面向细粒度 FPGA 管理的 CNN 异构加速框架

郭开诚[1)] 吴承刚[1)] 张伟丰[2)] 戚正伟[1)] 管海兵[1)]

[1)](上海交通大学电子信息与电气工程学院 上海 200240)

[2)](阿里巴巴集团 杭州 311121)

摘 要 近年来,现场可编程逻辑门阵列(FPGA)由于其灵活的可定制性和优秀的并行性,在硬件加速卷积神经网络(CNN)的研究和应用中吸引了广泛的关注.这些工作主要集中在两方面:对特定硬件加速模块的设计和优化以及对一类网络模型的通用加速硬件设计.前者一般是基于数据流的针对固定网络的设计,通过牺牲通用性来换取性能;后者一般是基于指令集能够加速一类模型的设计,通过牺牲性能来换取通用性.为了能够灵活地应对不同的需求,本文提出一种通过管理不同粒度算子来平衡性能与通用性的 fGrain 框架.该框架一方面利用底层基于数据流的算子设计来充分发挥硬件性能,另一方面通过虚拟化层来管理算子映射提供灵活性.实验表明,相比 GPU 推理延迟至多有 25%的提升,而虚拟化性能损失仅在 1.3%以下.

关键词 卷积神经网络;现场可编程逻辑门阵列;机器学习系统;用户态虚拟化;开放编程语言

中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2021.02529

A Heterogeneous Framework to Accelerate CNNs with Fine-Grained FPGA Management

GUO Kai-Cheng[1)] WU Cheng-Gang[1)] ZHANG Wei-Feng[2)] QI Zheng-Wei[1)] Guan Hai-Bing[1)]

[1)](*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai* 200240)

[2)](*Alibaba Group, Hangzhou* 311121)

Abstract In recent years, field-programmable logic gate arrays (FPGAs) have attracted much attention in hardware-accelerated convolutional neural network (CNN) research and applications due to their flexible customizability and excellent parallelism. In particular, FPGA-based accelerators have excellent performance in specific network structures and low-precision inference scenarios by customizing at the hardware level. These works have focused on two main areas: the design and optimization of specific hardware acceleration modules and the design of generic acceleration hardware for a class of network models. The former is generally a dataflow-based design for a fixed network, sacrificing generality for performance; the latter is generally an instruction set-based design capable of accelerating a class of models, sacrificing performance for generality. An accelerator under stream-based architecture accelerates CNN models by instantiating several processing units (PU). Usually, each PU only provides a specific functionality, such as computing a convolution layer. Compared with instruction-based architecture, stream-based architecture is more friendly to network-dependent optimizations. In an instruction-based architecture, the hardware accelerator comes with an instruction set. The software compiles the CNN model into a

收稿日期:2020-08-31;在线发布日期:2021-04-22.本课题得到国家自然科学基金(61672344,61525204,61732010)、国家重点研发计划(2016YFB1000502)、阿里巴巴创新研究计划(AIR)资助.**郭开诚**,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为机器学习系统、FPGA 虚拟化.E-mail:guokaicheng@sjtu.edu.cn.**吴承刚**,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为分布式系统、机器学习加速器.**张伟丰**,博士,阿里巴巴集团研究员,主要研究领域为异构计算、编译、高性能软硬协同优化.**戚正伟**,博士,教授,中国计算机学会(CCF)杰出会员,主要研究领域为系统软件、虚拟化、程序分析.**管海兵**,博士,教授,长江学者,国家杰出青年科学基金入选者,中国计算机学会(CCF)杰出会员,主要研究领域为虚拟化、云计算.

sequence of instructions, which is then executed by the accelerator. That is, flexibility comes from software rather than from firmware for the FPGA. As a result, the reconfigurability of FPGAs is not fully utilized. To respond to different requirements flexibly, this paper proposes the fGrain framework that balances performance and generality by managing the granularity of operators. The design of fGrain is based on modern machine learning systems. It combines the hardware accelerators design and the software framework, which are both based on dataflow graph design. We provide the abstraction of FPGA hardware resources as operators and manage FPGA resources at the granularity of operators in the software machine learning framework. We also introduce a user-mode virtualization layer between software framework and hardware operators, which decouples the computing model from the target platform and improves resource utilization and scalability. Equipped with the virtualization layer, we can fully utilize the computing resource in FPGA by employing multiple virtual FPGA devices to run several operators in parallel. This design also allows fGrain to optimize the execution of inference tasks at the software level. Also, we provide a series of configurable operators to support various types of CNN models. They are replaceable and specially optimized. The applicability of our framework is constrained by these kernels. We have already implemented kernels that are required to support most of the CNN models until now. Based on the design mentioned above, we implemented a fGrain prototype based on TensorFlow and OpenCL. We evaluate our implementation on a heterogeneous platform with a CPU, a GPU, and an FPGA device. Our evaluations show that fGrain provides better inference latency than GPU-based solutions for inference tasks with small batch sizes. fGrain achieves a 20% and 11% latency for AlexNet and ResNet inference latency at batch size 1, respectively. Optimization based on fGrain design brings more than 25% optimization to AlexNet execution. Furthermore, the virtualization performance loss is just under 1.3%.

Keywords convolution neural network; filed-programmable gate arrays; machine learning system; user-mode virtualization; open computing language

1 引 言

基于卷积神经网络(CNN)的各类应用,例如目标分割和图像分类[1-2],已在近年来得到广泛使用.以 Caffe、TensorFlow、MXNet 等为代表的机器学习框架[3-6]的提出,使算法开发人员可以更容易地使用异构计算资源来加速 CNN 应用.通过使用这些框架,CNN 设计人员在无需系统和硬件知识的情况下,也可以构建高性能神经网络的实现,并利用诸如 GPU、FPGA、ASIC 等异构硬件来进行训练和推断的加速.同时,近年也有一系列工作提出了不同的 CNN 加速器的设计方案.这些方案旨在增加 CNN 的吞吐量,减少推断延迟,并最小化任务的单位功耗[7-9].在这些加速器中,基于 FPGA 的解决方案在性能,能效和灵活性之间取得了很好的平衡[10-13].因此,基于 FPGA 的 CNN 加速器被认为在未来有被广泛部署的潜力,尤其是在小批量等特殊场景下[14]①.

基于 FPGA 的 CNN 加速器的架构可分为两类:基于指令的架构和基于数据流的架构[15].在基于指令的架构中,特定的硬件加速器带有特定的指令集.对应的编译软件将 CNN 模型编译为一系列指令,然后由加速器执行.已有的研究已经很好地探索了基于指令架构的软件支持.例如,DNNVM[16]是面向基于指令的 FPGA 加速器的全栈编译器.和基于指令架构的加速器不同,基于数据流架构的加速器没有对应的指令集,而是通过实例化几个不同功能的处理单元(PU)来实现对对应的 CNN 模型加速.每个 PU 仅提供特定的功能,例如卷积层的前向计算,池化等.与基于指令的架构相比,基于流的架构可以针对不同网络结构特点进行定制的优化.例如,基于流架构的 fpgaConvNet[17]可以根据网络结构,分别调整每个 PU 的资源分配,以获得更好的

① Microsoft. Azure machine learning accelerate with FPGAs. https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-accelerate-with-fpgas 2019, 4

全局性能.

同时，基于流的动态调整和部署也具有很大的灵活性. Lo 等人[18]在边缘计算的场景中提出了一种基于流的任务分配策略. 利用 FPGA 的部分可重配性，基于流架构的 CNN 加速器可以根据模型的计算特征，选择最合适的 PU 来加速不同的网络. 此外，近年来，随着 FPGA 芯片制程的提升和设计的演化，FPGA 片上资源的数量以及丰富性都有了很大的提升①. 基于数据流架构中的 PU，可以实现对这些资源进行有效管理和高效利用. 由于这些原因，我们在本文中仅讨论对基于流架构的系统支持.

现代机器学习系统为用户在机器学习模型定义、执行时优化、设备管理等方面提供了相当大的灵活性. 在这些机器学习系统的设计中，基于数据流图的方案是一种主流选择[5-6]. 例如，TensorFlow[6]在设计中对计算和控制采用了基于数据流图的统一表示，并使用了延迟执行模型. 借助这两个设计，TensorFlow 可以利用全局信息来对算法的执行进行优化. 同时，它还为异构设备定义了一套通用抽象接口，这为新设备或新架构引入系统提供了便利. 基于这些设计，TensorFlow 支持算子的多态性，即同一算子有多种对应不同硬件的实现. 此外，一些工作利用上述特性将系统级优化集成到 TensorFlow 中，例如，Mirhoseini 等人[19]提出了一种基于强化学习的算子放置算法，将计算图中的顶点，即算子，映射到更合适的设备中以提高整体的效率.

但是，这些机器学习系统无法很好地支持基于 FPGA 的加速器. 在之前的相关工作中，Caffeine[20]可以将 CNN 模型编译为寄存器传输级(RTL)代码，但这一解决方案需要在部署前对不同的模型进行综合和编译，面对模型切换等场景时，灵活性较为有限. TVM/VTA[21-22]是一个从对算子的编译入手，将深度学习的计算图编译成其特定的指令序列的框架. 但是，它仅针对基于指令的架构而设计. 如何在现代机器学习系统中支持基于流架构的加速器尚待探索. 这导致 CNN 设计人员无法享受基于流的加速器带来的好处.

为了进一步简化 CNN 在 FPGA 上的部署流程并提高灵活性，我们提出了 fGrain，这是对现代机器学习系统的一个基于细粒度 FPGA 管理的增强. 在设计中，我们将软件框架中的数据流图和硬件设计中的基于数据流的架构联系起来，并充分支持数据流图表示和延迟执行模型，对软硬协同优化和执行时优化更为友好.

我们的主要贡献可归纳如下：

(1) 我们提出了一种基于 OpenCL 的框架 fGrain，使得用户可以在现代机器学习系统中使用 FPGA. 我们基于一个广泛使用的机器学习框架——TensorFlow，提供了 fGrain 的一种实现. 在 fGrain 中，我们将 FPGA 的硬件资源编译为功能各异的 PU 并抽象为算子，并通过添加算子级的 FPGA 资源管理单元来管理这些资源. 开发人员可以直接使用原生的 TensorFlow 模型从 FPGA 的计算加速中受益.

(2) 我们在 TensorFlow 和物理算子之间引入了虚拟化管理层来提供用户模式的虚拟化功能. 这有助于将计算模型与目标平台解耦，并提高 FPGA 的资源利用率和可扩展性. 借助虚拟化层，我们可以通过使用多个虚拟 FPGA 并行运行多个算子来充分利用 FPGA 中的计算资源.

(3) 我们提供了一系列可配置的 FPGA 算子，用来支持各种类型的 CNN 模型. 算子的实现方式为 OpenCL 编程框架中的内核. 在 CNN 中，大部分计算量集中在了卷积层，为此我们针对不同规模的卷积做了针对性的优化. 我们同时还提供了支持大多数 CNN 模型所需的内核的算子库.

基于上述设计，我们评估了 fGrain 在具有 CPU、GPU 和 FPGA 的异构平台上的实现. CNN 模型的原生的 TensorFlow 代码稍经修改即可通过 fGrain 在 FPGA 器件上执行，并且 fGrain 也支持 TensorFlow 提供的系统级优化. 我们在一些广泛使用的模型上测试了 fGrain 的有效性，并验证了使能系统级优化时带来的性能收益. 我们的实验证明了 fGrain 在小批量 CNN 推理任务中有出色的表现.

本文第 2 节详细介绍 TensorFlow 和 OpenCL 编程模型的背景；第 3 节和第 4 节介绍 fGrain 的设计和实现；实验、相关工作和未来展望分别在第 5、6 和 7 节中介绍；第 8 节为本文的结论部分.

2　背景技术介绍

本节简要介绍了 TensorFlow 的整体架构和其典型的执行时优化. 同时，我们展示了 fGrain 设计

① Xilinx. Xilinx Versal Adaptive Compute Acceleration Platform. https://www.xilinx.com/products/silicon-devices/acap/versal.html 2020，8

实现背后的动机:旨在为现代机器学习系统提供灵活高效的 FPGA 接口.最后,我们还介绍了基于 OpenCL 的 FPGA 开发的一些基本知识.

2.1 TensorFlow

在 TensorFlow[6]的设计中,使用数据流图是一个核心的选择,其用来定义机器学习算法中的计算,表示计算节点的状态和相关的一些操作.这里的数据流图为有向无环图(DAG),由节点和边组成.数据流图中的每个节点都代表一个抽象的计算操作.例如,在神经网络中,通常将一层抽象为一个算子.算子内核(kernel,为了区别于别的内核,标记为 OP-kernel)是在特定类型的设备上运行的特定算子的实现.数据流图中的边(Edge)表示张量(Tensor),代表多维数组.

TensorFlow 的架构可以分为前段框架和后端实现.前端为用户提供了多种高级语言接口,以简单的方式定义机器学习模型,训练策略和系统级优化.前端还包括用于训练和推理机器学习模型的算子库.后端的设计则是为了保证整个系统的高效性.数据流执行器是其中最重要的组件之一.通过采用延迟执行模型,它收集了相关计算图的全部执行信息然后将算子与适当的计算内核相关联.此外,TensorFlow 还提供了针对异构设备的通用抽象,这为 TensorFlow 对异构加速器的设备管理带来了极大的便利.

通过采用数据流表示和延迟执行技术,TensorFlow 可以引入系统级优化,例如优化的算子放置算法[19]、算子的多态[6]以及对数据传输的覆盖等.

其他机器学习系统也通过类似的设计,为用户提供了灵活定义系统级优化的接口,在软件层面提升了机器学习系统的整体效率.但是,之前 FPGA 加速器设计和将 FPGA 集成到机器学习框架中的相关工作,忽略了 FPGA 硬件端与软件框架在系统级优化接口上的结合问题,因此削弱了软件框架和 FPGA 加速器之间的合作,并且导致许多潜在的软硬协同优化无法实施.fGrain 则试图借助虚拟化这一方法解决这个问题.

2.2 基于 OpenCL 的 FPGA 开发

OpenCL[23]是针对异构系统的一款开放的并行编程标准.作为与供应商无关的行业标准,它为各类现代异构平台上的任务并行和数据并行计算提供了可移植性.另一方面,高层次综合(HLS)工具在加速 FPGA 的硬件开发中已引起越来越多的关注.由于 OpenCL 已经广受欢迎,因此 Intel① 和 Xilinx 等 FPGA 供应商分别推出了对 OpenCL 的支持.

如图 1 所示,OpenCL 使用主机-设备编程模型和一组适用于主机和设备的 OpenCL C API.在主机端,用户使用带有 OpenCL API 的通用 C 编写应用程序,这些 API 通过 OpenCL 平台(OpenCL platform)和设备上下文(Context)管理 OpenCL 设备,并通过 OpenCL 主机-设备命令队列控制设备内核状态.主机端代码由通用 C 编译器编译,并与 OpenCL 库链接.在设备端,OpenCL 提供了用于优化内存访问的分层内存,以及用于执行单指令流多数据流(SIMD)的多维度内核(NDRangeKernel)模型.

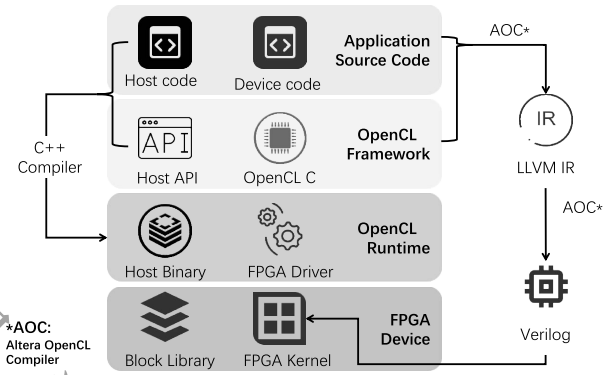


图 1 基于 OpenCL 的 FPGA 开发流程

英特尔为 FPGA 提供了 OpenCL 软件开发套装(SDK),它为 OpenCL 提供了一些扩展,包括设备端代码功能的一些增强.用户可以借助 SDK 定义内核的深度流水线模式来快速构建带有深度流水结构的 PU.同时,配套的离线 OpenCL-FPGA 编译器也提供了一些编译选项对设备端的实现做出如维度展开或循环展开的优化.OpenCL 的这些扩展可以帮助用户充分地利用 FPGA 上的计算资源.离线编译器将 OpenCL 设备代码编译为 LLVM IR,然后生成带有预定义组件和 IP 的 Verilog 代码.最终的二进制流与用户定义的应用程序内核以及一些 OpenCL IP 相结合,以实现 PCIe DMA 和对外部板载 DRAM 的控制.

TensorFlow 还通过使用 SYCL②(一种针对 OpenCL 的 C++单源异构编程)来支持常见的 OpenCL 设备.它扩展了原生 TensorFlow C++计

① Intel. Intel fpga sdk for OpenCL. https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html 2019, 6

② Khronos. SYCL: C++ single-source heterogeneous programming for OpenCL. https://www.khronos.org/sycl/2019, 4

算库 Eigen. 但是，它对 FPGA 的部分可重构等特性的支持较为有限.

3　fGrain 的设计

我们设计 fGrain 的目的是为了将 FPGA 器件集成到符合工业标准的机器学习系统 TensorFlow 中. 通过为 TensorFlow 提供多个虚拟 FPGA(vFPGA)，fGrain 实现了一种细粒度的利用 FPGA 的 CNN 推理加速. 这些虚拟 FPGA 还可适应 TensorFlow 的系统级优化，包括算子放置算法和内核多态性.

3.1　fGrain 的架构

fGrain 的架构图如图 2 所示. 基本设计遵循 TensorFlow 的分层架构，在此基础上，fGrain 增加了对细粒度 FPGA 管理的支持. fGrain 扩展了 TensorFlow 推理库中定义的算子的内核实现层. 为了使内核具有特定的输入和输出多态性，fGrain 在内核实现层中注册了不同的内核. 如图 2 中，我们提供对卷积算子(Conv2D)提供了不同优化的版本：Conv2D3×3、Conv2D1×1 以及 Conv2D. 这些内核将相同的操作映射到不同的 vFPGA，以执行具有特定优化的 CL 内核.

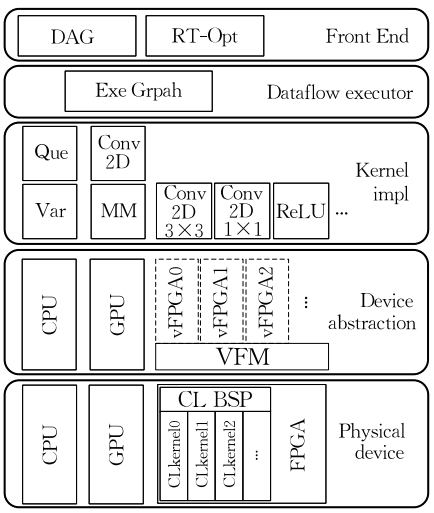


图 2　fGrain 的架构

fGrain 在虚拟 FPGA 监视组件(VFM)的帮助下向 TensorFlow 提供了一组虚拟 FPGA. 这些虚拟 FPGA 在 TensorFlow 中注册为普通设备. 而且 VFM 为 TensorFlow 的通用设备抽象所需的必要方法提供了支持. 因此，他们可以参与 TensorFlow 中的内核映射过程. 通过引入虚拟化，fGrain 借助软件框架实现了对 FPGA 器件的细粒度管理. fGrain 的设计还涉及板上设计. fGrain 基于 OpenCL，为 CNN 推断实现了一些内核. 对于关键操作，fGrain 根据不同参数采用了多种优化，并将其部署在一个 FPGA 设计中.

3.2　fGrain 中的虚拟化设计

如上所述，将虚拟化引入 fGrain 的目标是利用 TensorFlow 算子内核中的多态性，在一台物理 FPGA 器件中实现多个算子或者 TensorFlow 内核的并行性. 此外，这些 vFPGA 具有 TensorFlow 框架中设备抽象的友好接口，因此可以将它们注册为 fGrain 中的 TF 设备，以适应 TensorFlow 中加速器的系统级优化. 例如，fGrain 可以使用基于增强学习的设备放置算法[17]将适当的算子分配给具有特定内核的 vFPGA.

通过 fGrain 中的虚拟化设计，CL 内核被映射到 TensorFlow 框架中的虚拟 FPGA 器件中. 该设计由 3 个主要单元组成：虚拟 FPGA 监视器(VFM)，内核多路复用器和相应的板载设计. 内核多路复用器位于 TF 内核中，将每个算子的实现转发到选定的虚拟 FPGA. VFM 是 OpenCL 原生组件执行器(OpenCL Executor)的扩展，用于维护 vFPGA 之间的公共资源. 它基于预先设定的放置策略将 vFPGA 路由到对应的 CL 内核，并将 TensorFlow 内核映射到 vFPGA. 会话初始化时，fGrain 会将带有几个内核的特定配置的二进制流文件加载到板上.

3.3　设计空间讨论

3.3.1　FPGA 算子的粒度

fGrain 为在 FPGA 上实现 TensorFlow 推理算子库中的算子提供了足够的支持. 但是，在 FPGA CNN 加速器设计中通常会采用三个级别的算子，即完全网络级别、层级和基本操作级别. 网络算子代表一个完整的神经网络，它们是正向传播过程中计算的抽象. 在实现网络算子时，对带宽和内存进行优化很方便. 因此，网络算子具有最卓越的性能，但其灵活性不足以匹配神经网络结构迅速的变化和发展. 选择网络级别的算子一定程度上会牺牲 FPGA 的灵活性；TensorFlow 的算子库中包含的基本运算算子是指数组或矩阵运算. 尽管具有足够的灵活性，但用于基本运算的 FPGA 加速器通常由核心计算单元和相关组件组成. 实际上，部分定制芯片采用的就是这种设计，但 FPGA 的效能比不如类似设计的 ASIC. 虽然有可重构的特性，使其升级的成本较低，但是低等级算子并不具有和网络结构相关的升级空间，且优化的设计也比较固定. 因此，在 fGrain 的设计中，我们利用层算子来使新模型能够将 FPGA 集

成到机器学习系统中,以实现从软件框架和 FPGA 器件的共同优化.层级算子是神经网络中各个层的抽象.它们被广泛用作数据流图中的基本顶点.TensorFlow 的算子和内核库也对层级算子具有足够的支持.对于 FPGA 的设计,一个层级算子通常使用少量资源,因此我们可以部署具有不同优化和功能的各种算子,以适应不同的应用场景.

3.3.2 多态性扩展

fGrain 对 TensorFlow 中算子的"多态性"进行了扩展.TensorFlow 中的算子对不同数据类型和不同输入、输出的数目有着不同的实现和优化方式,这称为算子的多态性.而在传统的 FPGA 算子设计中,可以针对除数据类型之外的参数进行优化,以达到更好的性能.例如,具有不同内核大小的卷积操作常常会在一个 CNN 网络中执行.我们可以针对不同内核大小的算子,在 FPGA 设计上实现与之相对应的循环展开和访存优化.通过适当的调度,特定的加速器可以加速相应的算子.然而,TensorFlow 并不支持特定参数的多态性.因此,我们采用虚拟 FPGA 来实现,相关的讨论在 4.2 节.

3.3.3 基于 fGrain 的优化示例

基于 fGrain,FPGA 器件能够更好地与软件框架协调,对机器学习任务的执行进行系统级的优化.在 fGrain 的设计中,基于虚拟化的 FPGA 片上资源空分复用的方案进一步扩展了软件框架层的优化空间.文中我们选择了结合神经网络结构对 CNN 推断任务中的算子放置进行优化这一例子,作为基于 fGrain 的优化示例.

首先,借助 FPGA 虚拟化,fGrain 可以将 TensorFlow 数据流图中的不同算子的内核映射到同一个物理 FPGA 的不同区域,使得算子在 FPGA 上的执行可以突破软件框架的限制,进行算子内核上的并行执行.在这个基础上,不同区域的 FPGA,被映射为多个虚拟 FPGA.支持软件框架对算子进行合理放置的调整.

一个简单的示例为,若网络结构中存在局部并行的算子.我们可以手动指定,或者配合有效的算子放置算法,对算子进行合理分配.利用虚拟化提供的算子内核并行执行的特性,将并行部分的算子分配到不同的虚拟 FPGA 上.通过基于空分复用的并行执行提升整体效率.在 5.3 节的实验中,我们以网络结构中有并行部分的 AlexNet 网络为例,实验结果的对比显示了基于 fGrain 的系统优化带来的性能提升.

4 fGrain 的实现

在 fGrain 的实现中,我们将 OpenCL 和 TensorFlow 作为 fGrain 的基础.利用 TensorFlow 提供的统一设备接口作为基础,将 OpenCL 的功能进行封装以符合接口标准.

我们设计并实现了两个组件:CL 执行器(CLexecutor)和 CL 上下文(CLcontext)来构成虚拟 FPGA 监视器(VFM)的功能.CLexecutor 将 OpenCL 应用程序接口(API)封装为几种高度抽象的方法,以适应 TensorFlow 提供的设备接口.CLcontext 负责虚拟 FPGA 的管理.以下两节分别详细介绍了 CLexecutor 和 CLcontext 的实现细节.

4.1 CL 执行器

CLexecutor 旨在将 OpenCL 的较为繁杂的主机端 API 进行封装,将其包装为几个简洁的接口.其中有些接口符合 TensorFlow 定义的统一的硬件抽象.这样,我们可以通过 TensorFlow 来进行 FPGA 设备管理和内核调用,并向用户隐藏这些实现细节.另一些接口为 FPGA 或 fGrain 所特需的,如实现 FPGA 配置及 FPGA 虚拟化的接口.

如图 3 所示,在 CLexecutor 中实现的方法包括四种类型,即内存管理,数据传输,虚拟化单元和配置管理器.前两类方法是 TensorFlow 对于异构设备所需的通用抽象接口.在 TensorFlow 的执行过程中,当将算子分配给 FPGA 器件时,这些接口完成片上存储器的分配和回收,以及 CPU 和 FPGA 之间的数据传输;第三类与 CLcontext 有关,以实现与虚拟化相关的功能;第四类用于配置 FPGA,其会根据用户的配置在流程启动前将选用并编译好的 FPGA 配置烧录进 FPGA 中.

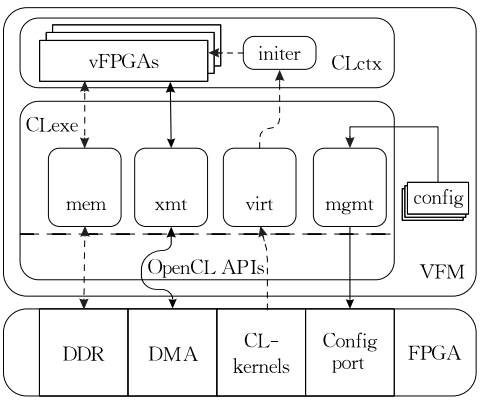


图 3 CL 执行器

这些方法的实现方式相似. 他们采用封装 OpenCL API 和获得的必需参数的想法. 例如，传输操作封装了从主机内存到 CL 缓存的地址转换，并使用 OpenCL API 创建 CL 队列. 转移操作进入队列. 然后，它等待一系列操作(例如传输完成)，然后完成从主机到设备的数据传输. 在特定的设备注册阶段，实例化 CL 执行程序，然后 TensorFlow 通过调用 CL 执行程序中的相应方法来执行相应的功能.

4.2　CL 上下文

通过虚拟化，CLcontext 为上层的 TensorFlow 提供了几种虚拟 FPGA 器件，从而在一个物理 FPGA 上完成了多个计算内核的管理和有效使用. 在 CLcontext 的实现中，我们使用了 API 转发的方法来提供虚拟化功能. 在这里，我们对 OpenCL API 进行了转发. 在 TensorFlow 框架中注册的 vFPGA 可以视为 API 转发代理，它与物理 FPGA 上的计算内核具有一一对应的关系. 虚拟化功能的初始化流程如下. 首先，设置 CL 执行器中的 vFPGA 数量. 然后，CLcontext 初始化对应的队列，并存储每个 vFPGA 的信息，包括 vFPGA 表识(ID)和对应的 CL 命令队列. 在注册过程中，每个 vFPGA 的特定功能都是通过隐式方法定义的. 具体地说，每个虚拟 FPGA 都映射到一个特定的 CL 内核.

图 4 的内容是虚拟 FPGA 的实现和 CLcontext 的结构. 它还描述了 fGrain 中虚拟设备一个典型的工作流程. 在这个例子中，算子被分配到 vFPGA0 上执行. 首先，CLcontext 在 TensorFlow 的设备层中注册 vFPGA. TensorFlow 系统将其维护的多个 vFPGA 视为独立的设备. CLcontext 维护每个 vFPGA 的 vFPGA ID 和 CL 命令队列. 然后，当将算子分配给 vFPGA 时，TensorFlow 会调用统一设备接口中定义的几种方法来执行完整的操作. CLcontext 捕获这些调用并将它们放入 vFPGA 的命令队列中. 如前所述，这些方法由 CLexecutor 实现. 在将操作入队后，CLexecutor 依次在命令队列中执行命令. 当 CL 执行程序执行“内核执行”(图 4 中的 Exe)时，计算流程将会分配给相应的 CL 内核. 计算完成后再将结果返回给数据流执行器，继续后续的计算.

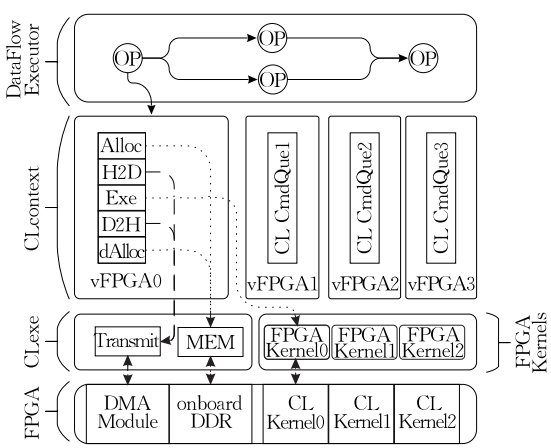


图 4　CL 上下文的结构

5　实验测试

5.1　实验设置

5.1.1　实验平台

我们的测试平台是一台配备 Intel i5-4590 CPU 的台式计算机；带有 Arria 10 FPGA 的 DE5a-netFPGA 开发板则通过 PCI-e Gen3x8 插槽连接到主机，读写带宽分别为 5927.61 MB/s 和 5763.12 MB/s. 我们的测试平台还包含 nVidia GeforceGTX 1080Ti GPU，同样通过 PCI-e Gen3x8 与主机连接，读写带宽为 11.3 GB/s 和 11.2 GB/s；软件层面上，fGrain 基于 TensorFlow(r1.9)实现. fGrain 由 Google 的编译工具 Bazel 编译，并打包为 Python 包. 其对应的算子库被打包为动态库. OpenCL 内核由面向 FPGA(v16.1)的 Intel OpenCL SDK 综合并编译. 整个系统在 ubuntu 16.04 下运行.

5.1.2　模型

两个 CNN 模型 AlexNet[1] 和 ResNet-18[2] 被用作衡量性能的基准. 如前文所述，fGrain 设计目标之一为向软件开发人员隐藏 FPGA 实现的细节. 因此我们在评估中使用的模型皆为原生的 TensorFlow 实现. 在实验中，我们需要插入设备指定相关的代码. 此外，所有评估均采用 32 位浮点数据格式.

5.1.3　FPGA 配置

我们设计并实现了带有可配置内存优化功能的二维卷积内核(Conv2D kernel). 使用不同的配置参数，我们在实验中实例化了四种不同的 FPGA 配置. 表 1 列出了这些配置的 FPGA 资源使用和配置参数. 在资源使用情况中，我们统计了不同配置方案对逻辑资源(Logic)、查找表(ALUT)、触发器(FF)、片上随机内存(RAM)、数字信号处理器(DSP)等资源等使用情况.

在 CL 内核配置中，计算单元数目(#CU)是指一个内核中的计算单元的数目，每个 CU 都是对应设计的实例化，所以增加 CU 数目，会占用更多的

表 1 FPGA 配置情况

FPGA 配置方案	FPGA 资源使用					CL 内核配置			虚拟化开启
	Logic/%	ALUT/%	FF/%	RAM/%	DSP/%	#CU	#CLkernel	BufS	
A	38	20	20	47	11	1	1	3×3	否
B	59	31	31	86	18	2	1	3×3	否
C	60	31	31	87	20	1	2	3×3	否
D	60	31	31	87	20	1	2	3×3	是

资源,同时也能带来更好的性能;CL 内核数量(#CLkernel)是指一个 FPGA 配置方案中 OpenCL 内核的数量.我们在二维卷积内核的设计中采用了基于缓冲区的内存优化,缓冲区大小(BufS)是指设计中的缓冲区大小.只有方案 D 开启了 fGrain 提供的虚拟化支持.

5.1.4 功耗测量

我们使用供应商提供的工具来测量功耗.英特尔功耗估算器①通过分析 RTL 代码估算了 FPGA 板的功耗.在我们的设计中,其工作时的功耗为 29 瓦特,空载待机状态时为 11.7 瓦特.

同时,我们通过软件工具②③获得了 CPU 和 GPU 的功率.在我们的评估中,通过累计每个设备在工作和空载时的功耗来计算总功耗.然后,我们将总功率除以端到端时间以得到平均功率.

5.1.5 实验设计

为了评估 fGrain 的性能,我们将 fGrain 的基于 FPGA 的解决方案与通用的 TensorFlow 的基于 GPU 的解决方案进行了比较.在测试中,我们使用了两种不同的 CNN 模型在批量大小为 1、2 和 3 时的推断性能延迟.

为了评估虚拟化功能带来的性能增益,我们使用三种不同的 FPGA 配置在不同的批处理量下测量了 AlexNet.

为了评估 fGrain 带来的性能损失,我们修改了 PipeCNN[24],使其适应 fGrain.这是一种基于 OpenCL 的开源 CNN 加速器设计.我们测量了在使用 fGrain 和不使用 fGrain 的情况下 PipeCNN 加速 Alexnet 的性能.

在所有实验中,我们测量了所有推理任务的端到端时间消耗,包括加载权重和图片的时间.

5.2 跨平台对比

图 5、图 6 显示了基于 FPGA 的解决方案与基于 GPU 的解决方案之间的比较结果.我们测量了两种方案各自的推断延迟和加速时间.在基于 FPGA 的解决方案中,我们使用了表 1 中的方案 B,并且数据流图中的所有 conv2d 算子都放置在 FPGA 设备上.其他操作则放置在 CPU.在基于 GPU 的解决方案中,所有 conv2d 算子都放置在 GPU 上,其他的放置在 CPU.

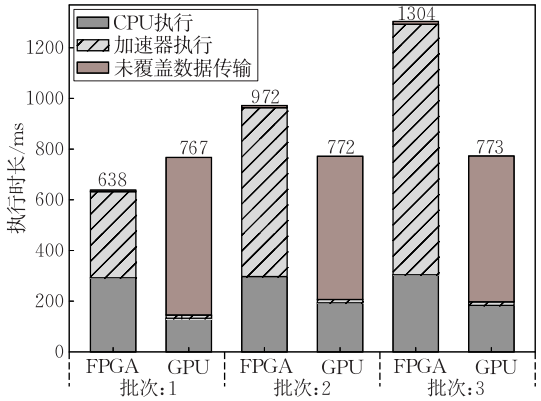


图 5 AlexNet 的推断延迟

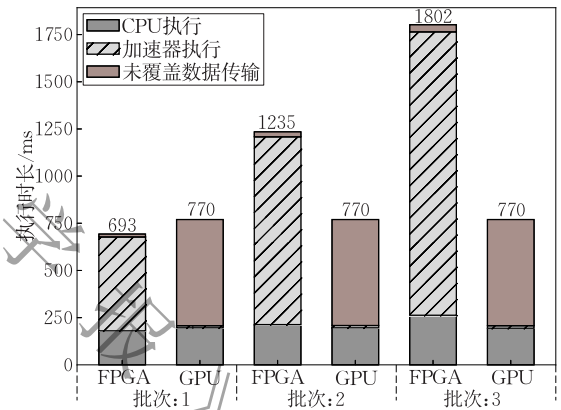


图 6 Resnet18 的推断延迟

图 5 比较了批量大小分别为 1、2、3 的基于 FPGA 和 GPU 的解决方案的 AlexNet 的推断时间.当批量为 1 时,基于 FPGA 的解决方案优于基于 GPU 的解决方案,且有 20.2%的提升.考虑到平均能耗,可将能源效率提高 50%左右.如图所示,GPU 的计算时间仅占整个执行过程的一小部分.这是因

① Intel. Arria 10 early power estimator. https://www.intel.com/content/www/us/en/programmable/support/support-resources/operation-and-testing/power/a10-power-estimator.html 2019, 4

② Brown L. Turbo stat report processor frequency and idle statistics. https://www.mankier.com/8/turbostat 2019, 4

③ Nvidia. Nvidia system management interface. https://developer.nvidia.com/nvidia-system-management-interface 2019, 4

为未覆盖传输开销比基于 FPGA 的解决方案要大得多. 在 ResNet18 的情况下，GPU 解决方案的未覆盖数据传输成本超过 562 ms，而 FPGA 解决方案的成本仅为 15 ms. 这主要得益于 FPGA 在小批量数据传输上的优势和 fGrain 的设计使数据传输可以被内核执行覆盖.

具体来说，这里传输时间成本悬殊的原因由两个原因构成，首先 GPU 的 DMA 由于需要经常处理大批量的数据，因此，在经过特殊优化后，在处理小批量的数据时性能反而会下降；其次，GPU 由于制程和频率的优势，且这里展示的 fGrain 中的算子实现并未发挥 FPGA 的全部性能. 所以如图所示，FPGA 方案的加速器执行时间要远高于 GPU 方案的. 在这种情况下，由于 fGrain 中的设计，过长的加速器执行时间反而覆盖了数据传输时间，所以最后的结果中，FPGA 的净传输时间较短.

图 6 展示了不同批量大小和加速方案的 ResNet-18 推理时间的实验结果. ResNet-18 具有 17 个卷积层算子，比 AlexNet 的计算量要大. 因此，基于 GPU 的解决方案的结果比 AlexNet 上的结果更为明显. 但是，当批量为 1 时，基于 FPGA 的解决方案仍然比基于 GPU 的方案有了 11.1% 的提升. 具体理由和对 AlexNet 做出的分析相同.

5.3　FPGA 虚拟化带来的增益

在 fGrain 的设计中，我们通过实现虚拟 FPGA 管理层来实现与 OpenCL 内核相对应的虚拟 FPGA. 在这一组实验中，我们使用虚拟化功能打破了 TensorFlow 框架的局限性，以在一个物理 FPGA 中实现 TensorFlow 内核的并行. 对于具有并行构造的模型，只要合理放置它们，就可以显着提高性能.

在这一系列实验中，我们使用了 AlexNet 作为测试模型. AlexNet 具有 5 组 8 二维卷积算子，我们可以将其表示为 $\{C_0, C_{1\text{-}0}, C_{1\text{-}1}, C_2, C_{3\text{-}0}, C_{3\text{-}1}, C_{4\text{-}0}, C_{4\text{-}1}\}$. 显然，在这个模型中，第一、第三、第四层中包含两个算子的并行.

在实验设置中，前两组启用了 FPGA 虚拟化功能，但是应用了不同的放置策略. 具体的放置策略为：第一组是：FPGA-0，{C0，C1-0，C2，C3-0，C4-0}；FPGA-1，{C1-1，C3-1，C4-1}. 第二组是：FPGA-0，{C0，C2}；FPGA-1，{C1-0，C1-1，C3-0，C3-1，C4-0，C4-1}.

显然，第一组的放置策略最充分地利用了模型的并行性，第二组则根本没有利用模型内部的并行，而第三组没有使用 FPGA 虚拟化的功能. 实验结果反映了由虚拟化解决方案带来的并行处理和性能优化. 在 FPGA 配置方案中，我们将配置 D 用于前两个组，将配置 C 用于第三组.

图 7 显示了在三种不同方案中批量大小为 1，执行 AlexNet 模型时 CL 内核的执行情况. 使用 FPGA 虚拟化和合理的算子放置的第一组方案通过合理地利用模型的并行性提高了性能. 它标记为“Pa”. 第二组称为“Pb”，使用虚拟化功能，但是算子的放置方案不合理，因此不会带来很多性能改进. 第三组是控制组，没有使用 FPGA 虚拟化功能，FPGA 上的资源没有得到充分利用.

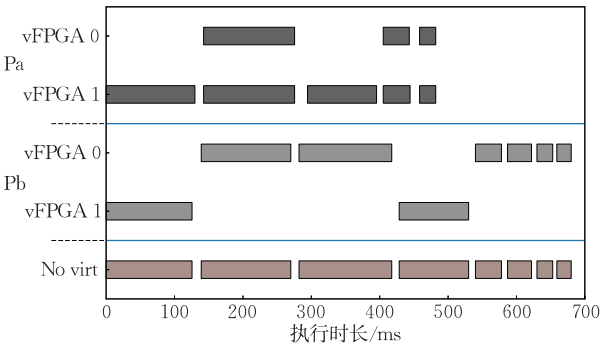


图 7　启用虚拟化对 AlexNet 执行情况对影响

图 8 显示了在批量为 1、2、3 情况下三组实验设置下 AlexNet 的推断延迟时间. 通过使用带有合理算子放置的 FPGA 虚拟化方案，我们充分地利用了 AlexNet 模型的并行特性. 在基于 FPGA 的方案内，算子的执行时间获得了近 40% 的优化，并且端到端执行时间也得到 25% 左右的改善.

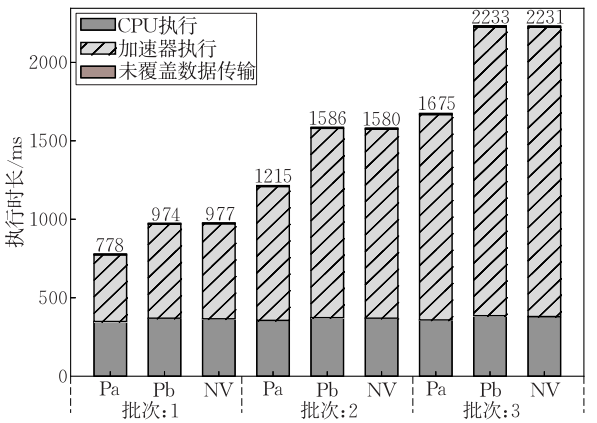


图 8　fGrain 的虚拟化执行

5.4　案例分析

在我们的案例研究中，我们将 PipeCNN[24] 修改以适应 fGrain. PipeCNN 是基于 OpenCL 开发的 CNN 的 FPGA 加速器. 它采用基于数据流的设计，

并很好地利用了 FPGA OpenCL 开发套件中提供的基于 FIFO(先入先出队列)的数据传输功能.在其实现的内核中,有两个内核专门负责从 FPGA 的全局内存抓取数据并实现了内存优化以提高性能.此外,PipeCNN 基于定点精度实现,很大地提升了性能.

在此评估中,我们将 PipeCNN 引入 fGrain.我们利用了 PipeCNN 实现的 OpenCL 内核,并将这些 OpenCL 内核映射为 fGrain 框架中的算子.基于这些算子,我们重新实现了 PipeCNN 项目中的两个 CNN 网络.其中,由于 fGrain 原型体所借助的 TensorFlow 版本对 8 位定点数并不支持,我们选择了在算子层对 8 位定点数进行支持.

在 fGrain+pipeCNN 的执行过程中,fGrain 负责 OpenCL 的初始化,FPGA 的配置以及 OpenCL 内核的主机端配置以及执行控制.算子间的数据传输也打包进算子中,以实现对低精度的支持.

我们比较了封装在 fGrain 中的 PipeCNN 和原生 PipeCNN 的性能,以测试使用 fGrain 带来的性能损失.测试基准是 AlexNet 和 VGG16 网络的推断的端到端的延迟时间,这个时间包括权重和图像加载到主内存的开销,权重和图像向 FPGA 内存的传输以及计算时间.批次为 1、2 和 3.

实验结果如图 9 所示,fGrain 带来的平均性能损耗为 1.28%.但是,fGrain 负责维持 OpenCL 运行时,可以帮助用户处理繁琐的平台维护,数据传输和内存管理.

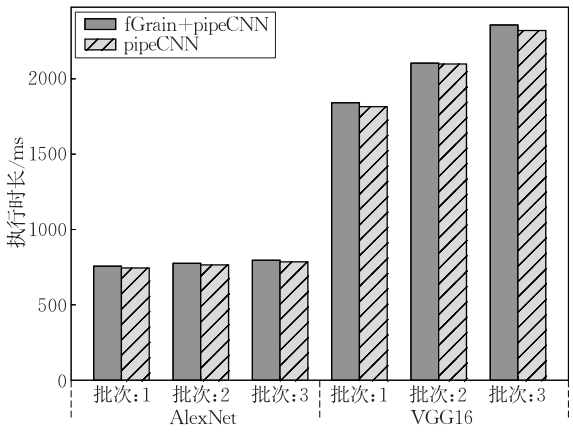


图 9 将 PipeCNN 引入 fGrain

6 相关工作

FPGA CNN 加速器的架构可以分为两种类型:基于指令的和基于流的.基于指令的加速器采用计算引擎和自定义指令集体系结构(ISA).VTA[22]与 TVM 集成在一起,并在 FPGA 上有一个实现.Sharma 等人开发了 DNN-Weaver[10],以根据 Caffe 中的高层次限定生成对应给定神经网络-FPGA(DNN,FPGA)对的可综合的加速器.他们提出了一种适用于 DNN 加速器的数据流 ISA 和与此 ISA 关联的手动优化模板.考虑到基于指令的加速器的可重构性有限,fGrain 的设计符合基于流的设计.最近基于流的设计从缓冲区设计[8,24],基于计算的优化[8]和内核融合[14,25]提高了效率.

先前的研究已经可以使 FPGA 集成入机器学习框架[10,20,26].Zhang 等人提出的 Caffeine[20],是一个软/硬件协同设计库,通过统一表示卷积层和全连接层来启用 Caffe 中的 FPGA.Guan 等人提出了 FP-DNN[26],这是一个使用预定义的寄存器-高级语义综合对(RTL-HLS)模板自动将 TensorFlow 模型映射为一个基于 FPGA 的加速器实现的框架.Zhao 等人的工作[27],在 TensorFlow 的框架下设计了一系列 C 风格的 API 完成对 FPGA 设备的管理,基于这些 API 实现将上层框架算子映射到底层 FPGA 上算子的流程控制.Hadjis 等人的工作[28],提出了一个端对端的开源工具链,可以将 TensorFlow 为代表的前端框架中定义的模型映射为经过优化的 DNN 加速器.这些工作主要集中在将 DNN 模型映射为 FPGA 设计的机制上.由于模板或表示形式是相对静态的,因此用户需要在 FPGA 平台更改时手动修改设计,这会给生产环境带来维护困难.同时,其工作的系统级优化(包括带宽平衡,计算任务的放置,通信时间的覆盖等)在硬件设计中是固定的,从而降低了软件框架提供的灵活性.fGrain 隐藏了所有硬件细节,并将细粒度的虚拟 FPGA 作为通用设备引入到软件框架中.因此,虚拟 FPGA 可以包含在由软件框架驱动的统一优化中.

7 未来展望

随着机器学习硬件加速系统的相关研究工作的推进,FPGA 在系统中的定位逐渐从核心加速器件变为功能器件.前者承担的是主要计算任务,而后者会负责数据路由,通信加速等功能性任务.在数据中心领域,FPGA 扮演的角色逐渐从计算加速卡转向以智能网卡为代表的通信加速卡.因此,在 fGrain 的下一步工作中,除了包含算子优化,多卡支持等基

础功能的完善外，fGrain 会进一步地适应 FPGA 器件在机器学习系统中的新的定位.

一方面，fGrain 会面向数据中心中大规模分布式训练任务的场景. 在此场景下，FPGA 作为通信加速卡加速服务期间数据传输. 进一步地，FPGA 可以在通信加速的同时承担线速下对部分 IO 密集型计算进行加速的任务. fGrain 可以在此基础上，向机器学习框架提供作为通信加速卡的 FPGA 的一个更为合理抽象. 即对 FPGA 负责的通信加速任务和计算加速任务向上提供统一抽象的算子，并将其纳入框架层进行整体的优化，提升整体效率.

另一方面，我们会探索扩展，优化 fGrain 的虚拟化功能. 目前 fGrain 提供了基于 API 转发的 FPGA 的空分复用的解决方案. 我们会考虑在这一框架下，丰富虚拟化技术的应用，如对 I/O 密集型的算子可以提供时分复用的虚拟化方案；以及考虑实现基于 fGrain 虚拟化技术的资源池化，提升数据中心背景下资源的利用率.

8　结　论

在本文中，我们提出了 fGrain，这是一种可通过细粒度的 FPGA 管理来加速 CNN 推理的异构框架. fGrain 将 FPGA 集成到现代机器学习系统 TensorFlow 中，用户在其中运行原生的 TensorFlow 模型即可从 FPGA 加速中受益. 借助虚拟化层和一系列可配置的运算符，fGrain 获得了可适应各种类型的 CNN 模型和 FPGA 平台的灵活性和可伸缩性.

参　考　文　献

[1] Krizhevsky A, Sutskever I, Hinton G E. ImageNet Classification with deep convolutional neural networks//Proceedings of the Neural Information Processing Systems. Lake Tahoe, USA, 2012: 1106-1114

[2] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep residual learning for image recognition//Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, USA, 2016: 770-778

[3] Jia Yangqing, Shelhamer E, Donahue J, et al. Caffe: Convolutional architecture for fast feature embedding//Proceedings of the ACM International Conference on Multimedia. Orlando, USA, 2014: 675-678

[4] Chen Tianqi, Li Mu, Li Yutian, et al. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv:1512.01274, 2015

[5] Al-Rfou R, Alain G, Almahairi A, et al. Theano: A python framework for fast computation of mathematical expressions. arXiv:1605.02688, 2016

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

[7] Jouppi N P, Young C, Patil N, et al. In datacenter performance analysis of a tensor processing unit//Proceedings of the 44th Annual International Symposium on Computer Architecture. Toronto, Canada, 2017: 1-12

[8] Aydonat U, O'Connell S, Capalija D, et al. An OpenCL deep learning accelerator on Arria 10//Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, USA, 2017: 55-64

[9] Jiao Yang, Han Liang, Jin Rong, et al. A 12nm programmable convolution-efficient neural-processing-unit chip achieving 825TOPS//Proceedings of the 2020 IEEE International Solid-State Circuits Conference. San Francisco, USA, 2020: 136-140

[10] Sharma H, Park J, Mahajan D, et al. From high-level deep neural models to FPGAs//Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture. Taipei, China, 2016: 17:1-17:12

[11] Zhang Xiaofan, Wang Junsong, Zhu Chao, et al. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs//Proceedings of the International Conference on Computer-Aided Design. San Diego, USA, 2018: 1-8

[12] Wang Chao, Wang Teng, Ma Xiang, et al. Research progress on FPGA-based machine learning hardware acceleration. Chinese Journal of Computers, 2020, 43(6): 1161-1182(in Chinese)
(王超，王腾，马翔等. 基于 FPGA 的机器学习硬件加速研究进展. 计算机学报，2020，43(6)：1161-1182)

[13] Wu Yan-Xia, Liang Kai, Liu Ying, et al. The progress and trends of FPGA-based accelerators in deep learning. Chinese Journal of Computers, 2019, 42(11): 2461-2480(in Chinese)
(吴艳霞，梁楷，刘颖等. 深度学习 FPGA 加速器的进展与趋势. 计算机学报，2019，42(11)：2461-2480)

[14] Fowers J, Ovtcharov K, Papamichael M, et al. A configurable cloud-scale DNN processor for real-time AI//Proceedings of the 45th ACM/IEEE Annual International Symposium on Computer Architecture. Los Angeles, USA, 2018: 1-14

[15] Venieris S I, Kouris A, Bouganis C-S. Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions. ACM Computing Surveys, 2018, 51(3): 1-39

[16] Xing Yu, Liang Shuang, Sui Lingzhi, et al. DNNVM: End-to-end compiler leveraging heterogeneous optimizations on FPGA-based CNN accelerators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(10): 2668-2681

[17] Venieris S I, Bouganis C-S. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs//Proceedings of the 24th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines. Washington, USA, 2016: 40-47

[18] Lo Chi, Su Yu-Yi, Lee Chun-Yi, et al. A dynamic deep neural network design for efficient workload allocation in edge computing//Proceedings of the 2017 IEEE International Conference on Computer Design. Boston, USA, 2017: 273-280

[19] Mirhoseini A, Pham H, Le Q V, et al. Device placement optimization with reinforcement learning//Proceedings of the 34th International Conference on Machine Learning. Sydney, Australia, 2017: 2430-2439

[20] Zhang Chen, Fang Zhenman, Zhou Peipei, et al. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks//Proceedings of the 35th International Conference on Computer-Aided Design. Austin, USA, 2016: 1-8

[21] Chen Tianqi, Moreau T, Jiang Ziheng, et al. TVM: An automated end-to-end optimizing compiler for deep learning//Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA, 2018: 578-594

[22] Moreau T, Chen Tianqi, Jiang Ziheng, et al. VTA: A open hardware-software stack for deep learning. arXiv:1807.04188, 2018

[23] Stone J E, Gohara D, Shi Guochun. OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in Science & Engineering, 2010, 12(3): 66-73

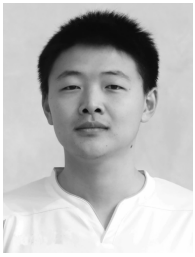
[24] Wang Dong, Xu Ke, Jiang Diankun. PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks//Proceedings of the International Conference on Field Programmable Technology. Melbourne, Australia, 2017: 279-282

[25] Xiao Qingcheng, Liang Yun, Lu Liqiang, et al. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs//Proceedings of the 54th Annual Design Automation Conference. Austin, USA, 2017: 1-6

[26] Guan Yijin, Liang Hao, Xu Ningyi, et al. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates//Proceedings of the 25th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines. Napa, USA, 2017: 152-159

[27] Zhao Jiacheng, Chang Yisong, Li Denghui, et al. On retargeting the AI programming framework to new hardwares//Proceedings of the Network and Parallel Computing-15th International Conference on Network and Parallel Computing. Muroran, Japan, 2018: 39-51

[28] Hadjis S, Olukotun K. TensorFlow to cloud FPGAs: Tradeoffs for accelerating deep neural networks//Proceedings of the 29th International Conference on Field Programmable Logic and Applications. Barcelona, Spain, 2019: 360-366



GUO Kai-Cheng, Ph. D. candidate. His research interests include machine learning system and FPGA virtualization.

WU Cheng-Gang, Ph. D. candidate. His research interests include distributed system and machine learning accelerator.

ZHANG Wei-Feng, Ph. D., fellow of Alibaba Cloud Intelligence. His research interests include heterogeneous computing, compiler, and high performance hardware software co-optimization.

QI Zheng-Wei, Ph. D., professor. His research interests include system software, virtualization, and program analysis.

GUAN Hai-Bing, Ph. D., professor. His research interests include cloud computing and virtualization.

Background

In recent years, artificial intelligence algorithms have achieved rapid development, and the algorithms represented by CNNs have been widely used in various scenarios. How to improve their efficiency has become a widespread concern. Due to the natural parallelism and high customizability of FPGAs, accelerators designed based on them have significant advantages in scenarios such as targeting specific models and low precision. However, due to the inherent problems of the FPGA ecosystem, such as high development threshold, cumbersome toolchain, and not being friendly enough for software, algorithm developers, it has become a widespread problem to make FPGAs efficiently manageable and exploitable by existing machine learning systems.

Related work considers the use of mapping. The mapping workflow input is the model code from the machine learning framework; it is mapped to the corresponding FPGA accelerator

design; workflow also includes the optimization of these designs; the output is the binary stream used to configure the FPGA. the designs of FPGA accelerators can be divided into two types, instruction-based and data stream-based. These works based on mapping generally target instruction-based FPGA accelerators. The accelerator based on the data stream still lacks the support of machine learning frameworks. Besides, mapping-based solutions are only connected to the upper-level machine learning framework during the mapping phase and are relatively independent during the execution phase, preventing framework-based optimization and management at runtime.

The work presented in this paper proposes a machine learning framework optimization for data stream-based FPGA accelerators, which allows the existing framework to manage and optimize the runtime of FPGA accelerators based on the data stream. Furthermore, using the user-state virtualization, a virtual FPGA based on the processing unit in the accelerator design is proposed, which is more in line with the abstraction of the device in the machine learning system and also enables the fine-grained management, use, and optimization of the FPGA device by the upper framework.

This work is partially sponsored by the National Natural Science Foundation of China (61732010), the National Key Research & Development Program of China (2016YFB1000502), and the Alibaba Group through the Alibaba Innovative Research (AIR) Program.