Springer-Verlag, 2018. 10825−10836.

[14] PRECISE center for safe AI. https://precise.seas.upenn.edu/safe-autonomy

[15] Amert T, Otterness N, Yang M, Anderson JH, Smith FD. GPU scheduling on the nvidia tx2: Hidden details revealed. In: Proc. of the 2017 IEEE Real-Time Systems Symp. (RTSS). IEEE, 2017. 104−115.

[16] Yang M, Wang S, Bakita J, Vu T, Smith FD, Anderson JH, Frahm JM. Re-Thinking CNN frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge. In: Proc. of the 25th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS). IEEE, 2019. 305−317.

[17] Han R, Zhang F, Chen LY, Zhan J. Work-in-Progress: Maximizing model accuracy in real-time and iterative machine learning. In: Proc. of the Real-Time Systems Symp. 2018. 351−353.

[18] Alcaide S, Kosmidis L, Hernandez C, Abella J. High-Integrity GPU designs for critical real-time automotive systems. In: Proc. of the 2019 Design, Automation & Test in Europe Conf. & Exhibition (DATE). 2019. 824−829.

[19] Bateni S, Liu C. ApNet: Approximation-aware real-time neural network. In: Proc. of the Real-Time Systems Symp. (RTSS). IEEE, 2019. 67−79.

[20] Zhou H, Bateni S, Liu C. S^3DNN: Supervised streaming and scheduling for GPU-accelerated real-time dnn workloads. In: Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS). IEEE, 2018. 190−201.

[21] Yang M, Otterness N, Amert T, Bakita J, Anderson JH, Smith FD. Avoiding pitfalls when using nvidia GPUS for real-time tasks in autonomous systems. In: Proc. of the 30th Euromicro Conf. on Real-Time Systems (ECRTS). Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2018. 1−21.

[22] McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 1943,5(4):115−133.

[23] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. In: Proc. of the Psychological Review. 1958. 65−386.

[24] Minsky M, Papert S. Perceptrons—An Introduction to Computational Geometry. MIT Press, 1987.

[25] Sejnowski TT. The Deep Learning Revolution. MIT Press, 2018.

[26] Hinton GE. Learning distributed representations of concepts. In: Proc. of the 8th Annual Conf. of the Cognitive Science Society. Oxford University Press, 1986. 112.

[27] Learning representation by back-propagation errors. Nature, 1986,323:533−536.

[28] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, eds. Proc. of the Advances in Neural Information Processing Systems 25. Curran Associates, Inc., 2012. 1097−1105.

[29] Zhou Z. Machine Learning. Beijing: Tsinghua University Press, 2016 (in Chinese).

[30] Nielsen M. Neural Networks and Deep Learning. Determination Press, 2015.

[31] TensorFlow. https://www.tensorflow.org/

[32] PyTorch. https://pytorch.org/

[33] Caffe. http://caffe.berkeleyvision.org/

[34] Peng JT, Lin J, Bai XL. In-Depth Understanding of Tensorflow Architecture Design and Implementation Principles. Beijing: Posts & Telecom Press, 2018 (in Chinese).

[35] Cook S. CUDA programming: A Developer's Guide to Parallel Computing with GPUS. San Francisco: Morgan Kaufmann Publishers Inc., 2013.

[36] CUDA zone. https://developer.nvidia.com/cuda-zone

[37] Meet jetson, the platform for ai at the edge. https://developer.nvidia.com/embedded-computing

[38] Chen GL, Sheng M, Yang G. A survey of hardware-accelerated neural networks. Journal of Computer Research and Development, 2019,56(2):240−253 (in Chinese with English abstract).

[39] Farabet C, Martini B, Corda B, Akselrod P, Culurciello E, LeCun Y. NeuFlow: A runtime reconfigurable dataflow processor for vision. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, 2011. 109−116.

[40] Zhang C, Prasanna VK. Frequency domain acceleration of convolutional neural networks on CPU-fpga shared memory system. In: Proc. of the ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA). ACM, 2017. 35−44.

[41] Gao M, Pu J, Yang X, Horowitz M, Kozyrakis C. TETRIS: Scalable and efficient neural network acceleration with 3D memory. In: Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM, 2017. 751−764.

[42] Kim D, Kung J, Chai SM, Yalamanchili S, Mukhopadhyay S. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. In: Proc. of the 43rd ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). IEEE Computer Society, 2016. 380−392.

[43] Shafiee A, Nag A, Muralimanohar N, Balasubramonian R, Strachan JP, Hu M, Williams RS, Srikumar V. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: Proc. of the 43rd ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). IEEE Computer Society, 2016. 14−26.

[44] Xu H, Mueller F, Carolina N. Work-in-Progress: Making machine learning real-time predictable. In: Proc. of the 2018 IEEE Real-Time Systems Symp. (RTSS). IEEE, 2018. 157−160.

[45] Kim H, Nam H, Jung W, Lee J. Performance analysis of CNN frameworks for GPUS. In: Proc. of the IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS). IEEE, 2017. 55−64.

[46] Wang Y. Towards customizable CPS: Composability, efficiency and predictability. In: Duan Z, Ong L, eds. Proc. of the 19th Int'l Conf. on Formal Engineering Methods (ICFEM). Vol.10610. Xi'an: Springer-Verlag, 2017. 3−15.

[47] Abdullah J, Dai G, Yi W. Worst-Case cause-effect reaction latency in systems with non-blocking communication. In: Proc. of the 2019 Design, Automation & Test in Europe Conf. & Exhibition (DATE). 2019. 1625−1630.

[48] Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley DB, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner PP, Staschulat J, Stenström P. The worst-case execution-time problem—Overview of methods and survey of tools. ACM Transactions on Computer Systems, 2008,7(3):36:1−36:53.

[49] Davis RI, Burns A. A survey of hard real-time scheduling for multiprocessor systems. ACM Computing Surveys, 2011,43(4): 35:1−35:44.

[50] Hestness J, Keckler SW, Wood DA. A comparative analysis of microarchitecture effects on CPU and gpu memory system behavior. In: Proc. of the 2014 IEEE Int'l Symp. on Workload Characterization (IISWC). IEEE, 2014. 150−160.

[51] Posluszny D. Avoiding pitfalls when using nvidia GPUS for real-time tasks in autonomous systems. In: Proc. of the 30th Euromicro Conf. on Real-Time Systems (ECRTS). IEEE, 2018. 1−21.

[52] Reineke J, Wilhelm R. Impact of resource sharing on performance and performance prediction. In: Proc. of the Design, Automation & Test in Europe Conf. (DATE). European Design and Automation Association, 2014. 1−2.

[53] Capodieci N, Cavicchioli R, Bertogna M, Paramakuru A. Deadline-Based scheduling for GPU with preemption support. In: Proc. of the 2018 IEEE Real-Time Systems Symp. (RTSS). IEEE, 2018. 119−130.

[54] Forsberg B, Marongiu A, Benini L. GPUguard: Towards supporting a predictable execution model for heterogeneous SoC. In: Proc. of the 2017 Design, Automation and Test in Europe (DATE). 2017. 318−321.

[55] Bavoil L. SetStablePowerState.exe: Disabling GPU boost on windows 10 for more deterministic timestamp queries on nvidia GPUS. 2016. https://developer.nvidia.com

[56] Shams S, Platania R, Lee K, Park SJ. Evaluation of deep learning frameworks over different HPC architectures. In: Proc. of the Int'l Conf. on Distributed Computing Systems. IEEE, 2017. 1389−1396.

[57] Mojumder SA, Louis MS, Sun Y, Ziabari AK, Abellán JL, Kim J, Kaeli D, Joshi A. Profiling DNN workloads on a volta-based DGX-1 system. In: Proc. of the 2018 IEEE Int'l Symp. on Workload Characterization (IISWC). 2018. 122−133.

[58] Stephenson M, Sastry Hari SK, Lee Y, Ebrahimi E, Johnson DR, Nellans D, O'Connor M, Keckler SW. Flexible software profiling of GPU architectures. ACM SIGARCH Computer Architecture News, 2015,43(3):185−197.

[59] Shen D, Song SL, Li A, Liu X. CUDAAdvisor: LLVM-based runtime profiling for modern GPUS. 2018. 214−227.

[60] Farooqui N, Kerr A, Eisenhauer G, Schwan K, Yalamanchili S. Lynx: A dynamic instrumentation system for data-parallel applications on GPGPU architectures. In: Proc. of the IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS). IEEE, 2012. 58−67.

[61] Qi H, Sparks ER, Talwalkar A. Paleo: A performance model for deep neural networks. In: Proc. of the ICLR. 2017. 1−10.

[62] Dong S, Gong X, Sun Y, Baruah T, Kaeli D. Characterizing the microarchitectural implications of a convolutional neural network (CNN) execution on GPUS. 2018. 96−106.

[63] Madougou S, Varbanescu AL, De Laat C, Van Nieuwpoort R. A tool for bottleneck analysis and performance prediction for GPU-accelerated applications. In: Proc. of the 2016 IEEE 30th Int'l Parallel and Distributed Processing Symp. (IPDPS). IEEE, 2016. 641−652.

[64] Ali W, Yun H. Protecting real-time GPU kernels on integrated CPU-GPU SoC platforms. In: Proc. of the 30th Euromicro Conf. on Real-Time Systems (ECRTS). Vol.106. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2018. 19:1−19:22.

[65] Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5MB model size. 2016. 1−13.

[66] Wang Y, Li H, Li X. Real-Time meets approximate computing: An elastic CNN inference accelerator with adaptive trade-off between qos and qor. In: Proc. of the 54th Annual Design Automation Conf. 2017. 2017. 33:1−33:6.

[67] Sainath TN, Kingsbury B, Sindhwani V, Arisoy E, Ramabhadran B. Low-Rank matrix factorization for deep neural network training with high-dimensional output targets. In: Proc. of the IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2013. 6655−6659.

[68] Lebedev V, Ganin Y, Rakhuba M, Oseledets I, Lempitsky V. Speeding-Up convolutional neural networks using fine-tuned cp-decomposition. 2014. 1−11.

[69] Jaderberg M, Vedaldi A, Zisserman A. Speeding up convolutional neural networks with low rank expansions. 2014.

[70] Zhang X, Zou J, Ming X, He K, Sun J. Efficient and accurate approximations of nonlinear convolutional networks. In: Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition. 2015. 1984−1992.

[71] Zhang X, Zou J, He K, Sun J. Accelerating very deep convolutional networks for classification and detection. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2016,38(10):1943−1955.

[72] Denil M, Shakibi B, Dinh L, Ranzato M, De Freitas N. Predicting parameters in deep learning. 2013. 1−9.

[73] Han S, Mao H, Dally WJ. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. 2015. 1−14.

[74] Han S, Pool J, Tran J, Dally WJ. Learning both weights and connections for efficient neural networks. 2015. 1−9.

[75] Courbariaux M, Hubara I, Soudry D, El-Yaniv R, Bengio Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. 2016.

[76] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H. MobileNets: Efficient convolutional neural networks for mobile vision applications. 2017.

[77] Chollet F. Xception: Deep learning with depthwise separable convolutions. In: Proc. of the 30th IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). 2017. 1800−1807.

[78] Zhang X, Zhou X, Lin M, Sun J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In: Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition. 2018. 6848−6856.

[79] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. MobileNetV2: Inverted residuals and linear bottlenecks. In: Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition. 2018. 4510−4520.

[80] Ma N, Zhang X, Zheng HT, Sun J. Shufflenet v2: Practical guidelines for efficient CNN architecture design. In: Proc. of the 15th European Conf. on Computer Vision (ECCV). LNCS 11218. Springer-Verlag, 2018. 122−138.

[81] Gholami A, Kwon K, Wu B, Tai Z, Yue X, Jin P, Zhao S, Keutzer K, Berkeley UC. SqueezeNext: Hardware-aware neural network design. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition Workshops (CVPR). IEEE Computer Society, 2018.

[82] Yao Q, Wang M, Chen Y, Dai W, Yi QH, Yu FL, Wei WT, Qiang Y, Yang Y. Taking human out of learning applications: A survey on automated machine learning. 2018. 1−26.

[83] Zoph B, Le Q V. Neural architecture search with reinforcement learning. 2016. 1−16.

[84] Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV. MnasNet: Platform-aware neural architecture search for mobile. 2018.

[85] Xu H, Mueller F. Hardware for machine learning: Challenges and opportunities. In: Proc. of the Real-Time Systems Symp. IEEE, 2019. 157−160.

[86] Berezovskyi K, Bletsas K, Andersson B. Makespan computation for GPU threads running on a single streaming multiprocessor. In: Proc. of the 24th Euromicro Conf. on Real-Time Systems (ECRTS). IEEE, 2012. 277−286.

[87] Berezovskyi K, Bletsas K, Petters SM. Faster makespan estimation for GPU threads on a single streaming multiprocessor. In: Proc. of the 2013 IEEE 18th Conf. on Emerging Technologies & Factory Automation (ETFA). IEEE, 2013. 1−8.

[88] Berezovskyi K, Santinelli L, Bletsas K, Tovar E. WCET measurement-based and extreme value theory characterisation of CUDA kernels. In: Proc. of the 22nd Int'l Conf. on Real-Time Networks and Systems (RTNS). ACM, 2014. 279.

[89] Betts A, Donaldson A. Estimating the wcet of GPU-accelerated applications using hybrid analysis. In: Proc. of the Euromicro Conf. on Real-Time Systems. IEEE, 2013. 193−202.

[90] Punniyamurthy K, Boroujerdian B, Gerstlauer A. GATSim: Abstract timing simulation of GPUS. In: Proc. of the Design, Automation & Test in Europe Conf. (DATE). IEEE, 2017. 43−48.

[91] GPGPU-Sim. http://www.gpgpu-sim.org/

[92] Bakhoda A, Yuan GL, Fung WWL, Wong H, Aamodt TM. Analyzing CUDA workloads using a detailed gpu simulator. In: Proc. of the IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS). IEEE, 2009. 163−174.

[93] Wang X, Zhang W. Cache locking vs. partitioning for real-time computing on integrated CPU-GPU processors. In: Proc. of the 35th IEEE Int'l Performance Computing and Communications Conf. (IPCCC). IEEE, 2016. 1−8.

[94] Picchi J, Zhang W. Impact of l2 cache locking on GPU performance. In: Proc. of the SoutheastCon 2015. IEEE, 2015. 1−4.

[95] Huangfu Y, Zhang W. Warp-Based load/store reordering to improve GPU data cache time predictability and performance. In: Proc. of the 19th IEEE Int'l Symp. on Real-Time Distributed Computing (ISORC). IEEE, 2016. 166−173.

[96] Huangfu Y, Zhang W. Warp-Based load/store reordering to improve gpu time predictability. JCSE, 2017,11(2).

[97] Chen G, Guan N, Lü MS, Wang Y. State-of-the-Art survey of real-time multicore system. Ruan Jian Xue Bao/ Journal of Software, 2018,29(7):2152−2176 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5580.htm [doi: 10.13328/j.cnki.jos.005580]

[98] Kato S, Lakshmanan K, Rajkumar R, Ishikawa Y. TimeGraph: GPU scheduling for real-time multi-tasking environments. In: Proc. of the 2011 USENIX Conf. on USENIX Annual Technical Conf. ACM, 2011. 2.

[99] Kato S, Lakshmanan K, Kumar A, Kelkar M, Ishikawa Y, Rajkumar R. RGEM: A responsive GPGPU execution model for runtime engines. In: Proc. of the 32nd Real-Time Systems Symp. (RTSS). IEEE, 2011. 57−66.

[100] Basaran C, Kang KD. Supporting preemptive task executions and memory copies in GPGPUS. In: Proc. of the 24th Euromicro Conf. on Real-Time Systems (ECRTS). IEEE, 2012. 287−296.

[101] Zhong J, He B. Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling. IEEE Trans. on Parallel Distrib. Syst., 2014,25(6):1522−1532.

[102] Verner U, Schuster A, Silberstein M, Mendelson A. Scheduling processing of real-time data streams on heterogeneous multi-GPU systems. In: Proc. of the 5th Annual Int'l Systems and Storage Conf. (SYSTOR). ACM, 2012.

[103] Verner U, Mendelson A, Schuster A. Batch method for efficient resource sharing in real-time multi-GPU systems. In: Proc. of the 15th Int'l Conf. on Distributed Computing and Networking (ICDCN). Springer-Verlag, 2014. 347−362.

[104] Verner U, Mendelson A, Schuster A. Scheduling periodic real-time communication in multi-GPU systems. In: Proc. of the 23rd Int'l Conf. on Computer Communication and Networks (ICCCN). IEEE, 2014. 1−8.

[105] Kim J, Andersson B, De Niz D, Rajkumar R. Segment-Fixed priority scheduling for self-suspending real-time tasks. In: Proc. of the 34th Real-Time Systems Symp. (RTSS). IEEE, 2013. 246−257.

[106] Chen G, Zhao Y, Shen X, Zhou H. EffiSha: A software framework for enabling effficient preemptive scheduling of GPU. In: Proc. of the PPoPP. 2017. 3−16.

[107] Wang J, Rubin N, Sidelnik A, Yalamanchili S. Dynamic thread block launch: A lightweight execution mechanism to support irregular applications on GPUS. ACM SIGARCH Computer Architecture News, 2015,43(3):528−540.

[108] Hosseinimotlagh S, Kim H. Thermal-Aware servers for real-time tasks on multi-core GPU-integrated embedded systems. In: Proc. of the 25th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS). IEEE, 2019. 254−266.

[109] Nugteren C, Van den Braak GJ, Corporaal H, Bal HE. A detailed GPU cache model based on reuse distance theory. In: Proc. of the 20th Int'l Symp. on High Performance Computer Architecture (HPCA). IEEE, 2014. 37−48.

[110] Liang Y, Li X. Efficient kernel management on GPUS. ACM Transactions on Computer Systems, 2017,16(4):115:1−115:24.

[111] Park JJK, Park Y, Mahlke S. Dynamic resource management for efficient utilization of multitasking GPUS. ACM SIGARCH Computer Architecture News, 2017,45(1):527−540.

[112] Elliott GA, Ward BC, Anderson JH. GPUSync: A framework for real-time GPU management. In: Proc. of the Real-Time Systems Symp. 2013. 33−44.

[113] Pellizzoni R, Betti E, Bak S, Yao G, Criswell J, Caccamo M, Kegley R. A predictable execution model for cots-based embedded systems. In: Proc. of the 17th Real-Time and Embedded Technology and Applications Symp. (RTAS). IEEE, 2011. 269−279.

[114] Alhammad A, Pellizzoni R. Time-Predictable execution of multithreaded applications on multicore systems. In: Proc. of the Design, Automation & Test in Europe Conf. (DATE). European Design and Automation Association, 2014. 1−6.

[115] Abdelouahab K, Pelcat M, Serot J, Berry F. Accelerating CNN inference on fpgas: A survey. 2018.

[116] Kwon K, Amid A, Gholami A, Wu B, Asanovic K, Keutzer K. Co-Design of deep neural nets and neural net accelerators for embedded vision applications. 2018. 1−6.

[117] Yang Y, Huang Q, Wu B, Zhang T, Ma L, Gambardella G, Blott M, Lavagno L, Vissers K, Wawrzynek J, Keutzer K. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. 2018. 23–32.

[118] Gao M, Yang X, Pu J, Horowitz M, Kozyrakis C. TANGRAM: Optimized coarse-grained dataflow for scalable NN accelerators. In: Proc. of the 24th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2019. 807–820.

[119] GPUOpen. https://gpuopen.com/

[120] OpenCL. https://www.khronos.org/opencl/

[121] Luo Y, Li S, Sun K, Renteria R, Choi K. Implementation of deep learning neural network for real-time object recognition in opencl framework. In: Proc. of the Int'l SoC Design Conf. (ISOCC). IEEE, 2017. 298–299.

附中文参考文献:

[3] 郑泽宇,顾思宇.TensorFlow:实战 Google 深度学习框架.北京:电子工业出版社,2017.

[29] 周志华.Machine Learning:机器学习.北京:清华大学出版社,2016.

[34] 彭靖田,林健,白小龙.深入理解 TensorFlow 架构设计与实现原理.北京:人民邮电出版社,2018.

[38] 陈桂林,胜马,阳郭.硬件加速神经网络综述.计算机发展与研究,2019,56(2):240−253.

[97] 陈刚,关楠,吕鸣松,王义.实时多核嵌入式系统研究综述.软件学报,2018,29(7):2152−2176. http://www.jos.org.cn/1000-9825/5580.htm [doi: 10.13328/j.cnki.jos.005580]

张政馗(1984－),男,博士,讲师,主要研究领域为实时系统模型检测,实时系统设计与时间分析.

吕鸣松(1980－),男,博士,副教授,主要研究领域为实时系统时间分析,实时操作系统.

庞为光(1989－),男,硕士,主要研究领域为实时嵌入式系统,实时人工智能系统.

王义(1961－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为形式化方法,实时嵌入式系统.

谢文静(1994－),女,学士,主要研究领域为实时系统时间分析,GPU 性能分析.