论文分享:面向特定领域加速器的计算图高效调度

Effectively Scheduling Computational Graphs of Deep Neural Networks toward Their Domain-Specific Accelerators

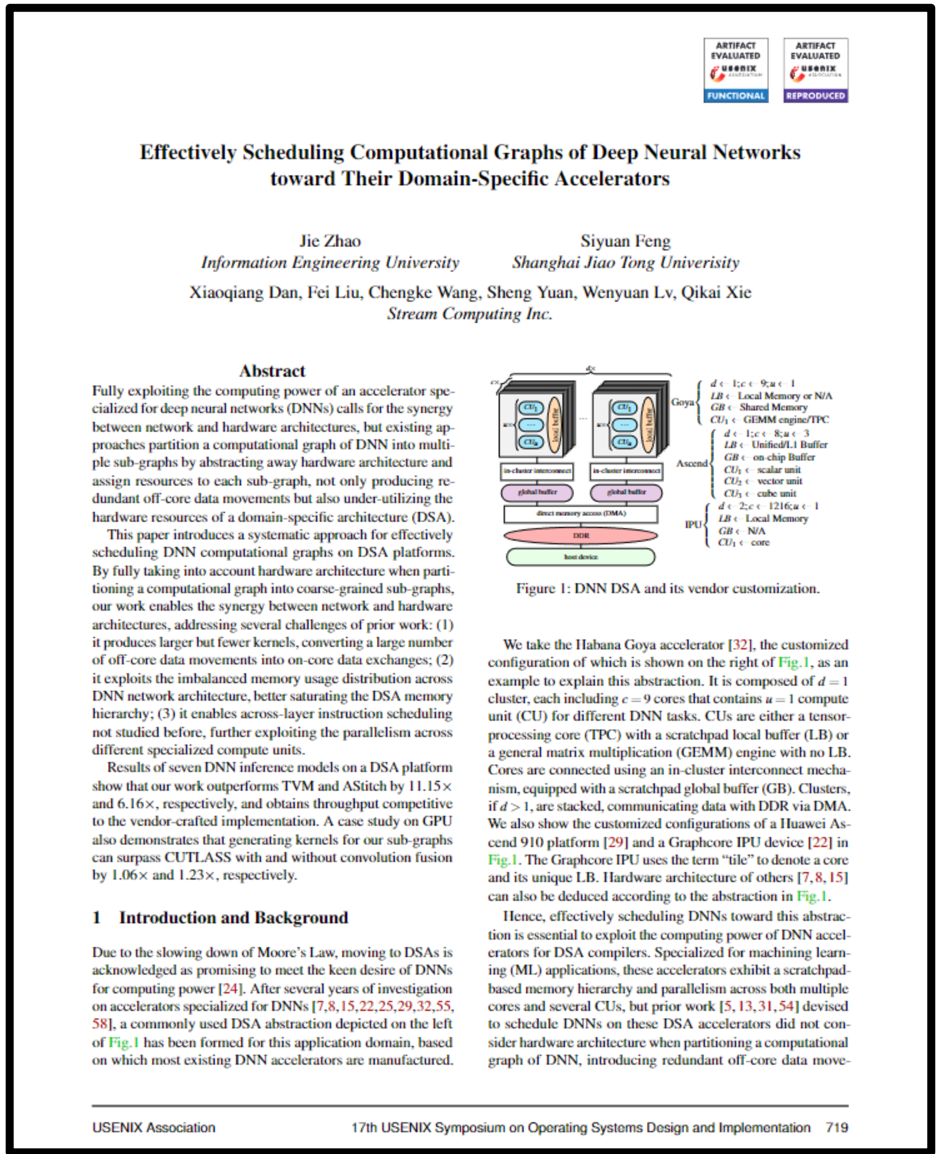
Jie Zhao
Information Engineering University

Siyuan Feng
Shanghai Jiao Tong Univerisity

Xiaoqiang Dan, Fei Liu, Chengke Wang, Sheng Yuan, Wenyuan Lv, Qikai Xie
Stream Computing Inc.

嘉宾:

论文提要



ARTIFACT EVALUATED usenix FUNCTIONAL
ARTIFACT EVALUATED usenix REPRODUCED

Effectively Scheduling Computational Graphs of Deep Neural Networks toward Their Domain-Specific Accelerators

Jie Zhao
Information Engineering University

Siyuan Feng
Shanghai Jiao Tong Univerisity

Xiaoqiang Dan, Fei Liu, Chengke Wang, Sheng Yuan, Wenyuan Lv, Qikai Xie
Stream Computing Inc.

Abstract

Fully exploiting the computing power of an accelerator specialized for deep neural networks (DNNs) calls for the synergy between network and hardware architectures, but existing approaches partition a computational graph of DNN into multiple sub-graphs by abstracting away hardware architecture and assign resources to each sub-graph, not only producing redundant off-core data movements but also under-utilizing the hardware resources of a domain-specific architecture (DSA).

This paper introduces a systematic approach for effectively scheduling DNN computational graphs on DSA platforms. By fully taking into account hardware architecture when partitioning a computational graph into coarse-grained sub-graphs, our work enables the synergy between network and hardware architectures, addressing several challenges of prior work: (1) it produces larger but fewer kernels, converting a large number of off-core data movements into on-core data exchanges; (2) it exploits the imbalanced memory usage distribution across DNN network architecture, better saturating the DSA memory hierarchy; (3) it enables across-layer instruction scheduling not studied before, further exploiting the parallelism across different specialized compute units.

Results of seven DNN inference models on a DSA platform show that our work outperforms TVM and AStitch by 11.15× and 6.16×, respectively, and obtains throughput competitive to the vendor-crafted implementation. A case study on GPU also demonstrates that generating kernels for our sub-graphs can surpass CUTLASS with and without convolution fusion by 1.06× and 1.23×, respectively.

1 Introduction and Background

Due to the slowing down of Moore's Law, moving to DSAs is acknowledged as promising to meet the keen desire of DNNs for computing power [24]. After several years of investigation on accelerators specialized for DNNs [7,8,15,22,25,29,32,55,58], a commonly used DSA abstraction depicted on the left of Fig.1 has been formed for this application domain, based on which most existing DNN accelerators are manufactured.

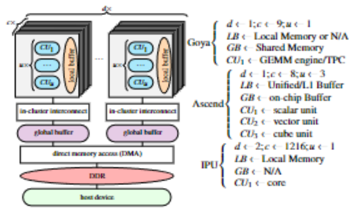

d×
c×
u×
CU_1
...
CU_u
local buffer
in-cluster interconnect
global buffer
direct memory access (DMA)
DDR
host device
Goya
$d \leftarrow 1; c \leftarrow 9; u \leftarrow 1$
$LB \leftarrow$ Local Memory or N/A
$GB \leftarrow$ Shared Memory
$CU_1 \leftarrow$ GEMM engine/TPC
Ascend
$d \leftarrow 1; c \leftarrow 8; u \leftarrow 3$
$LB \leftarrow$ Unified/L1 Buffer
$GB \leftarrow$ on-chip Buffer
$CU_1 \leftarrow$ scalar unit
$CU_2 \leftarrow$ vector unit
$CU_3 \leftarrow$ cube unit
IPU
$d \leftarrow 2; c \leftarrow 1216; u \leftarrow 1$
$LB \leftarrow$ Local Memory
$GB \leftarrow$ N/A
$CU_1 \leftarrow$ core

Figure 1: DNN DSA and its vendor customization.

We take the Habana Goya accelerator [32], the customized configuration of which is shown on the right of Fig.1, as an example to explain this abstraction. It is composed of $d = 1$ cluster, each including $c = 9$ cores that contains $u = 1$ compute unit (CU) for different DNN tasks. CUs are either a tensor-processing core (TPC) with a scratchpad local buffer (LB) or a general matrix multiplication (GEMM) engine with no LB. Cores are connected using an in-cluster interconnect mechanism, equipped with a scratchpad global buffer (GB). Clusters, if $d > 1$, are stacked, communicating data with DDR via DMA. We also show the customized configurations of a Huawei Ascend 910 platform [29] and a Graphcore IPU device [22] in Fig.1. The Graphcore IPU uses the term "tile" to denote a core and its unique LB. Hardware architecture of others [7, 8, 15] can also be deduced according to the abstraction in Fig.1.

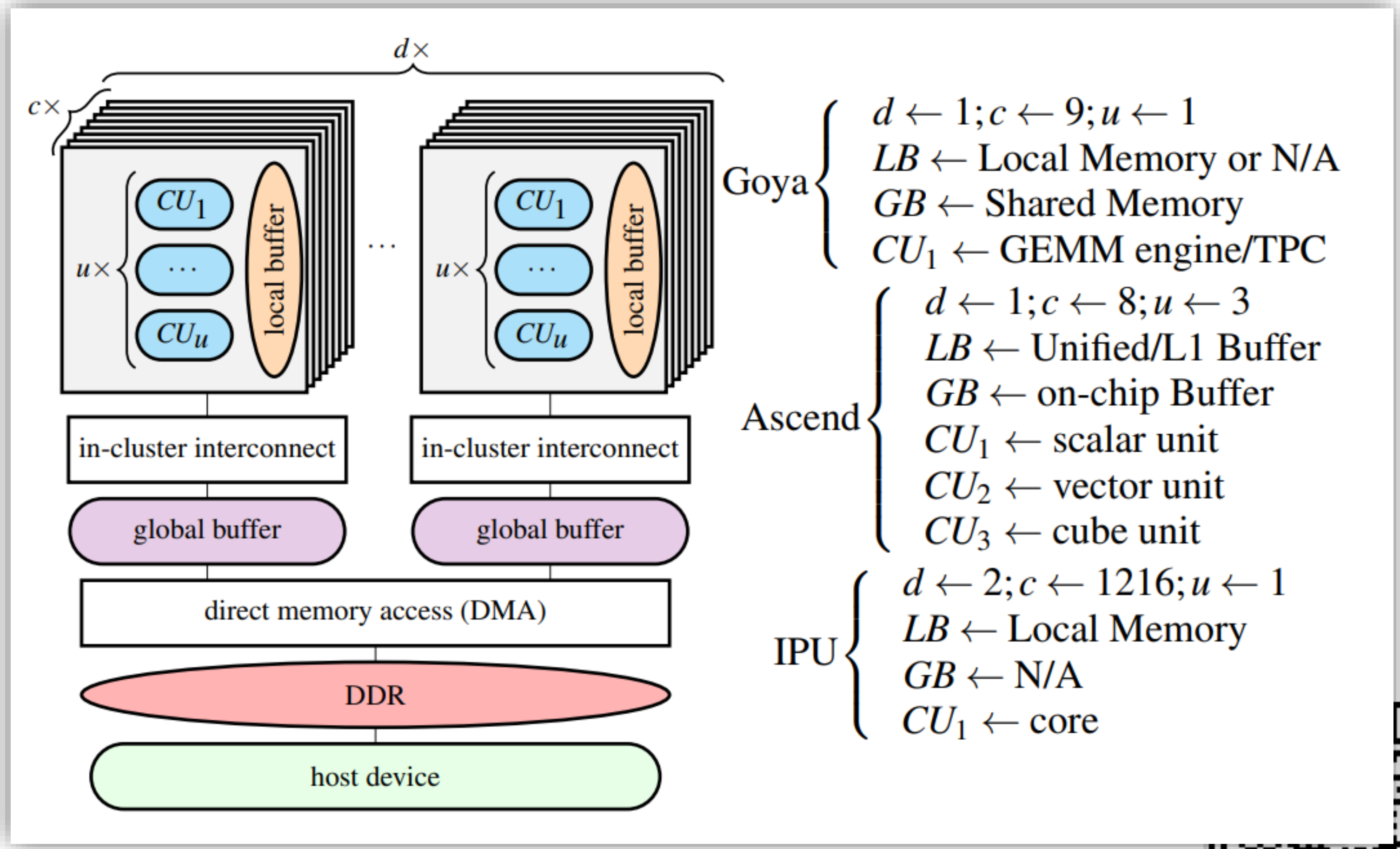
Hence, effectively scheduling DNNs toward this abstraction is essential to exploit the computing power of DNN accelerators for DSA compilers. Specialized for machining learning (ML) applications, these accelerators exhibit a scratchpad-based memory hierarchy and parallelism across both multiple cores and several CUs, but prior work [5, 13, 31, 54] devised to schedule DNNs on these DSA accelerators did not consider hardware architecture when partitioning a computational graph of DNN, introducing redundant off-core data move-

Effectively Scheduling Computational Graphs of Deep Neural Networks toward Their Domain-Specific Accelerators (OSDI '23)

- Introduction and Background
- Core Idea and Overview
- Scheduling Sub-graph Instances
- Kernel Generation for Sub-graph Instances
- Experimental Results
- Conclusion

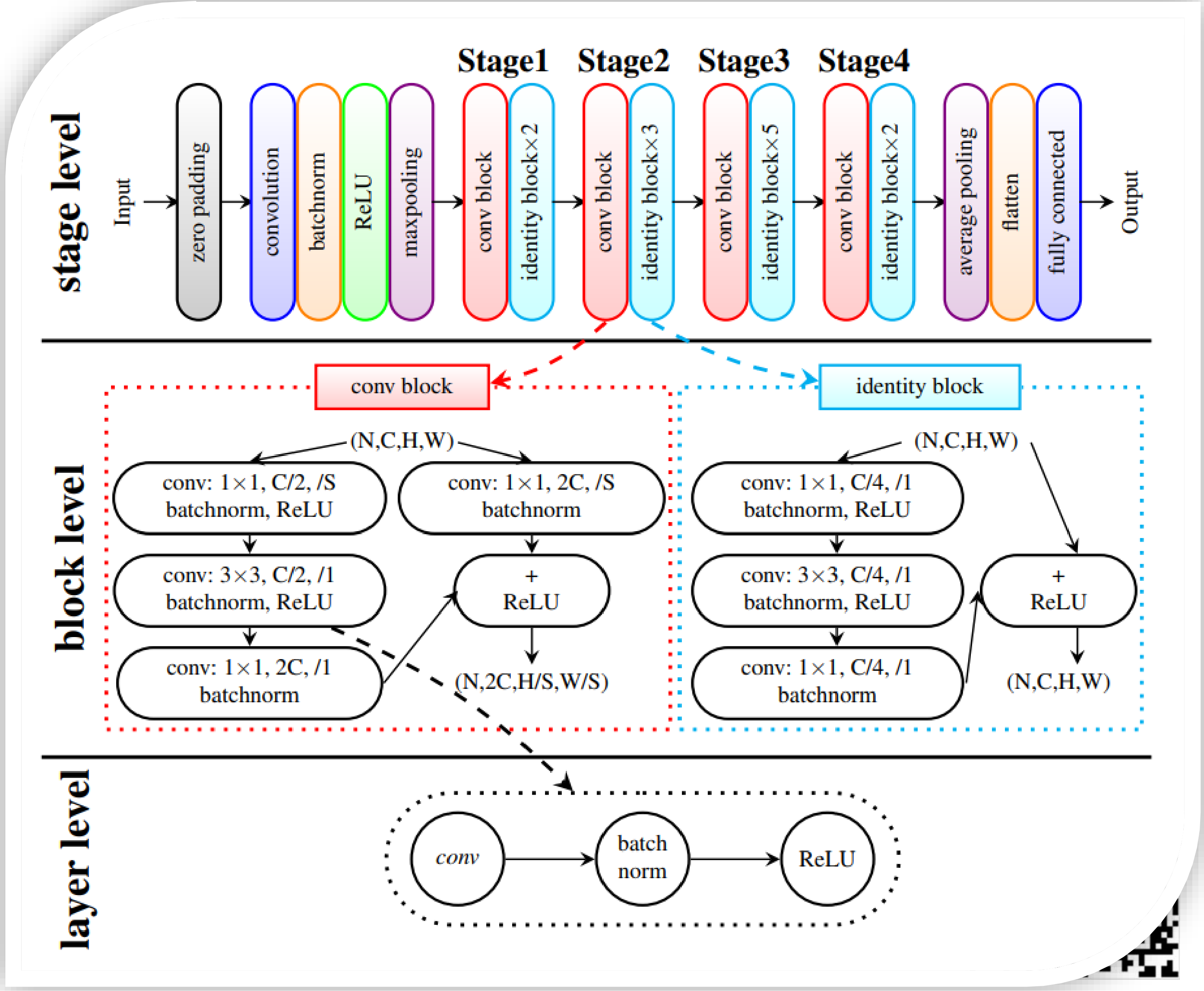
背景 > DNN DSA

- 摩尔定律的放缓 → DNN DSA 的发展
- DNN DSA 抽象
  - d：Cluster
  - c：Core
  - u：CU（Compute Unit）
  - LB
  - GB
  - DDR


$d\times$
$c\times$
$u\times$
CU_1
$\cdots$
CU_u
local buffer
in-cluster interconnect
global buffer
direct memory access (DMA)
DDR
host device
Goya
$d \leftarrow 1; c \leftarrow 9; u \leftarrow 1$
$LB \leftarrow$ Local Memory or N/A
$GB \leftarrow$ Shared Memory
$CU_1 \leftarrow$ GEMM engine/TPC
Ascend
$d \leftarrow 1; c \leftarrow 8; u \leftarrow 3$
$LB \leftarrow$ Unified/L1 Buffer
$GB \leftarrow$ on-chip Buffer
$CU_1 \leftarrow$ scalar unit
$CU_2 \leftarrow$ vector unit
$CU_3 \leftarrow$ cube unit
IPU
$d \leftarrow 2; c \leftarrow 1216; u \leftarrow 1$
$LB \leftarrow$ Local Memory
$GB \leftarrow$ N/A
$CU_1 \leftarrow$ core

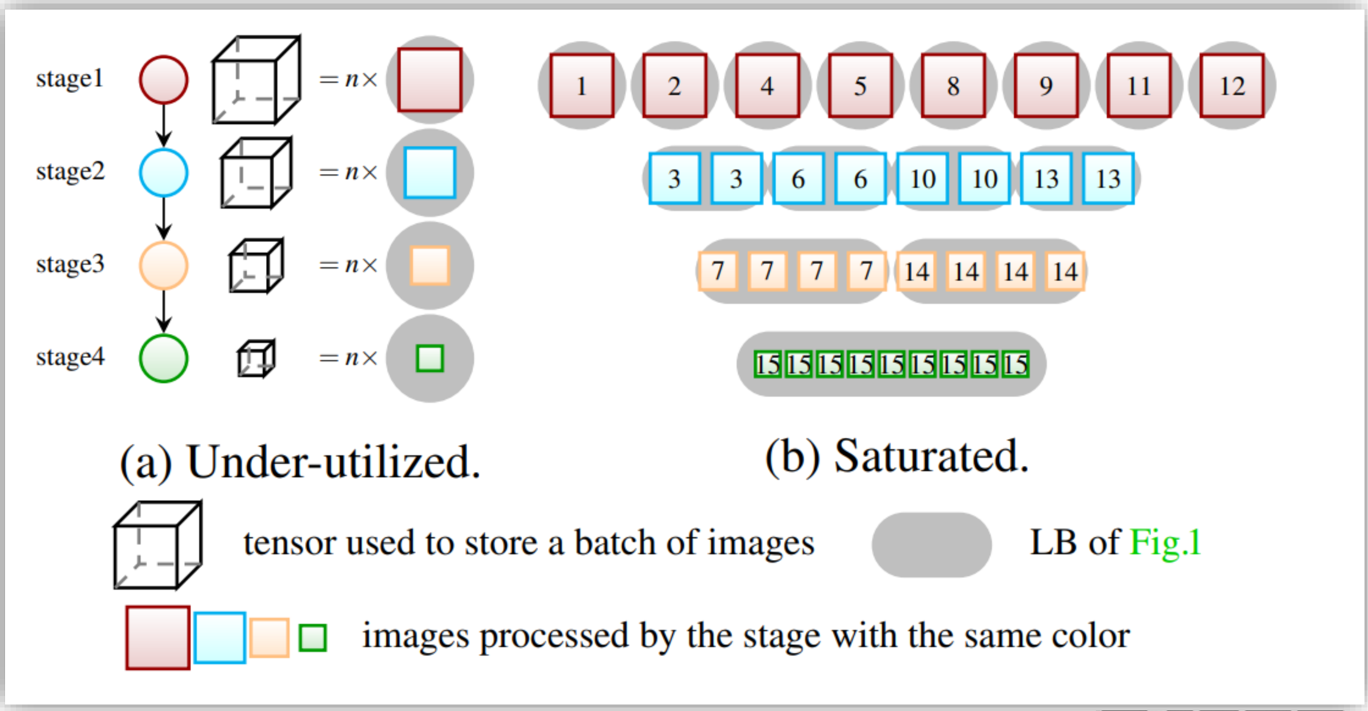
背景 > DNN Models

- DNN Models
  - Stage
  - Block
  - Layer
- 问题
  - Kernel间频繁的数据移动
  - 错失跨层指令调度的机会
  - 没有充分利用本地内存



设计

- 问题
  - Kernel间频繁的数据移动
  - 错失跨层指令调度的机会
  - 没有充分利用本地内存
- 解决方案
  - 最大限度地使输入张量驻留于本地内存
  - 子图划分后进行调度
  - 跨Core利用并行性并使本地内存饱和
- 详细方案—GraphTurbo
  - 调度子图实例
  - Kernel生成



设计 >调度子图实例

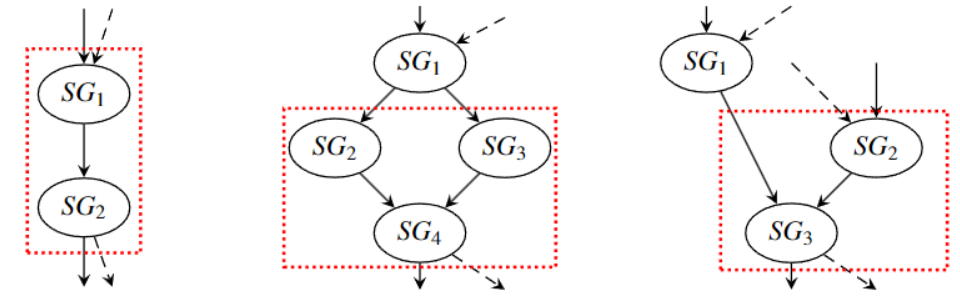
- 收集划分信息

  $SplitInfo = (split_d, n_d, f_d, d)$ Core数量 LB容量等硬件信息 输出张量 → 中间变量 → 输入张量

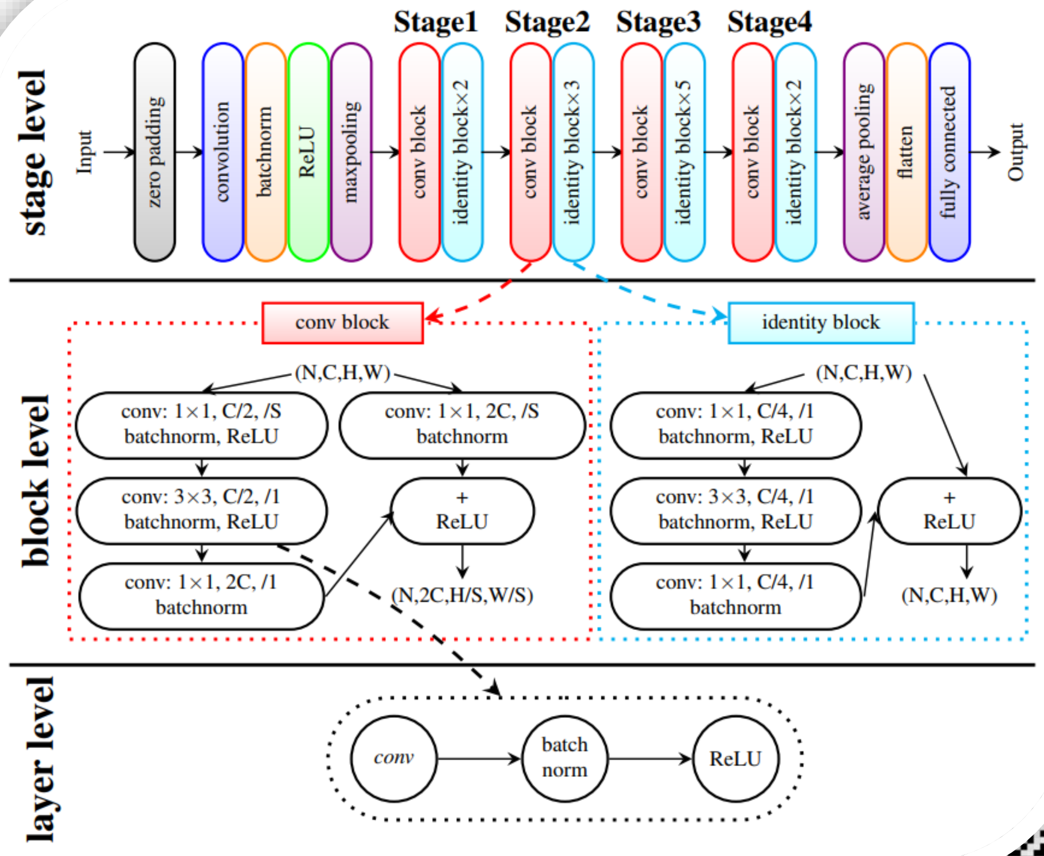
- 对子图进行分组

  拓扑排序

  三种分组模式：Straight、Diamond、Branch



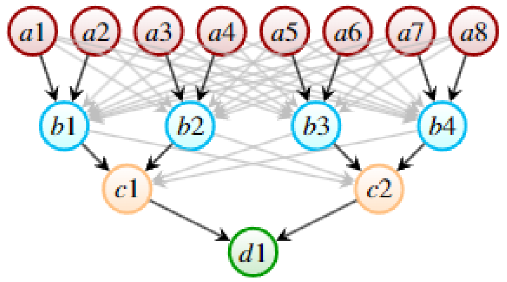
(a) Straight. (b) Diamond. (c) Branch.



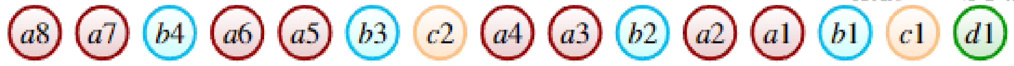
设计 >调度子图实例

- 收集划分信息
- 对子图进行分组
- 子图实例排序

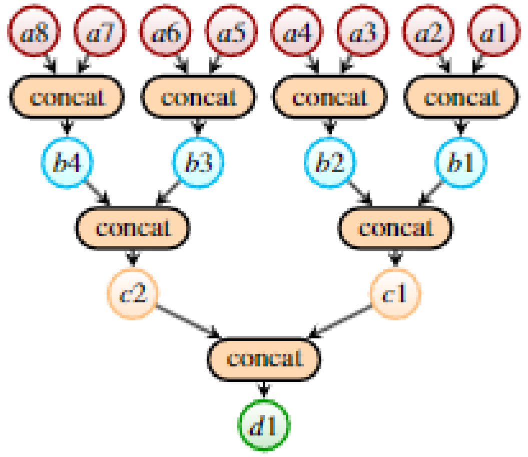
消除实例间冗余依赖



确定调度顺序

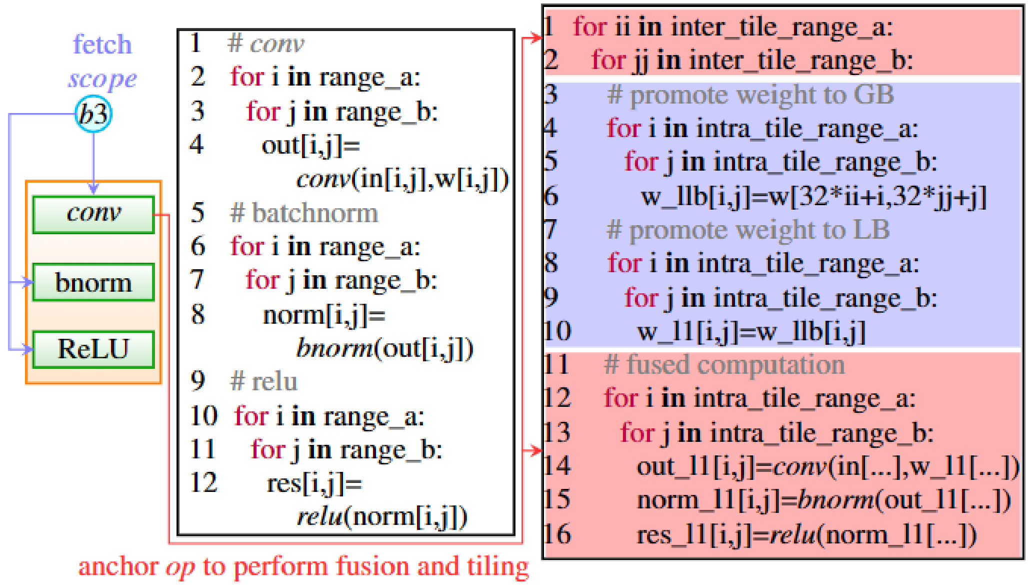


- 推断Core绑定和Buffer范围
- 连接实例的输出



设计 >Kernel生成

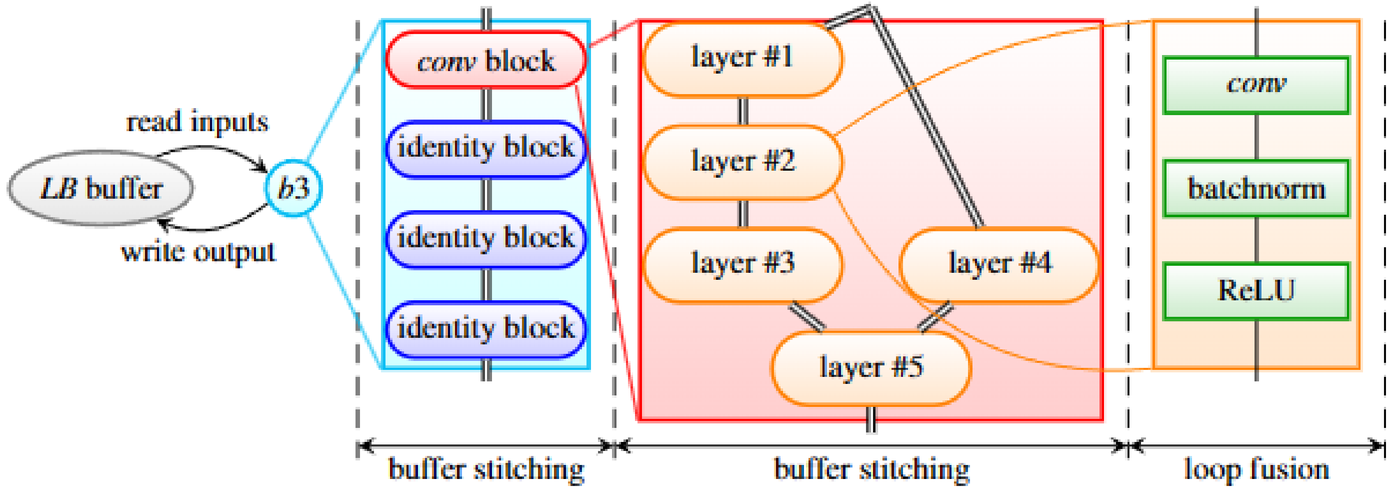
■ 层内循环融合



```
# conv
for i in range_a:
  for j in range_b:
    out[i,j]=
        conv(in[i,j],w[i,j])
# batchnorm
for i in range_a:
  for j in range_b:
    norm[i,j]=
        bnorm(out[i,j])
# relu
for i in range_a:
  for j in range_b:
    res[i,j]=
        relu(norm[i,j])
```

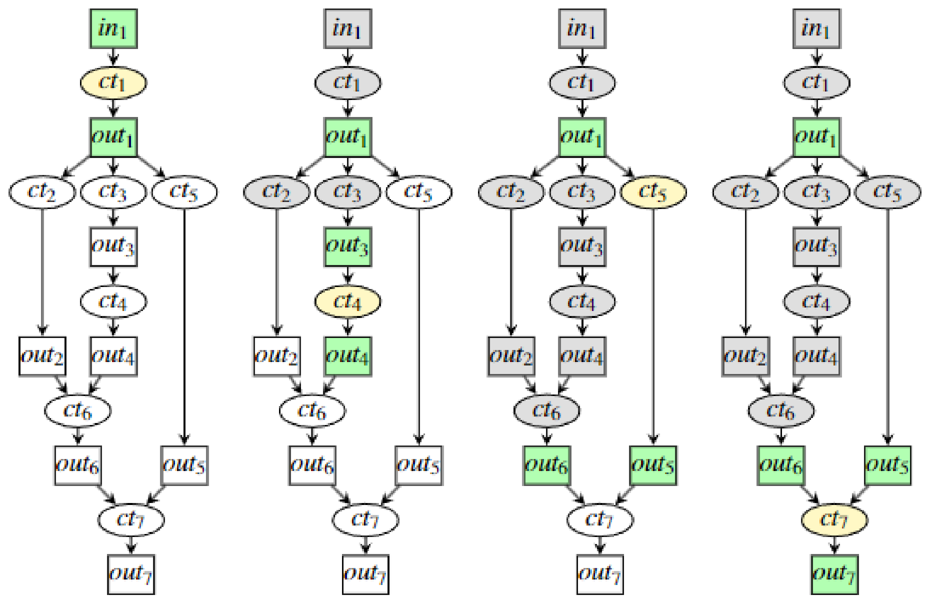
```
for ii in inter_tile_range_a:
  for jj in inter_tile_range_b:
    # promote weight to GB
    for i in intra_tile_range_a:
      for j in intra_tile_range_b:
        w_llb[i,j]=w[32*ii+i,32*jj+j]
    # promote weight to LB
    for i in intra_tile_range_a:
      for j in intra_tile_range_b:
        w_l1[i,j]=w_llb[i,j]
    # fused computation
    for i in intra_tile_range_a:
      for j in intra_tile_range_b:
        out_l1[i,j]=conv(in[...],w_l1[...])
        norm_l1[i,j]=bnorm(out_l1[...])
        res_l1[i,j]=relu(norm_l1[...])
```

■ 跨层/块Buffer缝合

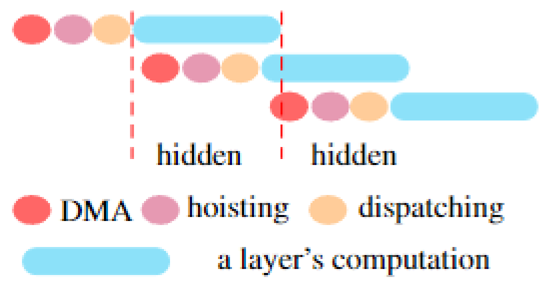


设计 >Kernel生成

- 层内循环融合
- 跨层/块Buffer缝合
- 内存分配和重用
- 跨层指令调度



(a) Execute ct_1. (b) Execute ct_4. (c) Execute ct_5. (d) Execute ct_7.



效果

- TVM

  在子图内进行算子融合，生成的Kernel通过DDR交换数据

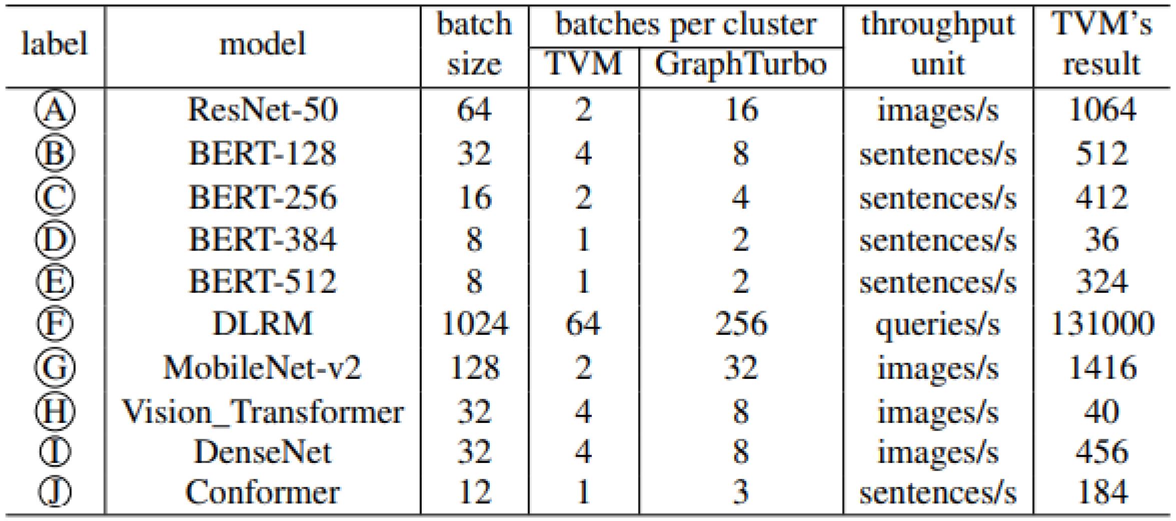
- Astitch

  没有对子图实例进行排序，也没有考虑计算密集型算子

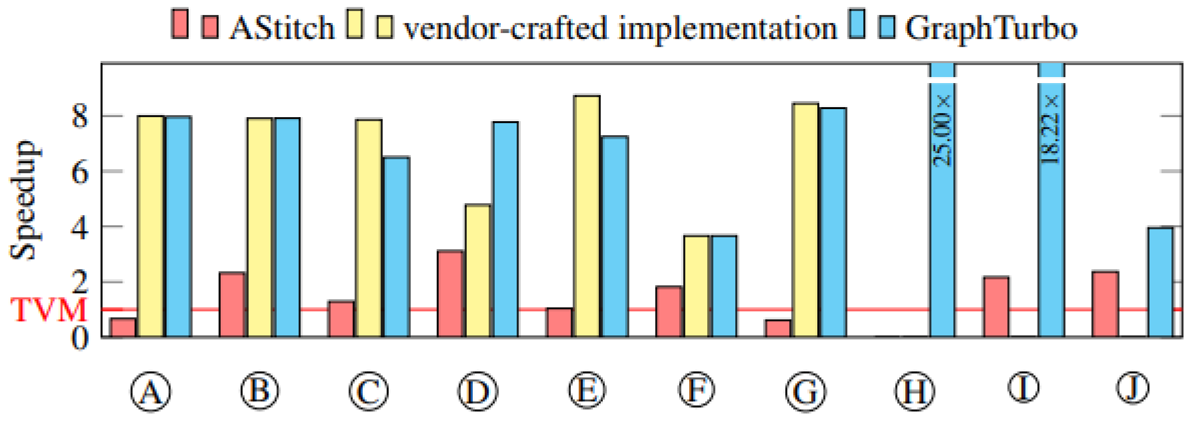
- GraphTurbo

  TVM 11.15× AStitch 6.16×

  供应商手工实现 1.04×



label	model	batch size	batches per cluster		throughput unit	TVM's result
			TVM	GraphTurbo		
Ⓐ	ResNet-50	64	2	16	images/s	1064
Ⓑ	BERT-128	32	4	8	sentences/s	512
Ⓒ	BERT-256	16	2	4	sentences/s	412
Ⓓ	BERT-384	8	1	2	sentences/s	36
Ⓔ	BERT-512	8	1	2	sentences/s	324
Ⓕ	DLRM	1024	64	256	queries/s	131000
Ⓖ	MobileNet-v2	128	2	32	images/s	1416
Ⓗ	Vision_Transformer	32	4	8	images/s	40
Ⓘ	DenseNet	32	4	8	images/s	456
Ⓙ	Conformer	12	1	3	sentences/s	184



参考资料

[1] Zhao J, Feng S, Dan X, et al. Effectively Scheduling Computational Graphs of Deep Neural Networks toward Their {Domain-Specific} Accelerators[C]//17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). 2023: 719-737.

[2]