先进编译实验室
Advanced Compiler

AStitch：机器学习访存密集计算编译优化框架

嘉宾：赵薇

先进编译实验室
Advanced Compiler

AStitch: Enabling a New Multi-dimensional Optimization Space for Memory-Intensive ML Training and Inference on Modern SIMT Architectures

Zhen Zheng[*]
Alibaba Group
China

Xuanda Yang
Alibaba Group
China

Pengzhan Zhao
Alibaba Group
China

Guoping Long
Alibaba Group
China

Kai Zhu
Alibaba Group
China

Feiwen Zhu
Alibaba Group
China

Wenyi Zhao
Alibaba Group
China

Xiaoyong Liu
Alibaba Group
China

Jun Yang
Alibaba Group
China

Jidong Zhai
Tsinghua University
China

Shuaiwen Leon Song
University of Sydney
Australia

Wei Lin
Alibaba Group
China

ABSTRACT

This work reveals that memory-intensive computation is a rising performance-critical factor in recent machine learning models. Due to a unique set of new challenges, existing ML optimizing compilers cannot perform efficient fusion under complex two-level dependencies combined with just-in-time demand. They face the dilemma of either performing costly fusion due to heavy redundant computation, or skipping fusion which results in massive number of kernels. Furthermore, they often suffer from low parallelism due to the lack of support for real-world production workloads with irregular tensor shapes. To address these rising challenges, we propose *AStitch*, a machine learning optimizing compiler that opens a new multi-dimensional optimization space for memory-intensive ML computations. It systematically abstracts four operator-stitching schemes while considering multi-dimensional optimization objectives, tackles complex computation graph dependencies with novel hierarchical data reuse, and efficiently processes various tensor shapes via adaptive thread mapping. Finally, *AStitch* provides just-in-time support incorporating our proposed optimizations for both ML training and inference. Although *AStitch* serves as a stand-alone compiler engine that is portable to any version of TensorFlow, its basic ideas can be generally applied to other ML frameworks and optimization compilers. Experimental results show that *AStitch* can achieve an average of 1.84× speedup (up to 2.73×) over the state-of-the-art Google's XLA solution across five production workloads. We also deploy *AStitch* onto a production cluster for ML workloads with thousands of GPUs. The system has been in operation for more than 10 months and saves about 20,000 GPU hours for 70,000 tasks per week.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Parallel computing methodologies**.

KEYWORDS

Machine Learning, Memory-Intensive Computation, Compiler Optimization, Fusion

ACM Reference Format:
Zhen Zheng, Xuanda Yang, Pengzhan Zhao, Guoping Long, Kai Zhu, Feiwen Zhu, Wenyi Zhao, Xiaoyong Liu, Jun Yang, Jidong Zhai, Shuaiwen Leon Song, and Wei Lin. 2022. AStitch: Enabling a New Multi-dimensional Optimization Space for Memory-Intensive ML Training and Inference on Modern SIMT Architectures. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3503222.3507723

1 INTRODUCTION

Machine learning models usually consist of two types of operations: *compute-intensive* operations and *memory-intensive* operations. Compute-intensive operations are typically composed of heavy computation kernels (e.g., GEMM/GEMV and Convolution), while memory-intensive operations are often bounded by memory bandwidth (e.g., element-wise and reduction operations). Many recent works [14, 16, 18, 41, 43, 55] have made significant efforts on optimizing compute-intensive operations since they dominate the execution of some DNN workloads, especially in the domains of computer vision (CV, such as image classification [29], segmentation[27]

[*]Email: james.zz@alibaba-inc.com

ASPLOS '22, February 28 – March 4, 2022, Lausanne, Switzerland
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9205-1/22/02...$15.00
https://doi.org/10.1145/3503222.3507723

Advanced Compiler

FusionStitching: Boosting Memory Intensive Computations for Deep Learning Workloads

Zhen Zheng, Pengzhan Zhao, Guoping Long, Feiwen Zhu, Kai Zhu,
Wenyi Zhao, Lansong Diao, Jun Yang, Wei Lin
Alibaba Group
{james.zz, pengzhan.zpz, guopinglong.lgp, feiwen.zfw, tashuang.zk,
kevin.zwy, lansong.dls, muzhuo.yj, weilin.lw}@alibaba-inc.com

Abstract

We show in this work that memory intensive computations can result in severe performance problems due to off-chip memory access and CPU-GPU context switch overheads in a wide range of deep learning models. For this problem, current just-in-time kernel fusion and code generation techniques have limitations, such as kernel schedule incompatibilities and rough fusion plan exploration strategies. We propose *FusionStitching*, a Deep Learning compiler capable of fusing memory intensive operators, with varied data dependencies and non-homogeneous parallelism, into large GPU kernels to reduce global memory access and operation scheduling overhead automatically. *FusionStitching* explores large fusion spaces to decide optimal fusion plans with considerations of memory access costs, kernel calls and resource usage constraints. We thoroughly study the schemes to stitch operators together for complex scenarios. *FusionStitching* tunes the optimal stitching scheme just-in-time with a domain-specific cost model efficiently. Experimental results show that *FusionStitching* can reach up to 2.78× speedup compared to TensorFlow and current state-of-the-art. Besides these experimental results, we integrated our approach into a compiler product and deployed it onto a production cluster for AI workloads with thousands of GPUs. The system has been in operation for more than 4 months and saves 7,000 GPU hours on average for approximately 30,000 tasks per month.

Keywords: deep learning, kernel fusion, code generation

1 Introduction

Recent years have witnessed a surge of industry scale applications of deep learning models, ranging from images/videos, text/NLP, to billion scale search and recommendation systems[50]. Such workloads are typically expressed as computation graphs, and mapped to hardware through domain specific frameworks (TensorFlow[2], PyTorch[1], MXNet[11], etc). Given the flexibility and expressiveness of modern execution frameworks, there are still challenges regarding to transforming high level computation graphs into efficient kernels to maximize the underlying hardware execution efficiency.

Many current research works mainly focus on dense tensor computations (GEMM and convolution)[6, 12, 34, 39, 47] as dense computations dominate the execution time for many DNN workloads (like CNN[20, 38, 42]). However, recent advancement of the deep learning domain has resulted in many novel model structures in which memory intensive patterns occupies a large proportion of time. (In this paper, we refer to GEMM and convolution as **compute intensive** op, and other operators as **memory intensive** ops, such as element wise[43], transpose[45] and reduction[44]). In addition, the amount of memory intensive operators in modern machine learning models can be very large, causing notable GPU kernel launch and framework scheduling overhead. Table 2 contains the collected metrics of various models with TensorFlow implementation. The execution time of memory intensive ops can be more than that of compute intensive ops in some cases, and the kernel calls can be up to 10,406. For these workloads, optimizing compute intensive ops alone is inadequate to unlock the full performance potential.

Existing human-crafted computation libraries, such as cuDNN/cuBLAS, handle compute intensive ops as the pattern of these ops are usually stable. While it is not feasible to build library for flexible and fast-changing memory intensive patterns. Some code generation frameworks, like TVM[12], mainly focus on tuning compute intensive ops and do not address memory intensive ones specifically.

A common approach to address memory intensive patterns is computation fusion, a technique to fuse multiple ops into a single kernel to reduce off-chip memory accesses. Prior works have explored the basic idea in AI workloads[8, 12, 28], database[51], image processing[4, 33, 34], and HPC applications[27, 49]. However, how to fuse kernels, with unpredictable varied dependencies and non-homogeneous parallelism, just-in-time (JIT) efficiently is still an open problem.

Existing JIT kernel fusion techniques use simple and straightforward strategy to explore the fusion possibilities and thus lose optimization potential. As for memory intensive ops, the rapidly evolving AI models introduce diverse and complex combination patterns. The existing works lack the ability to fuse and optimize complex patterns with irregular dimension changing (due to varied shapes and layouts of tensors) and complex inter-thread dependencies (like reduction). This

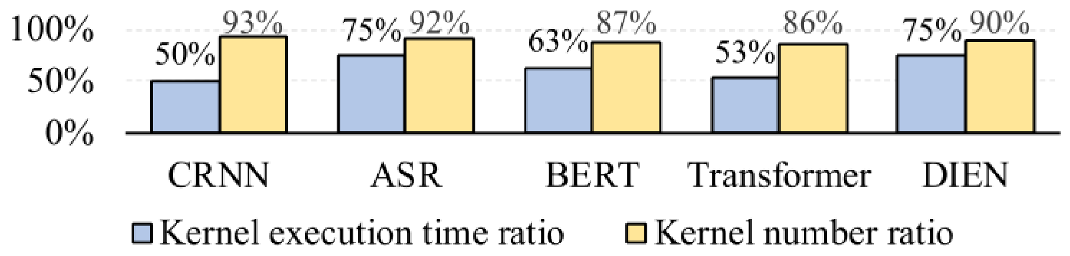
Preprint. Under review, 2020
2020.

目录

背景与挑战

背景

◆ 随着算法、硬件以及深度学习系统生态的发展，新出现的模型的性能瓶颈更多地表现在访存密集型算子上。业界主流编译器XLA最早开始关注访存密集型计算，但由于XLA采用kernel fusion的优化方法，没有利用shared memory来支持算子之间的数据传输，导致fusion粒度大大降低。



挑战

◆ 现有ML编译器不能够有效地融合带有两级依赖的运算结构。有些编译器以大量重计算为代价进行融合，有些直接放弃融合算子，也即是Fusion问题。

◆ 现有ML编译器对不规则张量形状的考虑不充分，导致生成的代码并行度低，也即是并行问题。

Astitch中的解决方案

针对Fusion问题：

AStitch从两级依赖关系出发，同时考虑硬件存储层次和并行度，为算子之间传递数据选择合适的存储媒介和并行策略，提出了stitch缝合技术，并且抽象出四种stitch策略。

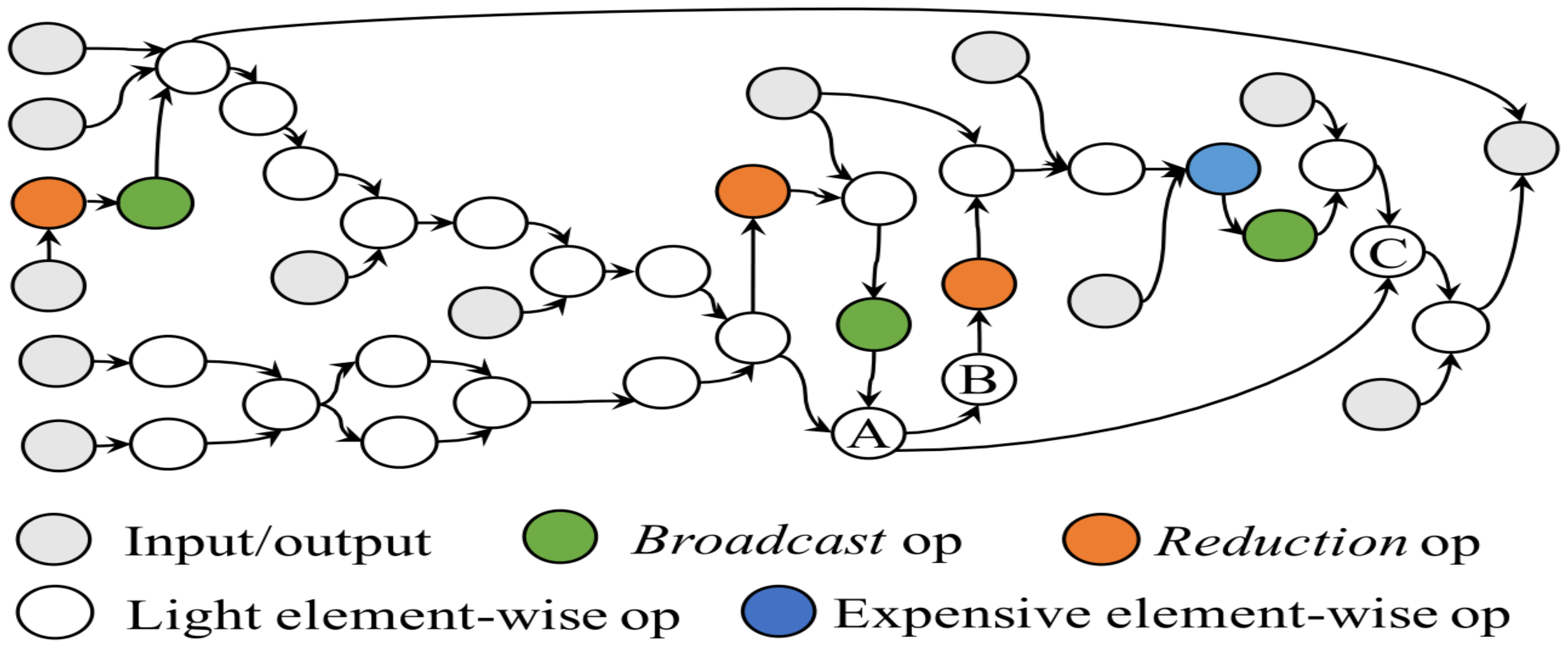
针对并行问题：

Astitch提出自适应线程映射技术，它的基本思想是，根据张量形状的大小特征，进行Task Packing和Task Splitting，以得到合适的CUDA thread block大小和数量。

Astitch中的Stitch策略

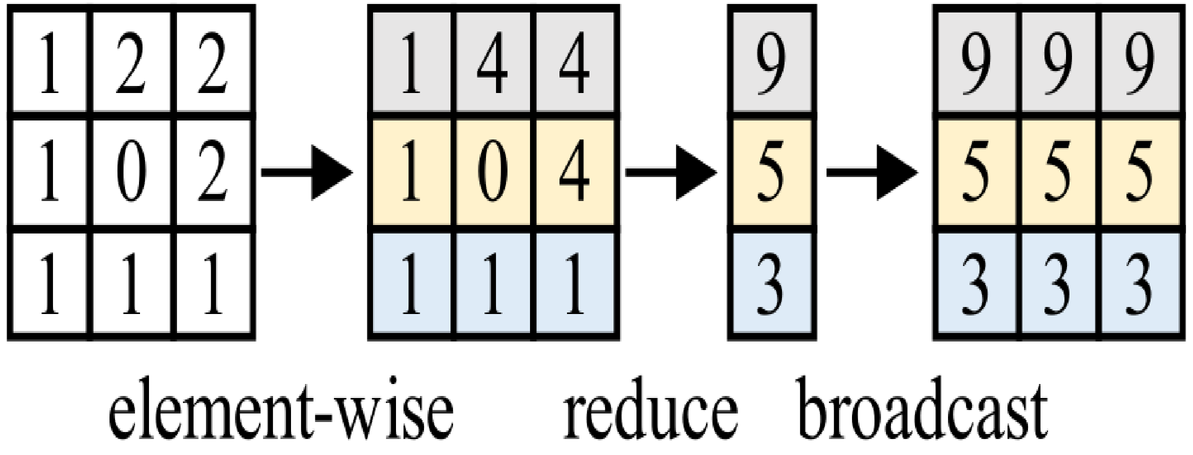
访存密集型子图通常由数十个甚至数百个算子组成，存在着两个级别的依赖关系：Operator-level 和Element-level。

◆ operator level依赖关系描述子图中表示的算子连接，例如，算子B和C依赖于中的算子A。



Astitch中的Stitch策略

◆ element level依赖表示张量内元素之间的依赖性，例如输出元素9依赖于输入元素1,4,4.。



Atitch中的Stitch策略

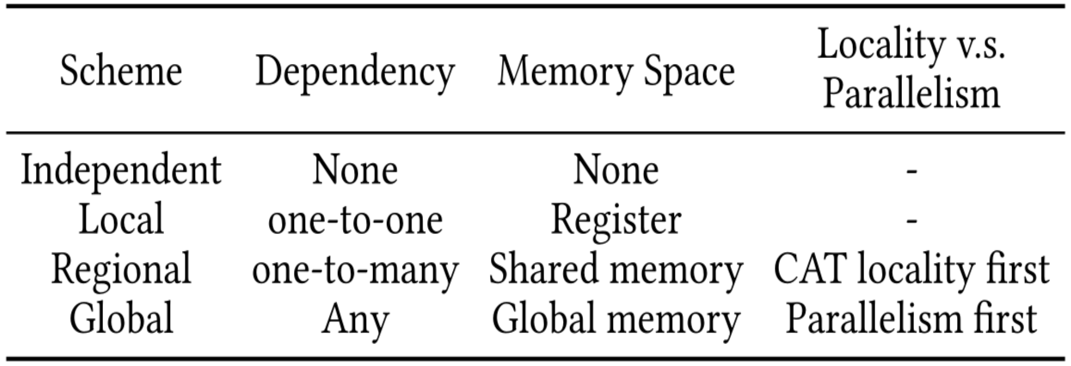
Astitch将stitch策略抽象为四种：无依赖（Independent）、本地的（Local）、区域的（Regional）和全局的（Global）

Independent策略应用于彼此独立的算子。

Local策略应用于具有元素级一对一依赖关系的算子并通过寄存器传输数据。

Regional策略应用于元素级一对多依赖关系的算子，将中间数据存储在GPU的shared memory中，支持GPU thread block locality。

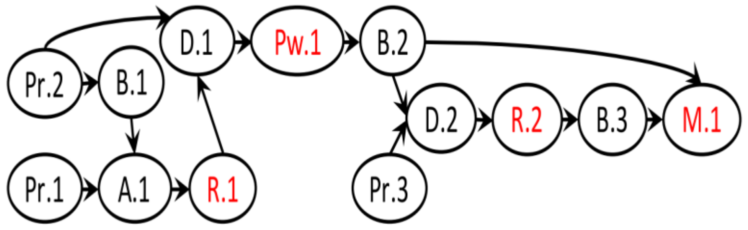
Global策略处理任何复杂的依赖关系，用global memory做数据传输，支持全局的locality。



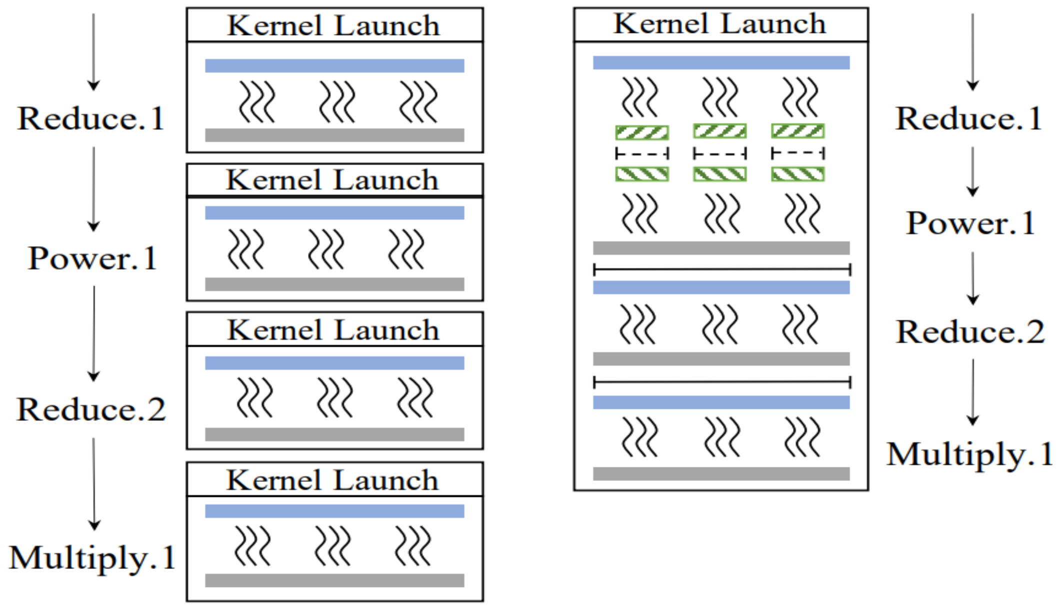
Scheme	Dependency	Memory Space	Locality v.s. Parallelism
Independent	None	None	-
Local	one-to-one	Register	-
Regional	one-to-many	Shared memory	CAT locality first
Global	Any	Global memory	Parallelism first

Atitch中的Stitch策略

图（a）是一个访存密集型子图，这里，算子的stitching策略是对于reduce.1采用regional策略；对于power.1和reduce.2采用global 策略；对于 multiply.1采用independent策略；对于其他算子采用local策略。图（b）和（c）展示了XLA和AStitch如何不同地形成针对此图生成kernel。



a) A memory-intensive sub-graph simplified from a real workload



b) Execution in XLA   c) Execution in *AStitch*

▬	Global memory load	▬	Global memory store		
▨	Shared memory load	▨	Shared memory store		
≋	Thread block	⊢--⊣	Thread block barrier	⊢—⊣	Globar-bar

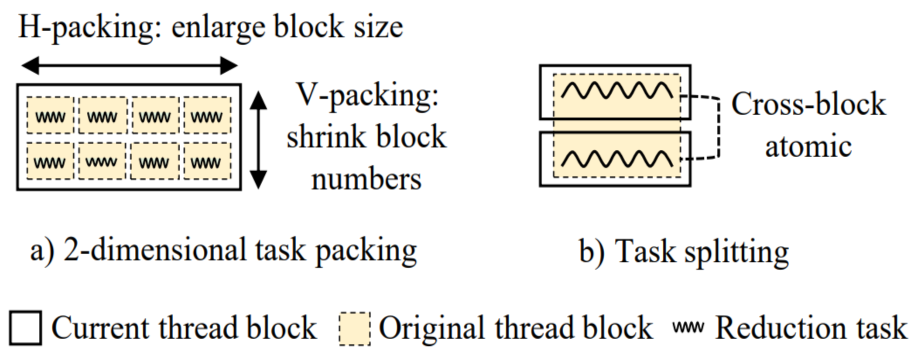
自适应线程映射

Task Packing

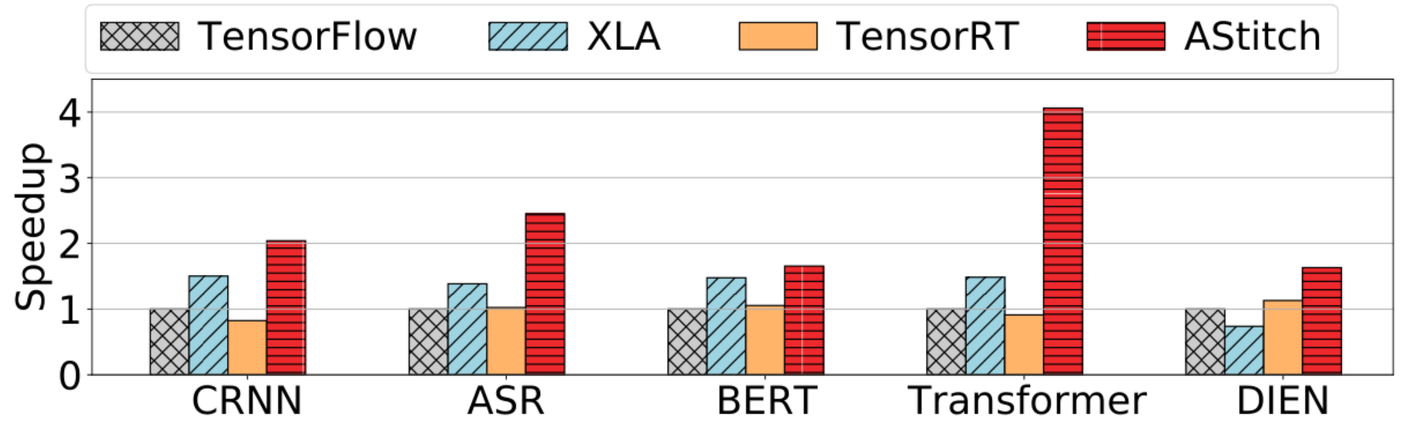
Task packing包括两个维度：水平(Horizontal)和垂直(Vertical) 。
Horizontal packing是将多个小线程块打包成一个大线程块，增加block自身的大小，也就是增加线程数。
Vertical packing是将多个线程块的任务打包成一个，以减少block的数量。
Task packing保证总的CUDA thread block数量不大于一次wave可以调度的最大数量，以满足global barrier的要求。

Task Splitting

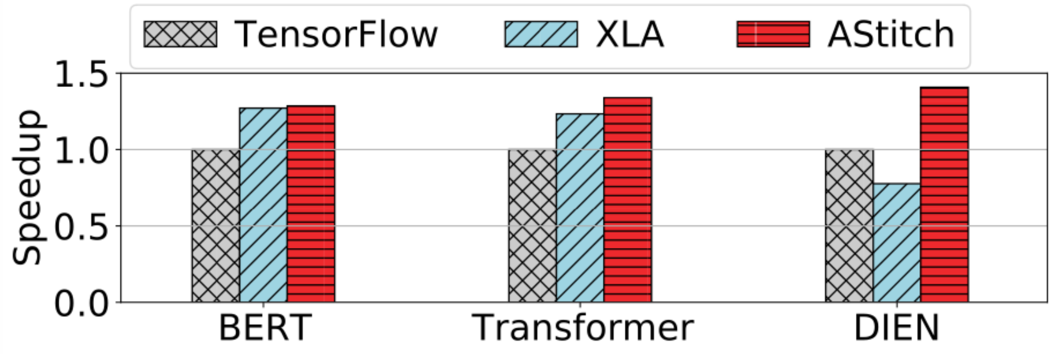
Task splitting是将一个线程块中的任务拆分为多个线程块，以防由于块数少而导致利用率不足。



性能评估



(a) Speedup of inference tasks.



(b) Speedup of training tasks.

总结

总结

Astitch针对现有ML编译器无法有效地融合带有两层依赖的运算结构和不规则张量形状所带来的并行性问题，提出了相应的Stitch融合策略和自适应线程映射技术，从两个层面的依赖关系出发，同时考虑硬件存储层次和并行度，为算子之间传递数据选择合适的存储媒介和并行策略，拥有强大的代码生成能力。

参考文献

AStitch: Enabling a New Multi-dimensional Optimization Space for Memory-Intensive ML Training and Inference on Modern SIMT Architectures