g

**学士学位论文中期报告**

**论文题目**：基于TVM的异构计算资源平台的设计与实现

**专 业**：计算机科学与技术

**本 科 生**：曲硕

**学 号**：17373482

**指导教师**：李莹

**北京航空航天大学计算机学院**

2021年4月10日

目录

[1 课题研究内容回顾 2](#_Toc1029701272)

[1.1 研究目的 2](#_Toc1172538992)

[1.2 研究背景 3](#_Toc786239673)

[1.3 研究意义 3](#_Toc890681535)

[1.4 研究现状 4](#_Toc1480475592)

[1.4.1 神经网络模型部署 4](#_Toc790743231)

[1.4.2 神经网络模型编译器 4](#_Toc531117107)

[1.4.3 TVM 5](#_Toc843439044)

[1.5 课题研究内容 5](#_Toc1632026519)

[1.5.1 环境部署 5](#_Toc1032318434)

[1.5.2 工具构建 6](#_Toc393066790)

[1.5.3 模型编译 6](#_Toc1045001955)

[1.5.4 平台设计与实现 7](#_Toc1771128733)

[1.5.5 平台测试 7](#_Toc134783418)

[1.6 关键技术和难点 7](#_Toc1746720643)

[1.6.1 平台的部署环境复杂 7](#_Toc1530371611)

[1.6.2依赖的软件和工具众多 7](#_Toc2043802118)

[1.6.3 需要支持的框架和设备众多 8](#_Toc1711820307)

[1.7 研究方案 8](#_Toc822683698)

[2 已完成的工作 8](#_Toc363024852)

[2.1 需求分析 8](#_Toc1427749470)

[2.1.1 用户需求 8](#_Toc578987838)

[2.1.2 产品需求 9](#_Toc1030835593)

[2.1.3 产品功能需求 9](#_Toc208435863)

[2.2 系统设计 9](#_Toc1419393009)

[2.2.1 接口设计 10](#_Toc1956215303)

[2.2.2 平台设计 11](#_Toc158396351)

[2.2.3 工具链设计 11](#_Toc1724892866)

[2.2.4 平台系统环境设计 11](#_Toc530796433)

[2.3 系统实现 11](#_Toc299765688)

[2.3.1 平台系统环境实现 12](#_Toc1456751464)

[2.3.2 工具链实现 14](#_Toc1560497706)

[2.3.3 平台实现 16](#_Toc1472304680)

[2.3.4 接口实现 19](#_Toc95507489)

[1.1.2. 平台的实现 20](#_Toc303695593)

[2.4 实验结果 21](#_Toc805296625)

[3 待完成的工作及问题 23](#_Toc886250720)

[3.1 待完成的工作 23](#_Toc834812700)

[3.2 存在的问题 23](#_Toc1648735669)

[4 后期进度安排 24](#_Toc370793591)

## 1 课题研究内容回顾

该部分主要介绍研究目的，研究背景，研究意义，研究现状，本课题的研究内容，关键技术和难点，以及研究的方案。

### 1.1 研究目的

基于TVM，实现一个用户友好，简单，易用的平台，使得用户能够把神经网络模型部署到不同架构的硬件设备上，而不需要了解TVM的具体使用和底层模型编译部署的具体细节。

### 1.2 研究背景

近些年来，以神经网络为工具的深度学习在众多领域都取得了重大的成功，如计算机视觉和自然语言处理。随着神经网络模型的研究深入和训练速度的加快，产生了大量的模型。传统的模型部署的方式是把模型部署在服务器上，应用通过网络来使用模型。但是，把模型部署在移动设备上可以降低网络延迟，降低网络负载，同时可以提高用户隐私。基于这些优点，把神经网络模型部署在移动设备上的需求越来越多。但是真正把模型落地，面临着一个难以解决的问题。目前存在着众多的深度学习的框架，如Pytorch，Tensorflow，Caffe等，得到的模型结构有所区别，同时，众多设备类型如CPUs，GPUs，FPGAs，TPUs，在指令架构上有所不同。所以，需要一个统一的方式把众多的模型结构部署到众多的设备类型上。

同时，神经网络模型需要占用庞大的计算资源也是一个不可忽视的问题，而在许多的设备上计算资源比较匮乏，如移动手机和一些嵌入式设备。所以在部署过程中还需要对模型进行优化。目前的模型优化大多基于深度学习框架在计算图上的优化，这样的优化层级太高，不能处理和具体硬件相关的算子层面上的优化。其次，深度学习框架大多只针对了服务器端的GPU进行了优化，而没有针对每种不同的硬件设备进行优化。

所以，研究人员希望能够通过编译器来解决这个问题，通过编译的技术把不同深度学习框架的模型结构部署到不同架构的异构设备上，同时处理高层面的计算图优化和面向具体设备的优化。目前，随着深度学习的不断发展，神经网络编译器得到了越来越多的关注和重大的发展。

### 1.3 研究意义

首先，神经网络模型的功能日益丰富，多种硬件设备对模型的需求日益增多。传统的模型部署到异构计算设备往往需要很多的人工的调整，浪费了大量的人力。所以，提供一个统一的，易于使用的异构计算平台的部署，同时保持和人工调整相近或更优的准确度，能够节省大量的人力。

其次，TVM是一个神经网络模型的编译器，能够实现不同的深度学习框架模型到不同硬件设备的部署同时执行计算图和面向硬件的优化。所以基于TVM来实现异构计算资源的平台能够更好的支持多种深度学习框架和众多的硬件设备，同时使得部署的模型效率更高。但是，TVM的使用涉及到复杂的环境部署，对于多种框架和多种硬件设备的使用复杂。所以，在TVM基础上封装异构计算资源的平台能够大大简化模型编译，部署的流程。

最终，通过封装的平台，能够使得用户更简洁，高效的实现神经网络模型到不同设备的部署。

### 1.4 研究现状

该部分分别介绍神经网络模型部署，神经网络模型编译器的研究现状，以及TVM工具的基本概念。

#### 1.4.1 神经网络模型部署

神经网络模型的部署往往都是与训练该模型的框架耦合在一起的，需要深度学习框架实现对应硬件平台的runtime。如Tensorflow Lite可以把Tensorflow的模型部署在Android设备上。Pytorch目前也推出了自己的模型部署到移动设备的runtime，但是这些解决方案只支持本身框架训练出的模型，且只支持移动设备。同时这些方式缺少针对具体硬件的优化，还需要人工对模型进行调整。

#### 1.4.2 神经网络模型编译器

人工调整模型并不高效，所以产生了神经网络模型的编译器，自动化的生产可以直接在特定平台上运行的网络模型。如，GLOW，TVM等。

为了同时支持众多的深度学习框架和广泛的硬件平台，目前主流的神经网络编译器都包含两部分，编译前端和编译后端[14]。同时包含两个中间表示，High-Level IR，Low-Level IR。编译前端能够处理多种主流的深度学习框架，如Tensorflow，Pytorch，MXNet等。编译前端把不同框架的不同模型结构转化为统一的计算图表示，High-Level IR。同时针对计算图进行优化，如计算图改写，算子融合等。编译后端把High-Level IR转换为Low-Level IR，并进行面向具体硬件的优化，如硬件元语对应，内存分配，并行计算等。

#### 1.4.3 TVM

TVM是一个端到端的神经网络模型的编译器工具链，支持目前主流的前端的深度学习框架，如Pytorch，Tensorflow，Caffe等，同时支持部署到广泛的后端硬件设备，如CPU，服务器端GPU，移动端GPU，FPGA等。在编译过程中TVM同时对模型进行优化，分别进行高层次的计算图优化和低层次的硬件相关的算子优化来保证部署到硬件设备上的模型的效率。

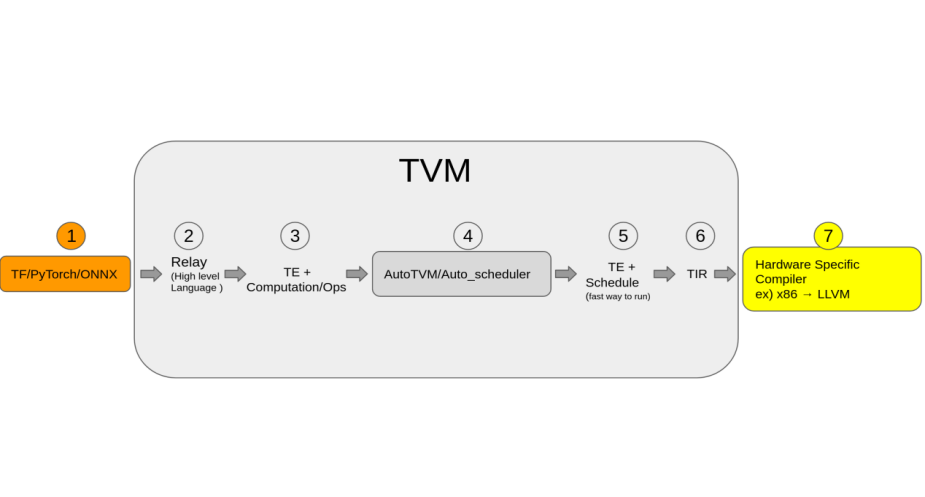


图1.1 TVM系统结构

TVM的具体执行流程为，对于不同深度学习框架训练得到的模型转化为统一的计算图表示，然后对该计算图进行数据流的重写，得到一个优化的计算图。之后对该计算图进行算子层面的优化，对该计算图进行算子的合并，同时针对不同不同硬件设备的内存和指令结构进行具体的优化。

### 1.5 课题研究内容

本课题的主要研究内容有环境配置，平台依赖的第三方工具的构建，神经网络模型的构建，异构计算资源平台的设计与实现，平台测试这几个方面。

#### 1.5.1 环境配置

为了实现多框架和多设备的支持，平台后端需要复杂的环境，如工具构建的一些编译软件，GPU的一些计算库，各个深度学习框架的Python库，构建Android应用时所依赖的Android SDK，JDK，和GPU的驱动等等。

**表1.1 平台环境**

|  |  |
| --- | --- |
| 驱动 | Nidia Driver |
| GPU计算库 | Cuda,Cudnn,OpenCL |
| 深度学习框架 | Pytorch,Tensorflow,Keras,Onnx |
| Android SDK | Emulator,Build Tools,Ndk,Platform Tools,Cmdline Tools |
| 构建工具 | GCC,LLVM,CMake,Make，Maven，Gradle |
| JAVA | JDK |

如上表所示，平台依赖的软件环境复杂，合理的安装和管理软件以及软件之间的版本依赖是本研究重要的工作内容。

#### 1.5.2 工具链构建

本研究所依赖的部分工具没有构建好的软件包，还需要从源码进行构建。

首先，需要构建本平台依赖的TVM工具，使用CMake和Make工具来编译TVM的动态连接库，为了使平台能够支持多种设备和计算平台，还需要依赖Cuda，Cudnn，OpenCL，LLVM等动态连接库。编译好TVM的动态连接库后需要打包为Python的库，使得能够通过TVM的Python前端来使用TVM。

其次，该平台需要把神经网络模型部署到Android设备上，所以还需要构建TVM的Java前端。使得用户能够使用Java来调用部署到Android设备上的模型。

最后，因为该平台需要把模型部署在Android设备上，所以需要一个Android App来使Android设备和本地电脑进行通信。在构建Android App时需要依赖Android SDK等软件，使用Maven和Gradle进行App的构建。

#### 1.5.3 模型编译

该平台需要支持多个神经网络框架模型的编译，所以需要了解每个框架的基本概念和一些基本的模型定义，模型保存，模型加载的API。其次，需要具体使用TVM对每种框架的模型进行编译，得到统一的中间表示和参数，需要了解TVM编译每种框架的API。

#### 1.5.4 平台设计与实现

平台需要把TVM编译，部署的流程进行封装，给用户暴露出一个简洁易用的接口。首先是接口的设计，接口要满足简洁易用的原则。接口通过三个关键的变量来实现多框架和多设备的支持。Frame变量表示用户输入模型框架，需要支持Pytorch，Tensorflow，Keras，Onnx等框架。Device表示模型部署到的设备，需要支持本地电脑，树莓派，Android，FPGA等设备。Target表示模型部署到的计算平台。对不同的设备来说，需要支持的计算平台不同，对本地电脑来说，需要支持CPU和Cuda，对Android设备来说，需要支持CPU，OpenCL和Vulkan。通过这个接口，用户得到部署好的模型deplpyed\_model，该模型可以像普通的Pytorch模型一样直接通过deplpyed\_model(input)来得到输出。

#### 1.5.5 平台测试

在实现异构计算资源平台后，需要对平台进行测试。分别对平台多个框架模型的编译和多设备的部署进行测试。

### 1.6 关键技术和难点

#### 1.6.1 平台的部署环境复杂

如上文所示，该平台需要依赖众多的软件，在安装软件过程中需要处理遇到的各种问题，同时解决不同软件之间的版本依赖问题更是本项研究中的难点。安装和管理众多的软件需要对系统有一定的了解，在遇到问题时也需要快速的定位和解决问题。

#### 1.6.2依赖的软件和工具众多

在编译TVM的动态连接库时需要用到GCC和CMake，在构建Android App时需要用到Maven和Gradle，同时还需要adb等Android工具来控制Android设备。所以需要对这些工具的使用有一个基本的掌握。

#### 1.6.3 需要支持的框架和设备众多

该框架需要支持很多的深度学习框架，所以需要了解这些框架的基本使用，尤其是模型定义，模型加载和模型保存的部分。同时该平台需要支持多种设备，所以对多种设备及其系统都需要有一定的了解，并且需要能够使用具体的软件对多种设备进行操作。最后，该平台还需要支持很多GPU计算平台，需要对GPU的计算和多种GPU计算库有一定的了解。

### 1.7 研究方案

首先阅读TVM相关的论文，了解TVM的基本架构，基本概念和应用。

其次通过搜索引擎来安装平台所需的各种软件。

1. 通过教程了学习每种构建工具的基本使用，之后使用工具具体构建出TVM工具和Android App，并通过搜索引擎来解决构建过程中出现的问题。
2. 阅读TVM官方的教程，结合多种深度学习框架的官方教程，实践TVM的模型的编译和部署的流程。
3. 设计并实现异构计算资源平台，能够达到预期的功能。

最后，对平台进行测试。

## 2 已完成的工作

该部分首先介绍该平台的需求分析，紧接着根据需求分析介绍给出系统的设计方案，然后阐述系统的实现，最后给出实验和结果。

### 2.1 需求分析

本需求分析部分将会从三个方面，用户需求，产品需求和产品功能需求来阐述。其中，用户需求主要是用户想要做什么。产品需求将用户的需求进行整理，说明本平台要做什么来满足用户的要求。产品功能需求进一步通过阐述产品需要哪些功能来实现产品的需求。

#### 2.1.1 用户需求

在深度学习具体的应用场景中，训练好的神经网络模型往往需要在具体的设备上执行，如服务器，移动手机，嵌入式设备等等。就导致了用户训练模型和执行模型的设备的架构不同。所以，用户面临的问题就是需要把训练好的模型部署到不同的设备上，在部署过程中还需要针对目标设备的架构进行模型的调整和优化。 所以，用户希望能够简化模型部署的流程，同时保证模型的准确率。

#### 2.1.2 产品需求

为了满足用户的需求，产品需要实现一个平台来帮助用户进行模型的部署。考虑到目前的神经网络模型多采用Python进行开发，所以本平台应该要以Python库的形式进行实现，方便用户在其项目中进行使用。其次，平台应该有应用的广泛性，即平台应该能够支持目前主流的深度学习框架，同时应该支持部署到主流的硬件设备上。最后，平台要保证易用性，保证用户简单，容易的部署模型。

#### 2.1.3 产品功能需求

首先， 平台应该实现为Python的库，同时实现模型部署的API，实现的API要保证统一，简洁，方便用户调用。

其次，平台需要支持Pytorch，Tensorflow，Keras，Onnx等深度学习框架。以及服务器，移动手机，FPGA等硬件设备。

最后，平台要能够对模型进行面向计算图和硬件架构的优化，保证部署模型的准确率。

### 2.2 系统设计

基于需求分析，把异构计算资源平台的系统设计分为以下几个方面，从上到下依次是，接口设计，平台设计，工具链设计，平台系统环境设计。

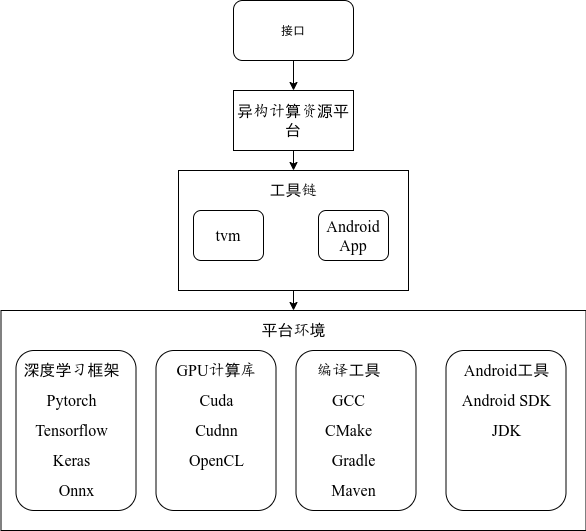


图2.1 系统设计

#### 2.2.1 接口设计

接口是平台暴露给用户使用API，与用户进行直接的交互，用户通过调用该API来进行模型的部署。

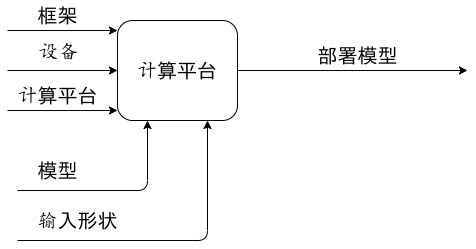


图2.1 接口设计

平台通过把每种设备的部署抽象为一个类来实现相应的功能，通过一些参数来与用户交互。其中：

* 框架参数：用户部署模型使用的深度学习框架。预支持Pytorch，Tensorflow，Keras，Onnx等框架。
* 设备参数：用户部署的目标设备，预支持本地电脑，Android设备，FPGA等设备。
* 计算平台：用户部署的目标计算平台，此目标计算平台与具体的设备相关，如本地电脑，预支持的计算平台为CPU，和Cuda。对Andorid设备来说，预支持CPU，OpenCL，Vulkan等。
* 模型：是用户待部署的模型对象。
* 输入形状：是用户模型输入的向量的形状。

通过该接口，用户最终得到部署后的模型对象，该对象可以像其他模型一样，计算输出。

#### 2.2.2 平台设计

异构计算资源平台由几个部分组成：

* 编译模块：把用户输入的模型编译为计算图的中间表示，把多种框架的不同计算图编译为统一的中间形式方便后续的部署。
* 部署模块：把得到的计算图中间表示生成能够在具体硬件执行的动态链接库，能够部署到对应的平台执行推理。
* 接口的实现：把接口实现为Python的类，在接口中封装编译和部署的功能，通过暴露出参数，来让用户调用。

#### 2.2.3 工具链设计

平台通过基于第三方的工具来辅助实现平台的功能。本项目中将会使用TVM和Android Rpc App来完成平台的功能。

TVM是一个端到端的神经网络模型的编译器工具链，支持目前主流的前端的深度学习框架，如Pytorch，Tensorflow，Onnx等，同时支持部署到广泛的后端硬件设备，如CPU，服务器端GPU，移动端GPU，FPGA等。所以，本项目采用TVM来实现模型的编译和部署。

其次，为了能够部署到Android设备，需要主机和Android设备进行通信，所以使用Android Rpc App来使主机和Android设备进行通信，能够把模型上传到Android设备。

#### 2.2.4 平台环境设计

平台系统环境指的是该平台依赖的所有底层软件和操作系统的总和。实现该平台需要依赖复杂的软件。其中包括深度学习框架，Pytorch，Tensorflow，keras，Onnx等。GPU计算库，Cuda，Cudnn，OpenCL等。编译工具GCC，CMake，Gradle，Maven等。构建Android App需要的JDK和Android SDK等。

### 2.3 系统实现

这一部分主要阐述异构计算资源平台的具体实现过程，具体的实现细节。这一部分将以和系统设计相反的顺序，从底到上来阐述系统的实现，分别是平台系统环境实现，工具链实现，平台实现和接口实现。

#### 2.3.1 平台环境配置实现

平台系统环境的实现主要包括，实现平台的操作系统和硬件信息，平台依赖的底层软件安装和不同软件版本的管理方法。

##### 2.3.1.1 本地环境

本地的主机采用的是64位的Ubuntu系统，CPU为64位的intel i5 10210U，GPU为Nvidia MX250。内存为16G内存。

##### 2.3.1.2 软件安装

对于Nidia Driver和Cuda和Cudnn，从Nidia的官网下载下载器进行安装，在安装显卡驱动前，要在BIOS中关闭安全启动的选项。同时要关闭nvidia-drm模块，具体的安装命令如下：

|  |
| --- |
| # 关闭图形界面  sudo systemctl isolate multi-user.target  # 关闭nvidia-drm模块  sudo modprobe -r nvidia-drm  # 执行安装文件  sudo sh install.sh  # 启动图形界面  sudo systemctl start graphical.target |

对于GCC，CMake，Make，Maven，Gradle等构建工具使用Ubuntu的包管理器进行安装，使用命令sudo apt install <package>进行安装。

对于Android SDK等软件，使用Android的sdkmanager进行安装，使用命令sdkmanager --list来查看可安装的包，使用sdkmanager --install <package>来安装所需的软件包.

对于深度学习框架，使用Python的包管理器pip进行安装，使用命令pip install <package>进行安装。

**表2.1 软件版本**

|  |  |  |
| --- | --- | --- |
|  | 软件 | 版本 |
| 驱动 | Nidia Driver | 460.39 |
| GPU计算库 | Cuda | 11.1 |
| Cudnn | 8.0.5 |
| OpenCL | 2.2 |
| 深度学习框架 | Pytorch | 1.8.1+cu111 |
| Tensorflow | 2.4.1 |
| Keras | 2.4.0 |
| Onnx | 1.8.1 |
| Android SDK | Emulator | 30.5.4 |
| Build Tools | 27.0.3 |
| Platform Tools | 31.0.1 |
| Cmdline Tools | 3.0 |
| Ndk | 22.1 |
| 构建工具 | GCC | 9.3.0 |
| CMake | 3.16.3 |
| Make | 4.2.1 |
| Gradle | 4.4.1 |
| Maven | 3.6.3 |
| LLVM | 10 |
| JAVA | JDK | 8 |

##### 2.3.1.3 软件版本管理

安装软件后，平台还需要采用合理的方式对软件及其版本进行管理，同时需要设置一些环境变量来保证软件的正常使用。

对于Cuda来说，把Cuda的文件放在系统的/usr/local/cuda/cuda-version目录下，不同的Cuda版本放在不同的cuda-version目录下，同时设置一个链接文件current指向当前使用的版本，可以方便的进行版本的管理，同时设置CUDA\_HOME=/usr/local/cuda/current环境变量，同时需要把Cuda的库文件目录加入LD\_LIBRARY\_PATH环境变量中，保证连接器ld能够找到Cuda的动态连接库，把Cuda的可执行文件目录加入PATH环境变量中。

对于JDK来说，采用和Cuda相同的管理方法，把不同的JDK版本放在/usr/local/jdk/jdk-version目录下，设置链接文件指向当前使用的版本。同时需要指定环境变量JAVA\_HOME=/usr/local/jdk/current，同时把JDK的可执行文件目录加入PATH环境变量中。

对于Android SDK来说，需要指定环境变量ANDROID\_SDK\_ROOT=/usr/local/android-sdk。使用sdkmanager安装的ndk，build-tools，platform-tools，cmdline-tools，emulator等软件包会安装到这个目录下。

#### 2.3.2 工具链实现

平台依赖的TVM和Android Rpc App需要从源码进行编译和安装，这一部分分别阐述TVM和Android App的编译流程。

##### 2.3.2.1 编译TVM

这部分给出TVM工具从源码具体的编译环境和编译步骤。

###### 2.3.2.1.1 编译环境

TVM的构建环境采用上述本地电脑的环境，采用64位Ubuntu系统，intel i5 10210U CPU，Nidia MX250 GPU。依赖的Cuda版本为11.1，Cudnn版本为8.0.5，OpenCL的版本为2.2，LLVM版本为10。编译工具GCC版本为9.3.0，CMake版本为3.16.3，Make版本为4.2.0。同时按照上述环境搭建所述设置环境变量，保证在编译时能够找到Cuda，Cudnn，OpenCL，LLVM的动态连接库。

###### 2.3.2.1.2 编译步骤

首先从TVM的github仓库使用命令git clone --recursive <repo>下载TVM的源代码。

其次进入TVM源代码的目录，创建构建目录并设置编译配置文件，在源代码的根目录下创建build构建目录，使用命令cp cmake/config.cmake build将配置文件复制到build目录中，之后修改配置文件，使编译的动态链接库链接指定的动态链接库，具体的配置如下：

|  |
| --- |
| set(USE\_CUDA ON)  set(USE\_CUDNN ON)  set(USE\_OPENCL ON)  set(USE\_LLVM ON)  set(USE\_RPC ON)  set(USE\_GRAPH\_EXECUTOR ON)  set(USE\_PROFILER ON) |

1. 使用如下命令编译TVM的动态链接库：

|  |
| --- |
| cd build  cmake ..  make -j4 |

编译完成后，会在build目录下出现libtvm.so和libtvm\_runtime.so等文件。

最后，把TVM的路径加入到PYTHONPATH环境变量中，使得Python的解释器能够找到TVM包。

##### 2.3.2.2 编译Android App

这部分给出，Android Rpc App的编译环境，编译步骤，以及将得到的Apk文件安装到Android手机的具体细节。

###### 2.3.2.2.1 编译环境

该App的编译环境采用64位Ubuntu系统，intel i5 10210U CPU，Nvidia MX250 GPU。依赖的JDK版本为8。Android SDK的Build Tools版本为27.0.3，Ndk的版本为22.1。

###### 2.3.2.2.2 编译步骤

首先，构建TVM的JAVA前端的包，该App依赖了TVM的JAVA前端，在TVM源代码的根目录下，使用make jvmpkg来构建这个包，这个命令会使用Maven进行构建，构建完成后，使用命令make jvminstall来把这个包安装到Maven仓库中。

其次，构建Android Rpc App，使用命令cd apps/android\_rpc来进入这个App的根目录，因为Google修改了Maven仓库，所以在build.gradle文件中把Maven的仓库修改为https://dl.google.com/dl/android/maven2/。之后创建local.properties文件，文件内容为sdk.dir=/path/to/android-sdk来指定Android SDK的路径。

1. 创建构建App的配置文件，该配置文件的具体内容如下：

|  |
| --- |
| APP\_ABI = arm64-v8a  APP\_PLATFORM = android-24  USE\_OPENCL = 1  # 加入OpenCL的头文件目录  ADD\_C\_INCLUDES = /usr/include/CL |

1. 使用命令gradle clean build来构建该App。

最后，使用adb install <apk>来安装此App到Android设备上。



图3.1 Android App界面

#### 2.3.3 平台实现

这部分分别介绍平台基于TVM工具具体实现编译模块和部署模块的细节。

##### 2.3.3.1 编译模块

使用TVM对多种框架的模型进行编译，同时使用具体的框架的接口来进行模型文件的加载，得到模型的对象后使用TVM的接口对模型对象进行编译得到中间表示的计算图和模型的参数。

下面以Pytorch框架的模型为例，展示具体的编译过程：

|  |
| --- |
| import torch  import torchvision  import tvm  from tvm import relay  # 得到Pytorch模型  model\_name = 'resnet18'  model = getattr(torchvision.models, model\_name)(pretrained=True)  model = model.eval()  # 使用Pytorch的接口将模型转化为Scripted模型  input\_shape = [1, 3, 224, 224]  input\_data = torch.randn(input\_shape)  scripted\_model = torch.jit.trace(model, input\_data).eval()  # 使用TVM进行编译  input\_name = 'input0'  shape\_list = [(input\_name, img.shape)]  mod, params = relay.frontend.from\_pytorch(scripted\_model, shape\_list) |

编译后得到mod为计算图的中间表示，params为模型的参数。

##### 2.3.3.2 部署模块

分别介绍部署到本地的电脑和部署到Android设备的细节。

###### 2.3.3.2.1 部署到本地

TVM把得到的计算图的中间表示和参数部署到本地需要指定部署的target\_host和target，target\_host用来指定在CPU端执行的代码，一般为llvm，target用来指定具体的计算平台，在本地可以使用CPU计算或者GPU计算，所以target可以为llvm或cuda。下面为具体的代码：

|  |
| --- |
| # 使用CPU进行计算  target = 'llvm'  target\_host = 'llvm'  ctx = tvm.cpu(0)  # 得到动态链接库  with tvm.transform.PassContext(opt\_level=3):  lib = relay.build(mod, target=target, target\_host=target\_host, params=params)  # 通过TVM执行动态链接库  m = graph\_runtime.GraphModule(lib['default'](ctx))  m.set\_input(input\_name, input)  m.run()  tvm\_output = m.get\_output(0) |

###### 2.3.3.2.2 部署到Android

部署到Android与部署到本地不同，部署到Android，需要让本地电脑能够与Android设备进行通信，能够把得到的动态链接库上传到Android设备，让动态链接库在Android设备上执行。采用Rpc来使本地电脑和Android进行通信，首先在本地，使用命令python -m tvm.exec.rpc\_tracker --host=<host> --port=<port>来创建一个Rpc的服务端。在Android设备上，使用前文构建安装的App，输入指定的ip和端口，这样就可以使Android设备和本地进行通信。使用TVM部署到Android，同样需要指定target\_host和target，target\_host是在Android端的CPU执行的代码，通常为llvm，target是Android端的计算平台，可以是CPU，OpenCL和Vlukan。其次，在创建可以在Android端执行的动态链接库时，因为Android端的架构和本地不同，所以不能使用本地的编译器，需要使用TVM的ndk工具进行编译，然后将此上传到Android端，下面是具体的代码：

|  |
| --- |
| with tvm.transform.PassContext(opt\_level=3):  lib = relay.build(mod, target=target, params=params)  # 使用ndk来编译为动态链接库  fcompile = ndk.create\_shared  lib.export\_library('models/net.so', fcompile)  # 创建一个Rpc tracker来进行通信  tracker\_host = os.environ.get('TVM\_TRACKER\_HOST', '0.0.0.0')  tracker\_port = int(os.environ.get('TVM\_TRACKER\_PORT', '9190'))  key = 'android'  tracker = rpc.connect\_tracker(tracker\_host, tracker\_port)  remote = tracker.request(key, priority=0, session\_timeout=60)  # 上传动态链接库到Android端  remote.upload('models/net.so')  rlib = remote.load\_module('net.so')  # 指定使用OpenCL进行计算  dev = remote.cl(0)  module = runtime.GraphModule(rlib['default'](dev))  module.set\_input(input\_name, tvm.nd.array(x.astype(dtype)))  module.run()  out = module.get\_output(0) |

#### 2.3.4 接口实现

对于平台的接口实现来说，通过一个基类BaseTVMHelper来表示共通的编译部署流程，对于每种具体的设备，通过继承该基类来实现一个针对具体设备的编译部署流程，如部署到本地的LocalTVMHelper和部署到Android的AndroidTVMHelper。

**表3.1 BaseTVMHelper接口**

|  |  |  |
| --- | --- | --- |
| 变量 | 类型 | 含义 |
| device | string | 部署到的设备 |
| frame | string | 模型使用的框架 |
| model | object | 模型对象 |
| input\_shape | list | 模型输入的形状 |
| opt\_level | int | 优化等级 |
| target | string | 设备上的计算平台 |
| compile\_arg\_dict={} | dict | 额外的模型编译参数 |

其中，device是部署到的设备，目前支持local，和android。frame是模型的框架，目前支持pytorch，tensorflow，keras，和onnx。model是具体框架加载后的模型对象。input\_shape是模型输入的形状。opt\_level是模型优化的等级，目前支持1，2，3。target是具体设备上的计算平台，部署到不同设备上target的值不同，如部署到local，target为cpu或cuda。部署到android设备上，target为cpu，opencl，或vulkan。

部署到本地的接口与BaseTVMHelper相同，部署到android设备上还需要一些额外的参数。

**表3.2 AndroidTVMHelper接口**

|  |  |  |
| --- | --- | --- |
| 变量 | 类型 | 含义 |
| tracker\_host | string | 创建Rpc Tracker的host |
| tracker\_port | int | 创建Rpc Tracker的端口 |
| key | string | 连接Rpc的key |

下面通过具体的代码来展示平台接口的使用：

|  |
| --- |
| import torch  import torchvision  import numpy as np  # 引入Helper类  from PlatfromTVM import LocalTVMHelper, AndroidTVMHelper  # 获得模型  model = getattr(torchvision.models, ‘resnet18’)(pretrained=True)  # 部署到local设备，使用CPU计算  deployed\_model = LocalTVMHelper(device=’local’,frame=’pytorch’  model=model,input\_shape=[1,3,224,224],opt\_level=3,target=’cpu’)  # 通过部署的模型计算  out = deployed\_model(input) |

下面是使用该平台和不是该平台把模型部署到android设备上的代码对比，左侧是使用该平台，右侧是不使用该平台：

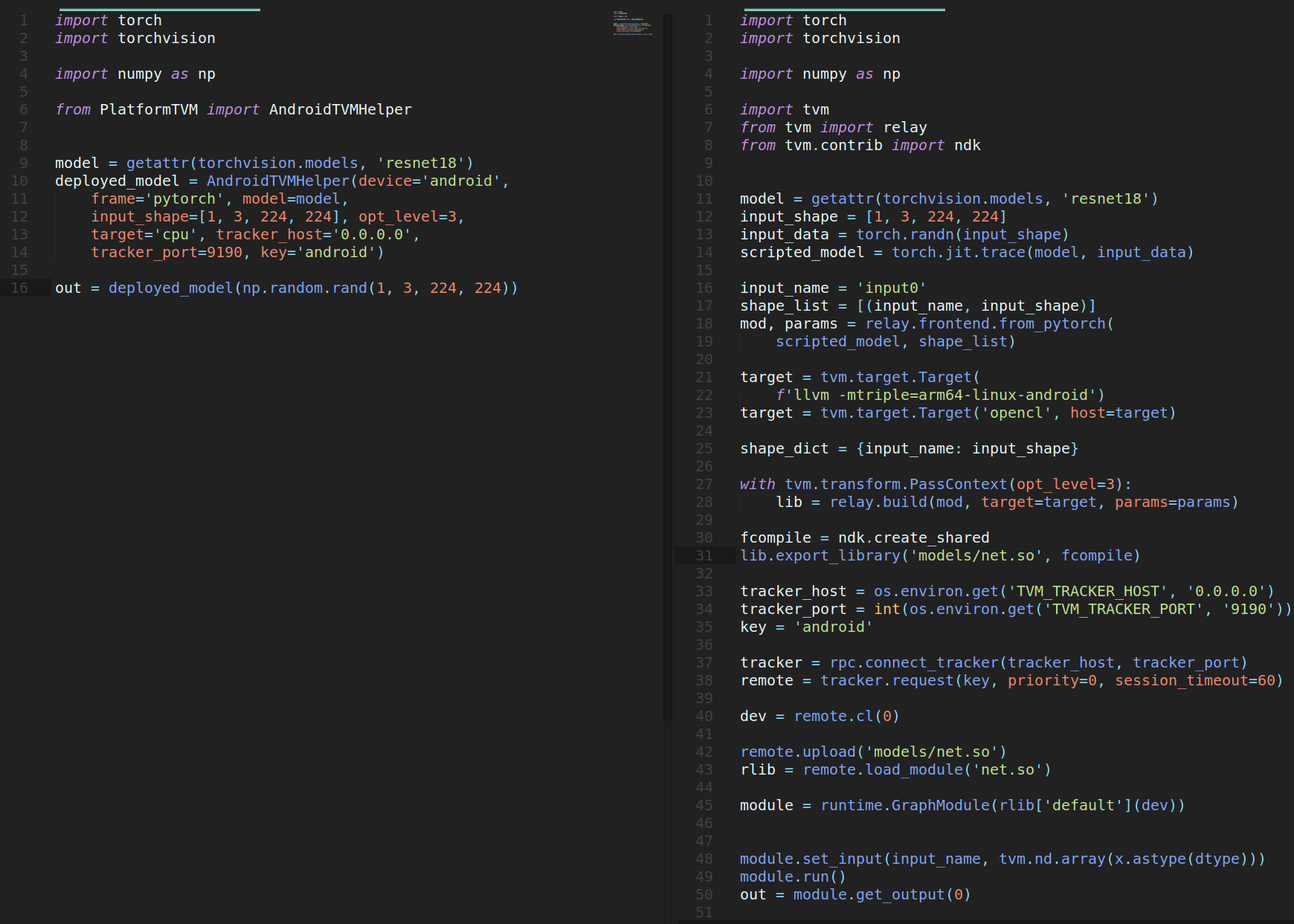


图 3.2 平台使用对比

可以看到，使用该平台大大简化了模型编译部署的难度，同时使用户不需要了解使用TVM编译部署的具体细节或是使用Rpc通信的具体细节。

### 2.4 实验结果

这部展示使用该平台分别把模型部署到本地电脑和部署到Android设备的实验流程和实验结果。

#### 2.4.1 部署到本地实验

部署到本地的实验主要验证平台部署到本地的功能的正确性，同时分别把模型部署到CPU和Cuda的计算平台上，分别验证不同的推理时间。

##### 2.4.1.1 实验环境

模型部署的本地电脑采用的是64位的Ubuntu系统，使用intel i5 10210U CPU，Nidia MX250 GPU。

##### 2.4.1.2 实验步骤

首先,从torchvison库中获得预训练的resnet18模型。

其次，使用该平台的接口，将模型部署到本地设备上，分别使用CPU和GPU进行计算。

最后，分别使用CPU和GPU执行推理100次，记录执行的时间。

##### 2.4.1.3 实验结果

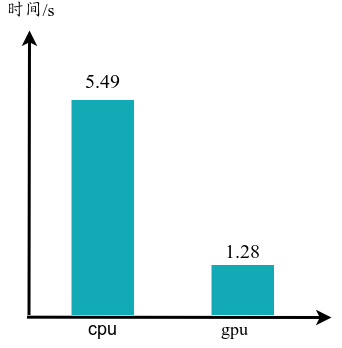
****

图4.1 部署本地实验结果

可以看到，部署到本地的模型在CPU和GPU上分别执行100次推理时,GPU只需要1.28s而CPU需要5.49s。

#### 2.4.2 部署到Android

部署到Android的实验来验证本地主机与Andorid设备的通信是否正常，同时验证该平台部署到Android设备的功能的正确性。该实验分别把模型部署到Android端的CPU，OpenCL，Vulkan计算平台上，验证每种计算平台执行推理的时间。

##### 2.4.2.1 实验环境

本实验的本地端采用64位Ubuntu系统，使用intel i5 10210U CPU，Nidia MX250 GPU。Android端是使用android11系统，使用MTK Helio G90T CPU，ARM Mali G76 MC4 GPU。

##### 2.4.2.2 实验步骤

首先，使用torchvision库获得预训练的resnet18模型。

其次，使用平台的接口把模型部署到Android设备上，同时在Android端使用前文构建的App，输入指定的ip，port，key来打开Rpc通信。同时可以使用命令python -m tvm.exec.query\_rpc\_tracker --host=host --port=port来检验Android设备是否连接到了本地电脑。

最后，分别使用CPU，OpenCL，Vulkan执行推理，记录执行时间。

##### 2.4.2.3 实验结果

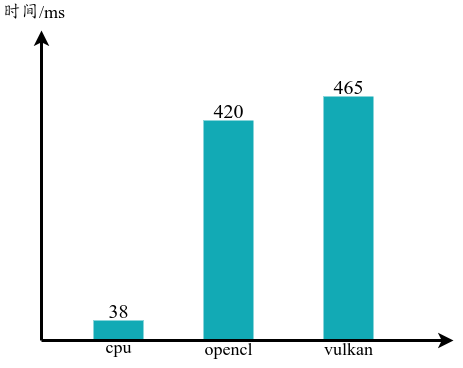
****

图4.2 部署Android实验结果

虽然在Android设备上使用了GPU，但是反而比CPU慢，因为TVM没有针对该GPU的优化，所以导致了执行速度下降明显。

## 3 待完成的工作及问题

### 3.1 待完成的工作

* 支持部署到FPGA
* 完成平台整体的代码实现
* 平台功能测试
* 论文撰写

### 3.2 存在的问题

* 本地电脑与Android设备连接超过一定时间会断开，仍需要解决。

## 4 后期进度安排

|  |  |
| --- | --- |
| 2021年4月18日-2021年4月25日 | 完成FPGA的部署 |
| 2021年4月25日-2021年5月10日 | 完成平台整体实现和测试， |
| 2021年5月10日-2021年5月30日 | 撰写论文 |
| 2021年6月1日- | 完成终期答辩 |