RAMMER：Enabling Holistic Deep Learning Compiler Optimizations with rTasks

嘉宾：赵薇

RAMMER: Enabling Holistic Deep Learning Compiler Optimizations with *r*Tasks

Lingxiao Ma[*] [†◇] Zhiqiang Xie[*] [‡◇] Zhi Yang[†] Jilong Xue[◇] Youshan Miao[◇]
Wei Cui[◇] Wenxiang Hu[◇] Fan Yang[◇] Lintao Zhang[◇] Lidong Zhou[◇]

[†]*Peking University* [‡]*ShanghaiTech University* [◇]*Microsoft Research*

Abstract

Performing Deep Neural Network (DNN) computation on hardware accelerators efficiently is challenging. Existing DNN frameworks and compilers often treat the DNN operators in a data flow graph (DFG) as opaque library functions and schedule them onto accelerators to be executed individually. They rely on another layer of scheduler, often implemented in hardware, to exploit the parallelism available in the operators. Such a two-layered approach incurs significant scheduling overhead and often cannot fully utilize the available hardware resources. In this paper, we propose RAMMER, a DNN compiler design that optimizes the execution of DNN workloads on massively parallel accelerators. RAMMER generates an efficient static spatio-temporal schedule for a DNN at compile time to minimize scheduling overhead. It maximizes hardware utilization by holistically exploiting parallelism through inter- and intra- operator co-scheduling. RAMMER achieves this by proposing several novel, hardware neutral, and clean abstractions for the computation tasks and

perform DNN computation. Efficient DNN computation on these devices is an important topic that has attracted much research attention in recent years [23, 28, 32, 40, 52]. One of the key factors that affect the efficiency of DNN computation is scheduling, i.e. deciding the order to perform various pieces of computation on the target hardware. The importance of scheduling in general is well known and has been thoroughly studied [20, 39]. However, there is little work discussing scheduling for DNN computation on hardware devices specifically.

The computational pattern of a deep neural network is usually modeled as a data flow graph (DFG), where each node corresponds to an operator, which represents a unit of computation such as matrix multiplication, while an edge depicts the dependency between operators. This representation naturally contains two levels of parallelism. The first level is the inter-operator parallelism, where operators that do not have dependencies in the DFG may run in parallel. The second level is the intra-operator parallelism, where an operator such

目录

背景与挑战

PyTorch

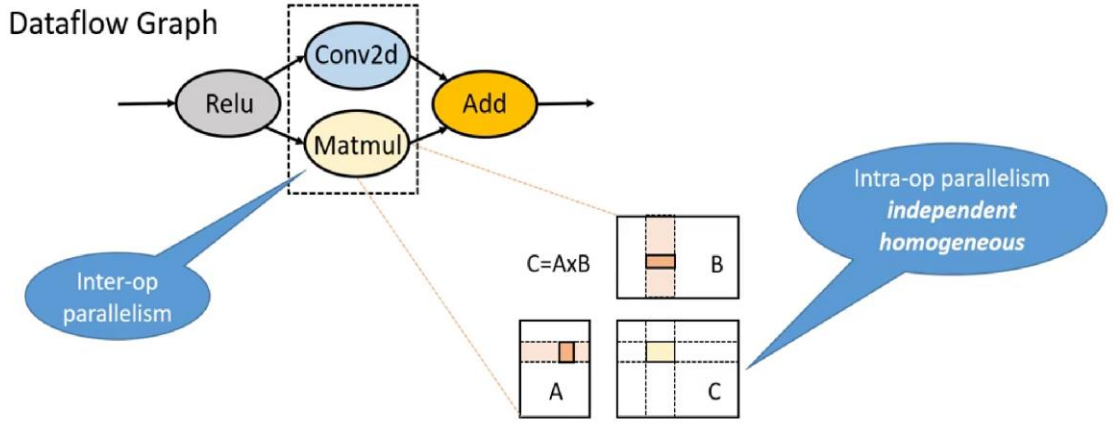
mxnet

TensorFlow

现有的深度学习框架通常把神经网络计算抽象为由算子与依赖关系构建而成的数据流图DFG

两个级别的并行性：

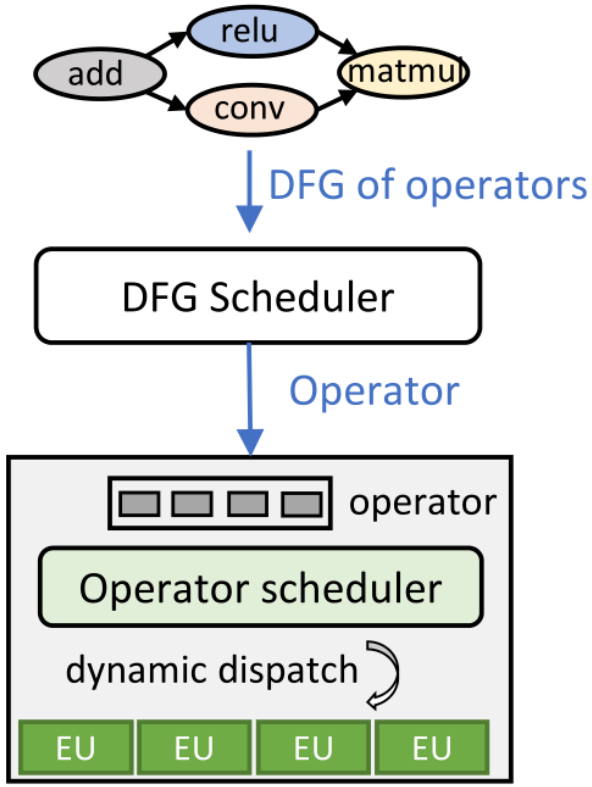
- 算子间并行性（inter-operator parallelism）
- 算子内并行性（intra-operator parallelism）



三个挑战

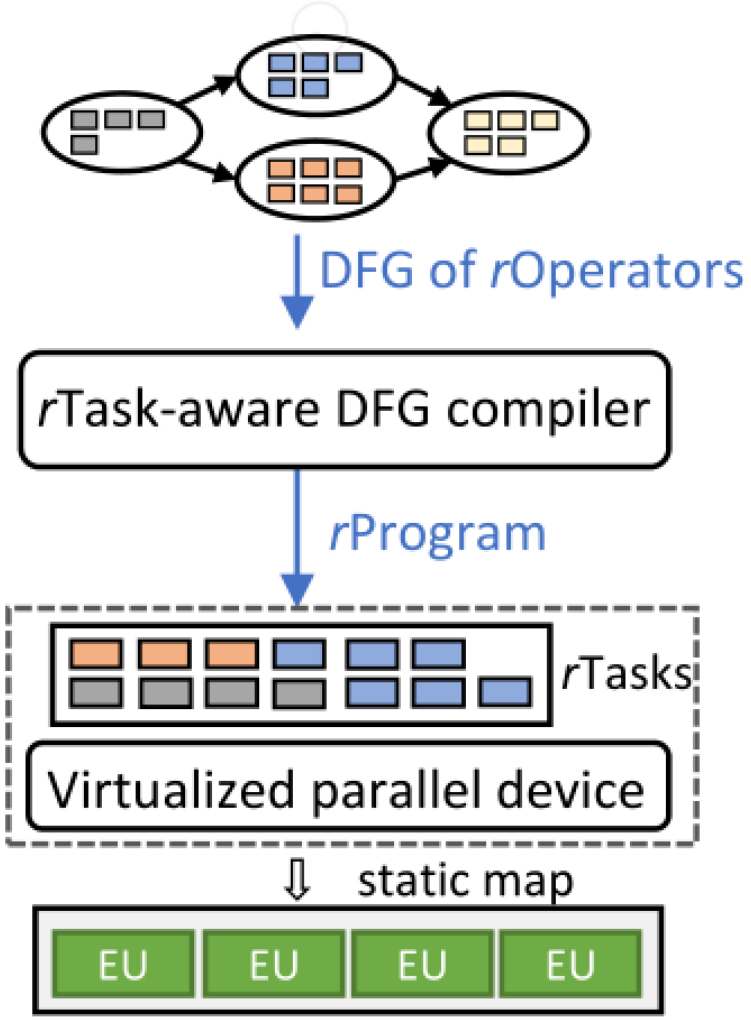
RAMMER：统一算子内和算子间调度

- 挑战一：算子是不透明的函数，不会暴露细粒度的算子内并行性。
- 挑战二：加速器（例如GPU）不显示算子内调度的接口。
- 挑战三： 细粒度调度可能会导致更多的运行时开销。



现有DNN框架

RAMMER系统设计



Rammer算子解析为 rOperator 并将其分解为更小的调度单元 rTask,将底层的硬件抽象为由多个虚拟执行单元vEU组成的 vDevice。

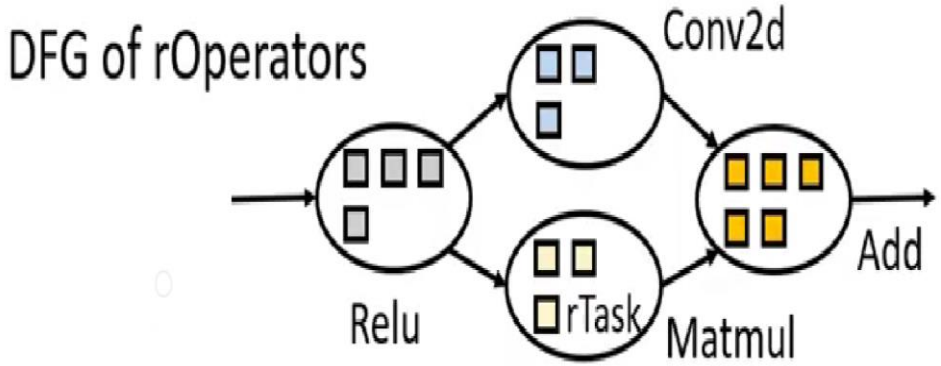
Rammer将DFG编译为一个称为rProgram的静态执行计划,rProgram包含一系列的rtasks,并通过vDevice抽象将其映射到硬件,在这套新的抽象下,用户可以通过更细粒度的 rTask 将数据流图调度到多个 vEU 之上。

整个调度方案在编译期生成并“静态”映射到硬件计算单元上,因此可以天然地消除掉许多原本存在的调度开销。

解决挑战一

rOperators与rTasks

- rTasks是EU上的最小计算单元
- rOperator是为一组独立的、同质的rTask的集合
- 这种抽象公开了算子内的并行性



解决挑战二

vEU和vDevice

- RAMMER将硬件加速器抽象为软件管理的虚拟设备，称为虚拟化并行设备vDevice

- vDevice还提供多个并行虚拟执行单元vEU，每个单元可以独立执行rTask

先进编译实验室
Advanced Compiler

解决挑战三

rProgram

在编译时生成执行计划（rProgram）

- 机制：调度接口和分析器
  两个调度接口，Append和Wait；分析器提供三种类型的信息。

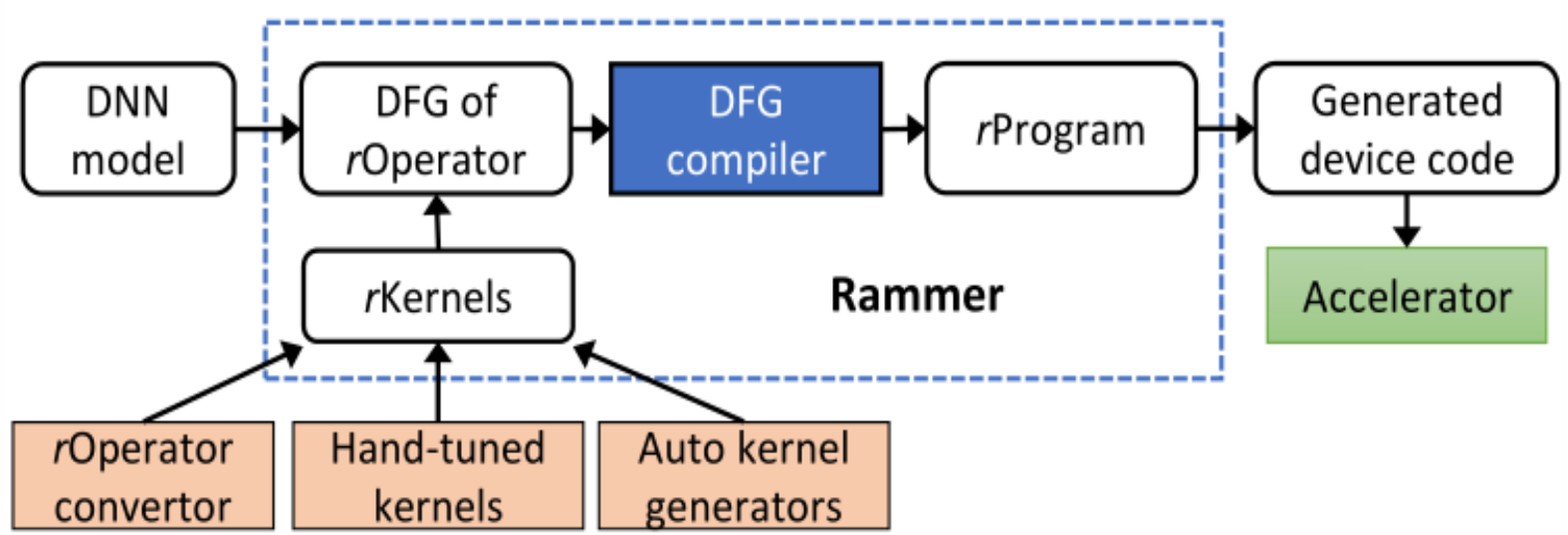
- 策略：Wavefront Scheduling Policy
  rOperator的实现称为rKernel，它实现了具体的rTask计算逻辑，并决定了rTask的总数。Rammer中为每一个roperator都实现了一种或多种rkernels。有的kernel计算很块（fast），有的kernel使用资源较少（resource-efficient）。Rammer默认使用fastest kernels；在资源比较紧张、可以占满所有EU的时候，使用resource-efficient kernels。

Algorithm 1: Wavefront Scheduling Policy

```
Data: G: DFG of rOperator, D: vDevice
Result: Plans: rPrograms
Function Schedule(G, D):
    P_curr = {};
    for W = Wavefront(G) do
        P_1 = ScheduleWave(W, P_curr, D);
        P_2 = ScheduleWave(W, {}, D);
        if time(P_1) ≤ time(P_curr) + time(P_2) then
            P_curr = P_1;
        else
            Plans.push_back(P_curr);
            P_curr = P_2;
    return Plans;
Function ScheduleWave(W, P, D):
    SelectRKernels(W, P);
    for op ∈ W do
        for r ∈ op.rTasks do
            vEU = SelectvEU(op, P, D);
            P.Wait(r, Predecessor(op).rTasks);
            P.Append(r, vEU);
    return P;
```

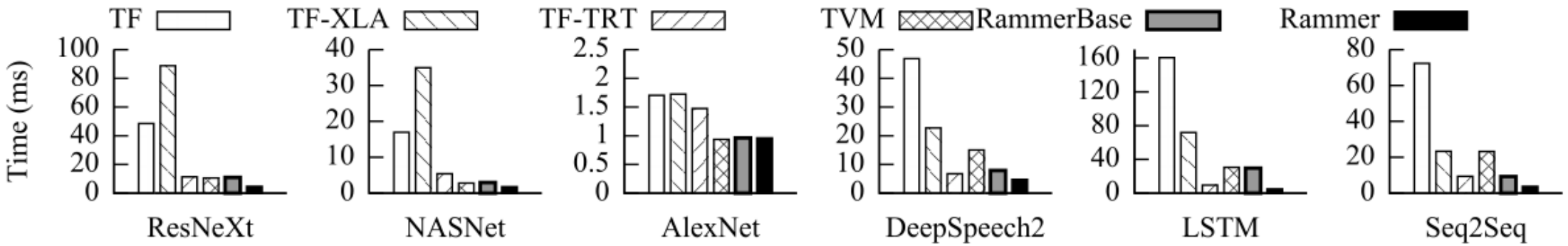
先进编译实验室
Advanced Compiler

RAMMER的总体工作流程



RAMMER首先将输入模型转换为rOperators的DFG，在DFG基础上进行常见的图优化。对于来自优化后的DFG中的每个rOperator，RAMMER从不同的源加载一个或多个版本的rKernel实现。RAMMER编译器然后将DFG划分为子图并将每个子图编译为rProgram。作为输出，每个rProgram被进一步生成为在加速器上运行的设备代码。

性能评估



NVIDIA V100 GPU上批量大小为1的端到端模型推断时间。

参考文献

Rammer: Enabling Holistic Deep Learning Compiler Optimizations with rTasks

论文链接:
https://www.usenix.org/conference/osdi20/presentation/ma

代码地址:
https://github.com/microsoft/nnfusion/