IOS: **I**nter-**O**perator **S**cheduler for CNN Acceleration

Yaoyao Ding[1 2], Ligeng Zhu[3], Zhihao Jia[4],

Gennady Pekhimenko[1 2], Song Han[3]

1 UNIVERSITY OF TORONTO

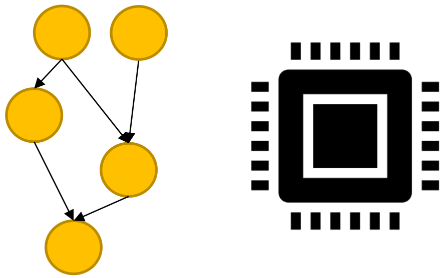
2 VECTOR INSTITUTE

3 MIT

4 Carnegie Mellon University

Executive Summary

Motivation

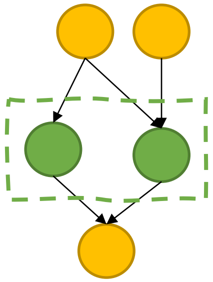


CNN Model GPU

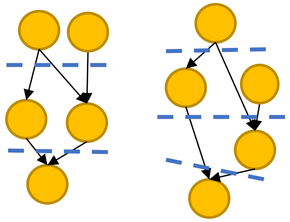
Sequential Execution

Under-utilization Problem

Inter-Operator Scheduler



Inter-Operator Parallelization



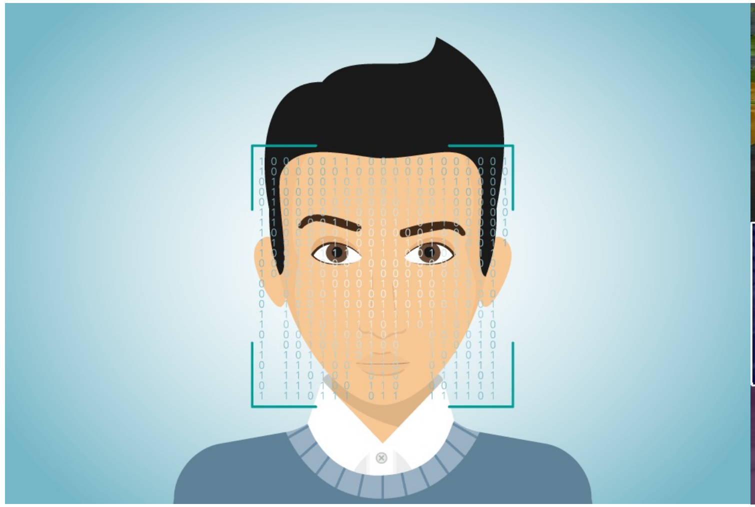
Dynamic Programming

Optimal Schedule

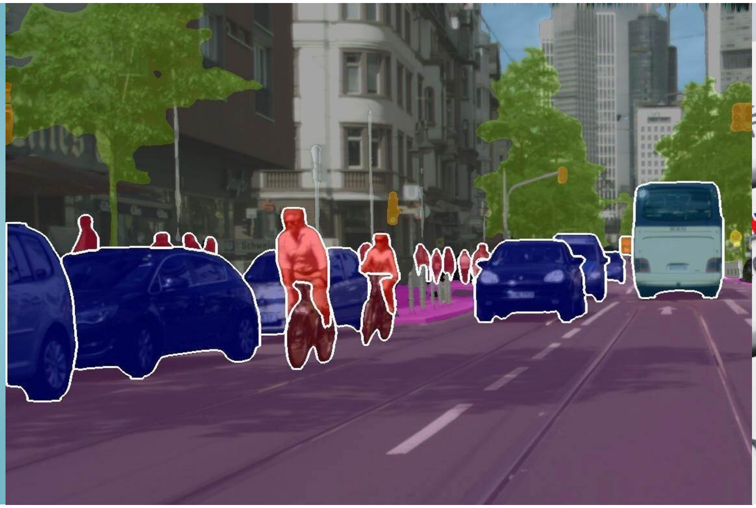
Inter-Op Schedules

1.1-1.5x speedup

Efficient Deployment of CNNs is Important



Face Recognition



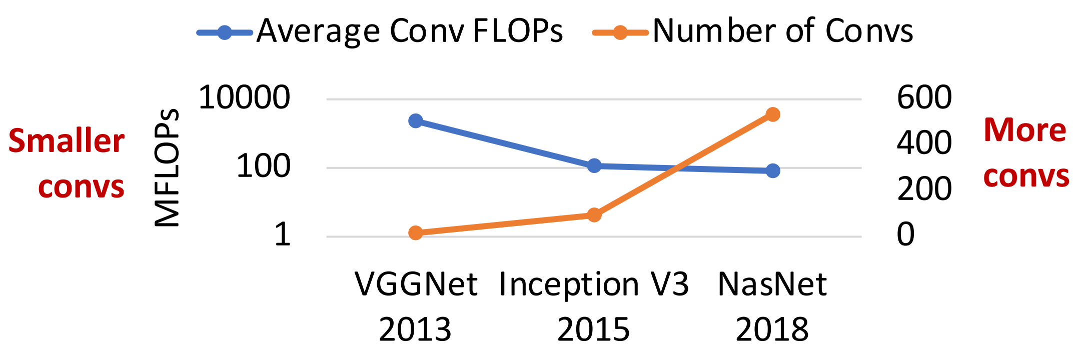
Self Driving

Language Translation

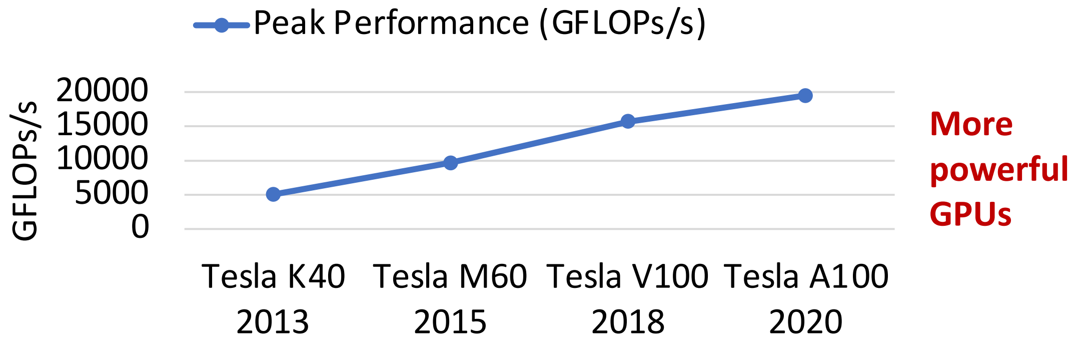
Is CNN inference in current DL libraries well utilizing underlying hardware?

Motivation for Inter-Operator Parallelization

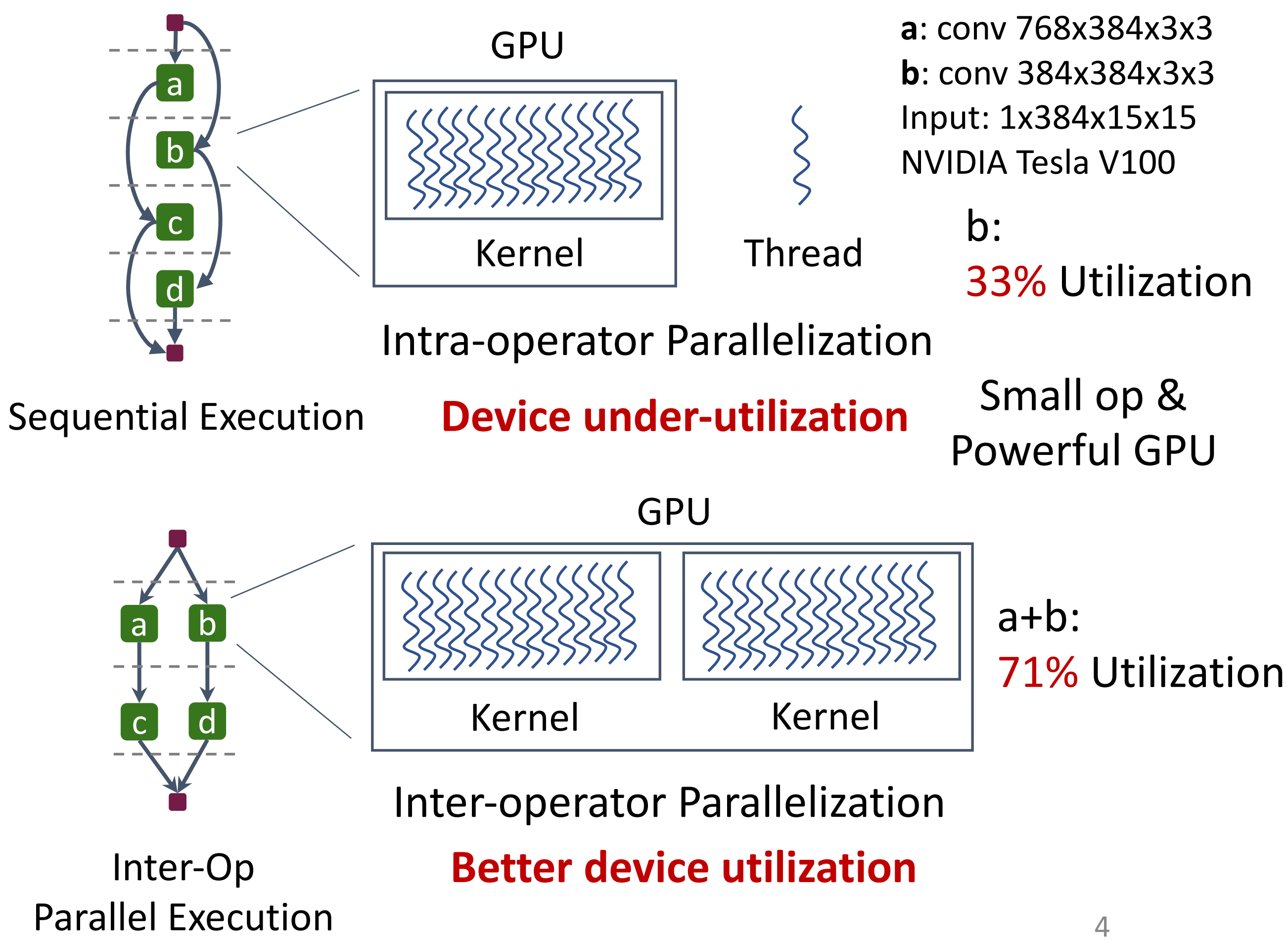
1. More small convs in CNN design.



2. GPU peak performance increased

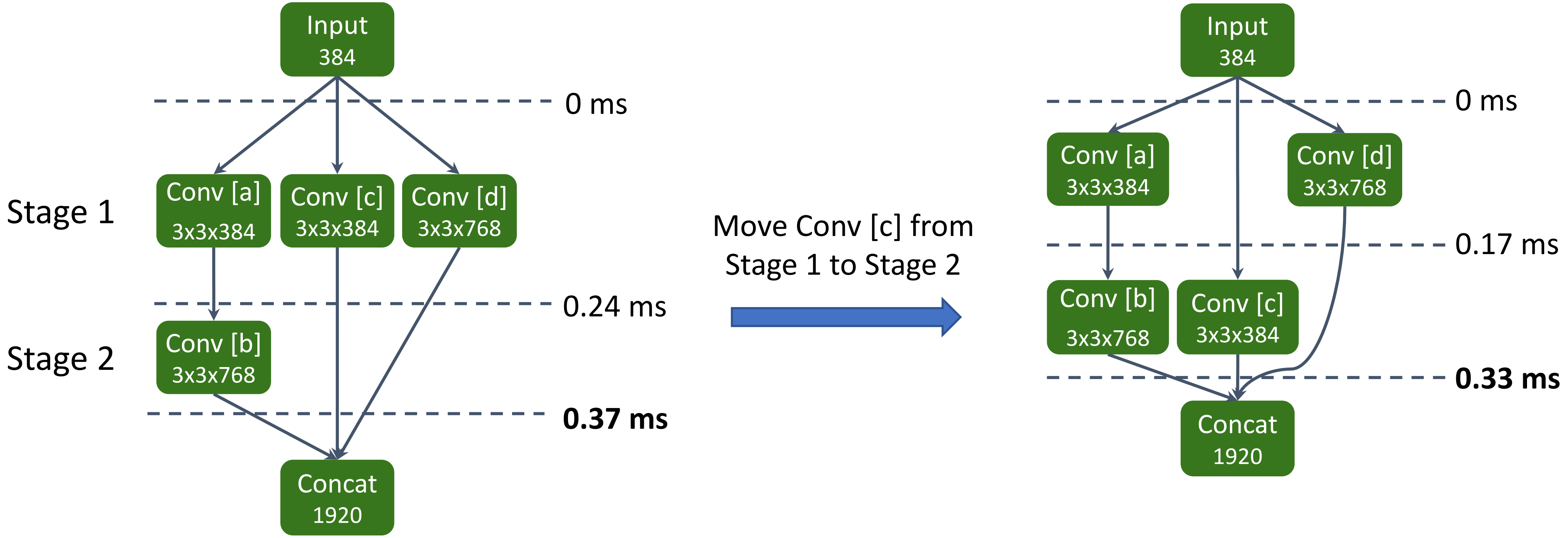


3. Intra- and Inter-operator Parallelization



Background: Wavefront Schedule Policy

Wavefront Schedule Policy: Execute **all available** operators stage by stage

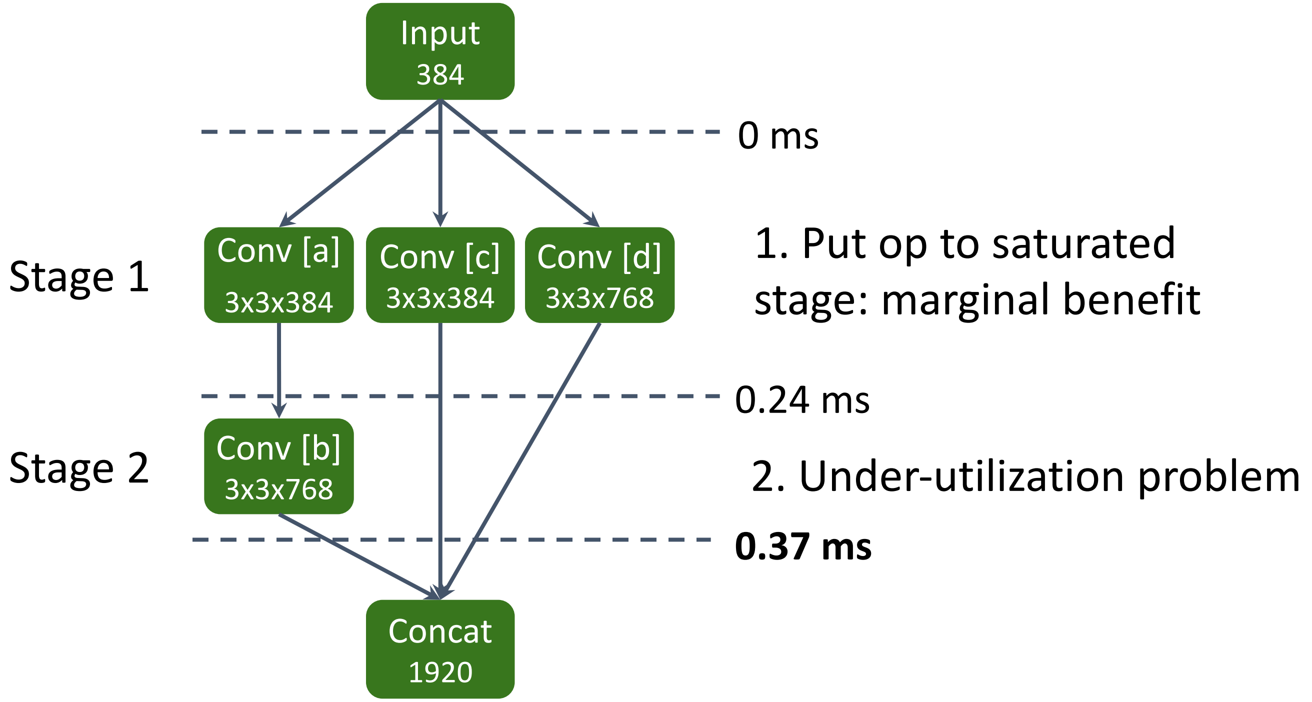


Wavefront Schedule

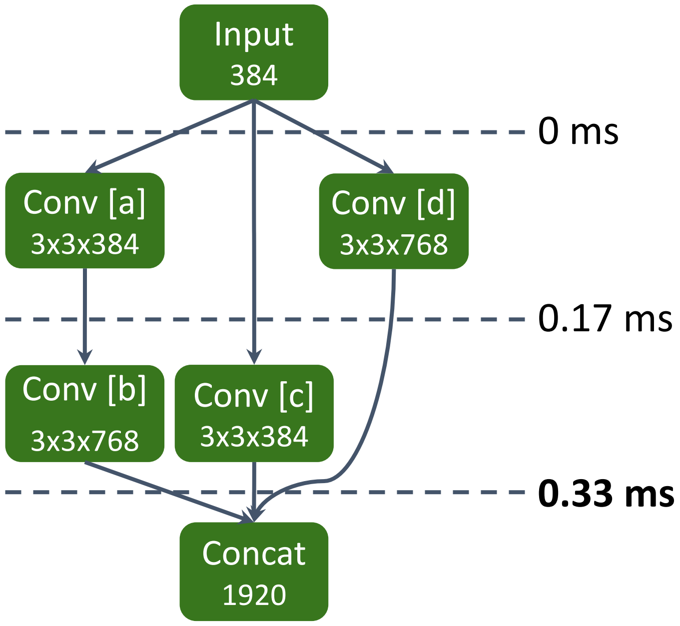
A Better Schedule

Background: Wavefront Schedule Policy

Wavefront Schedule Policy: Execute **all available** operators stage by stage



Wavefront Schedule

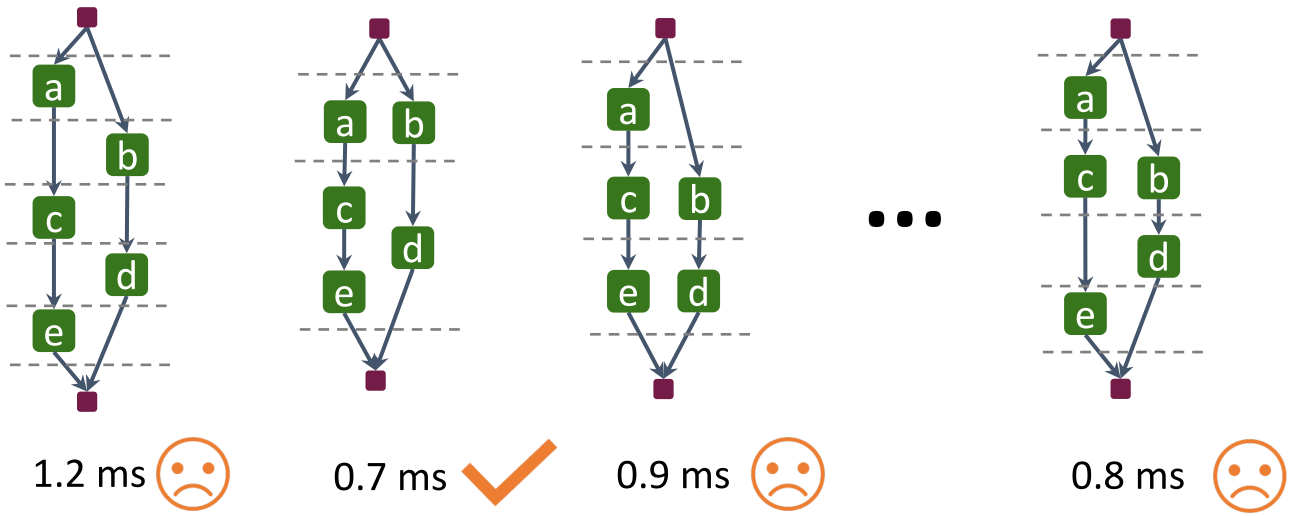


A Better Schedule

Wavefront schedule policy is sub-optimal

Inter-Operator Scheduler (IOS)

General Idea: Explore the schedule space **exhaustively**.



Inter-Operator Scheduler (IOS)

General Idea: Explore the schedule space **exhaustively**.

Challenge: The number of schedules is **exponential** in the number of operators.

e.g., NASNet has more than 10^{12} schedules

⬇

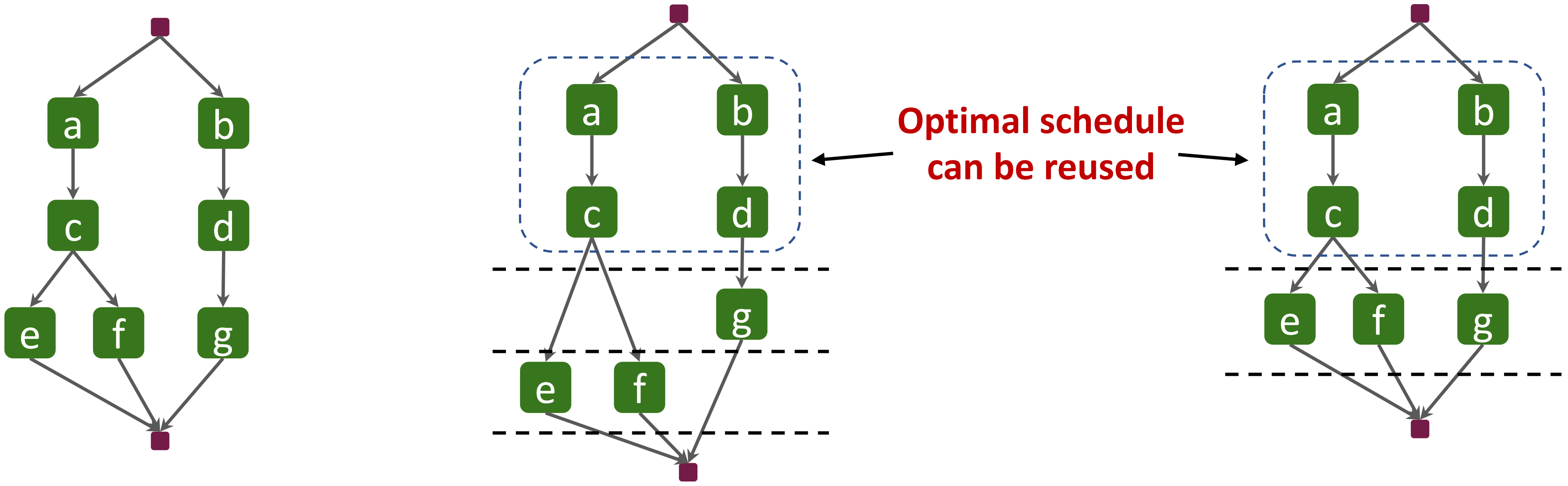
Prohibitive to enumerate

Inter-Operator Scheduler (IOS)

General Idea: Explore the schedule space **exhaustively**.

Challenge: The number of schedules is **exponential** in the number of operators.

Observation 1: Optimal schedule for a subgraph can be reused



Model | Incomplete Schedule | Incomplete Schedule

Inter-Operator Scheduler (IOS)

General Idea: Explore the schedule space **exhaustively**.

Challenge: The number of schedules is **exponential** in the number of operators.

Observation 1: Optimal schedule for a subgraph can be reused

Key Idea: Dynamic Programming

Inter-Operator Scheduler (IOS)

General Idea: Explore the schedule space **exhaustively**.

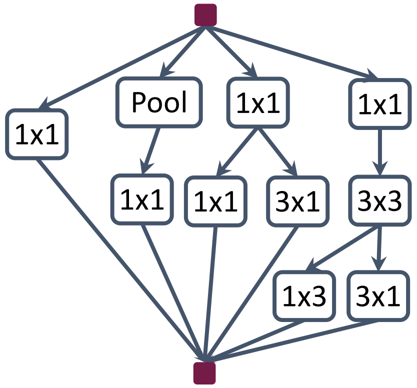
Challenge: The number of schedules is **exponential** in the number of operators.

Observation 1: Optimal schedule for a subgraph can be reused

Key Idea: Dynamic Programming

Observation 2: The width of the computation graph is usually small

Maximum number of parallelizable operators



Inception Block

The width of Inception V3 is 6.

Inter-Operator Scheduler (IOS)

General Idea: Explore the schedule space **exhaustively**.

Challenge: The number of schedules is **exponential** in the number of operators.

Observation 1: Optimal schedule for a subgraph can be reused

Key Idea: Dynamic Programming

Observation 2: The width of the computation graph is usually small

Key Result: Time complexity is only exponential in the width

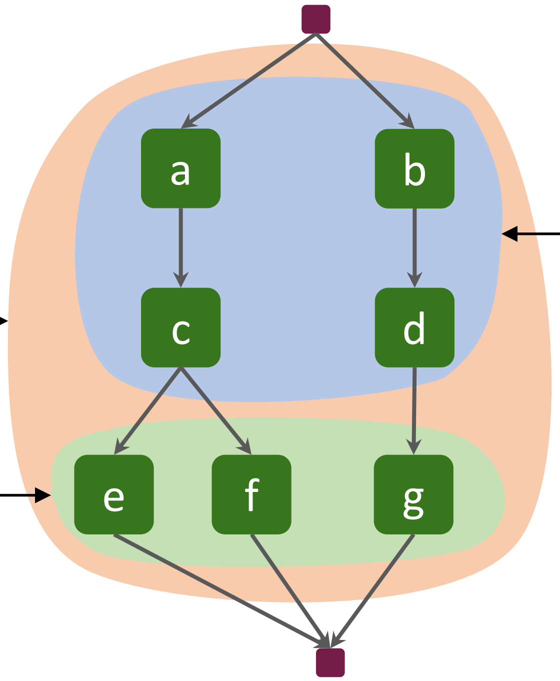
Inter-Operator Scheduler (IOS)

$$Latency[S] = \min_{S' \text{ can be a last stage of } S} \left(Latency[S - S'] + StageLatency(S') \right)$$

Best schedule's latency of S

S is the ops to be scheduled

S' is a candidate for last stage of S

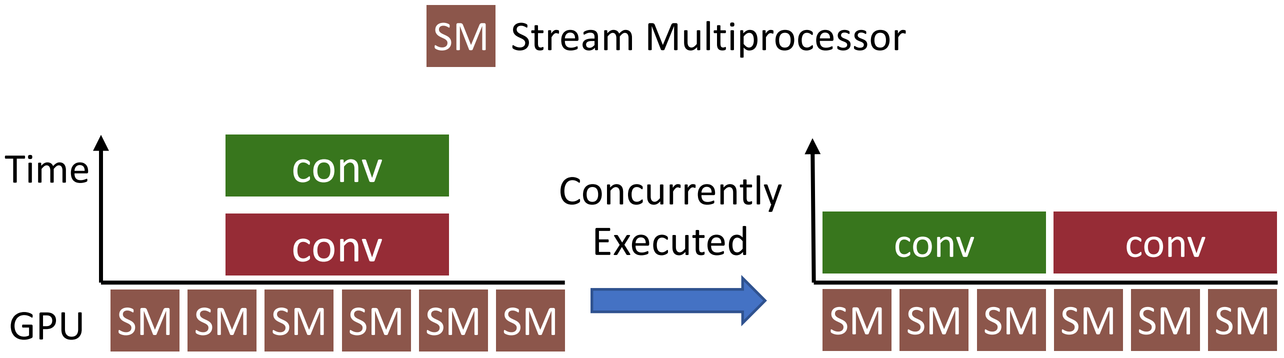


$S - S'$ are ops remaining to be scheduled in a sub-problem

Latency of stage S'

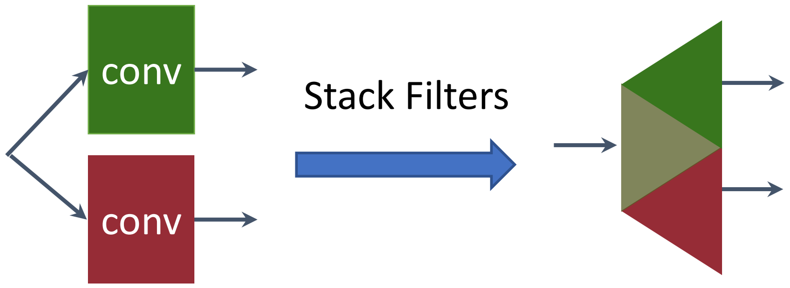
Parallelization Strategy Selection

$$Latency[S] = \min_{S' \text{ can be a last stage of } S} \left(Latency[S - S'] + StageLatency(S') \right)$$



Sequential Execution

Concurrent Execution



Two Convolutions

Merged Convolution

Concurrent Execution

General

Sub-optimal performance

Profile & Select

Operator Merge

Specialized

Usually better performance

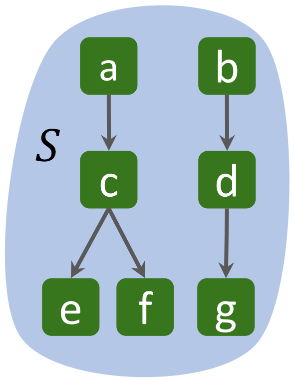
Last Stage Candidates

$$Latency[\,S\,] = \min_{S' \text{ can be a last stage of } S} (\,Latency[\,S - S'\,] + StageLatency(\,S'\,)\,)$$

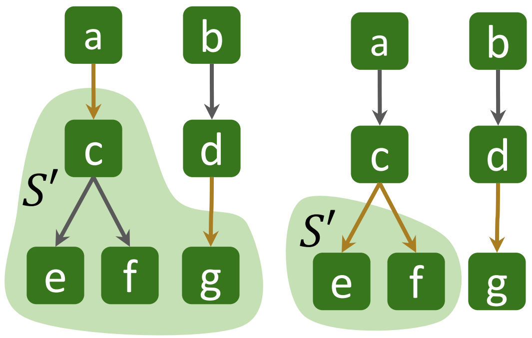
S' can be a last stage of S $\Leftrightarrow$ There is no edge from S' to $S - S'$

↓ Edge from $S - S'$ to S'

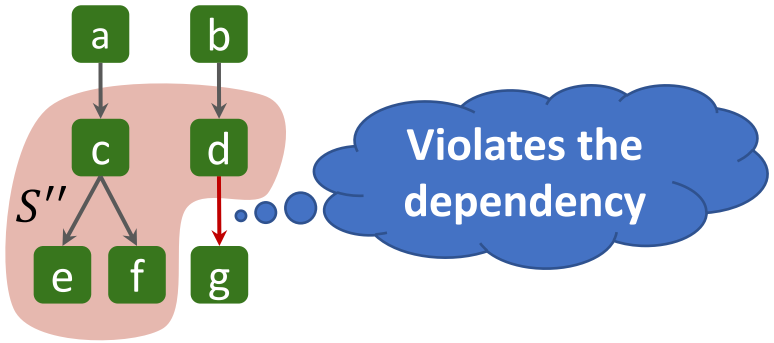
↓ Edge from S' to $S - S'$



Operators S to be scheduled



S' can be a last stage of S



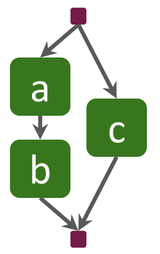
S'' can **NOT** be a last stage of S

Transition Graph and Time Complexity

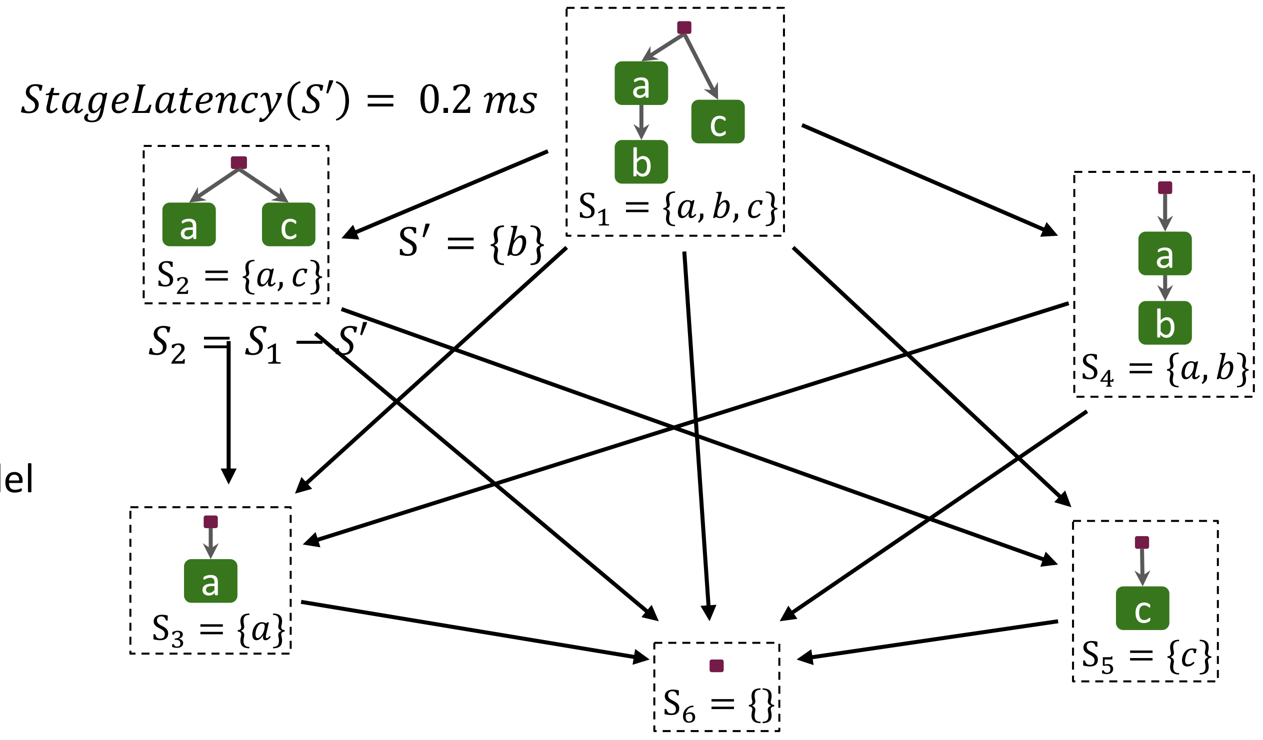
$$Latency[S] = \min_{S' \text{ can be a last stage of } S} \left(Latency[S - S'] + StageLatency(S') \right)$$

Vertices: all valid *state* S

Edges: $S \rightarrow (S - S')$



A Simple Model



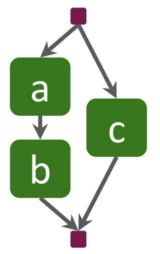
Transition Graph

Transition Graph and Time Complexity

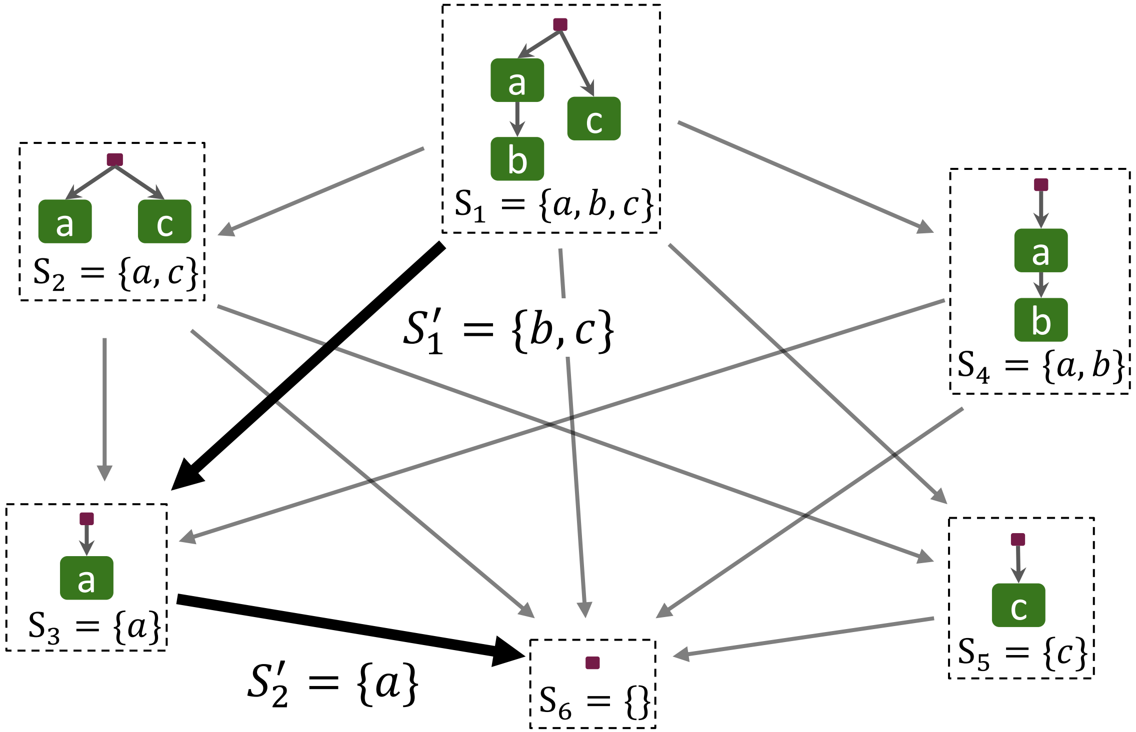
$$Latency[S] = \min_{S' \text{ can be a last stage of } S} (Latency[S - S'] + StageLatency(S'))$$

Vertices: all valid state S

Edges: $S \rightarrow (S - S')$

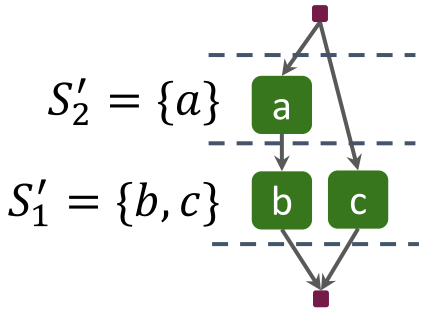


A Simple Model



Transition Graph

IOS: Find the **shortest** path



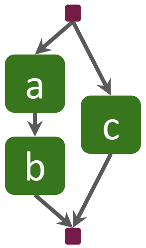
Any path from S_1 to S_6 is a schedule

Transition Graph and Time Complexity

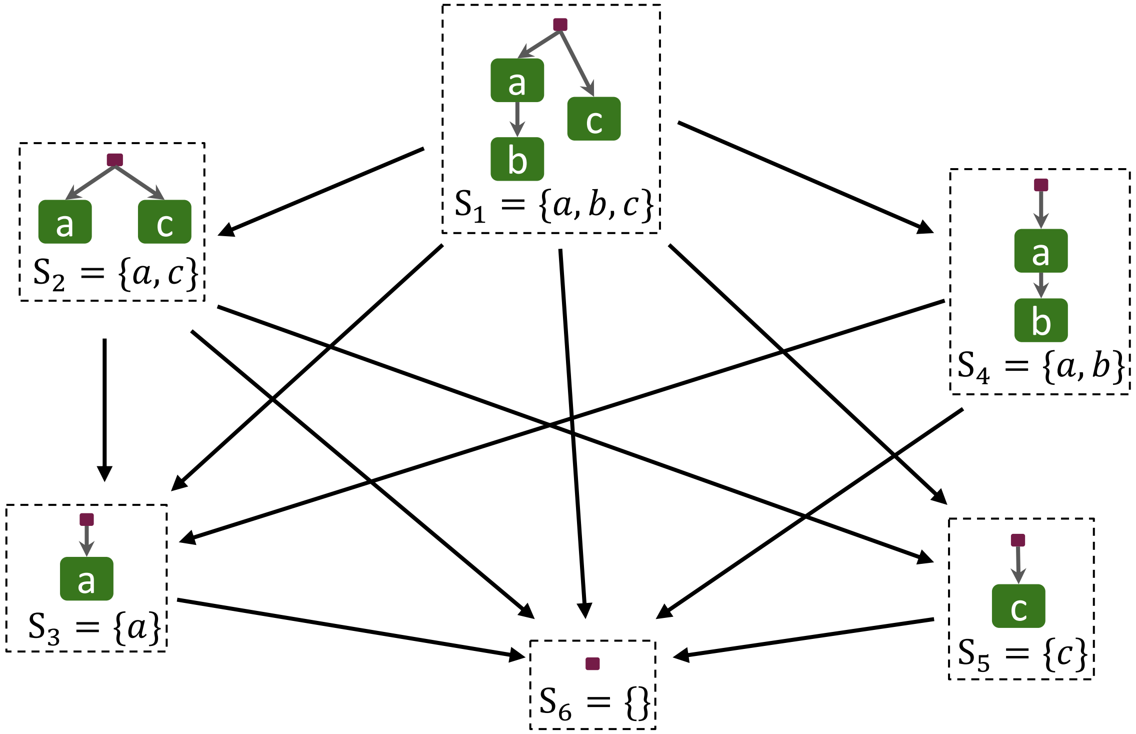
$$Latency[S] = \min_{S' \text{ can be a last stage of } S} \left(Latency[S - S'] + StageLatency(S') \right)$$

Vertices: all valid state S

Edges: $S \rightarrow (S - S')$



A Simple Model



Transition Graph

IOS: Find the **shortest** path

Time Complexity of IOS:

$$\mathcal{O}\left((n/d + 1)^{2d}\right)$$

$\boldsymbol{n}$: number of operators

$\boldsymbol{d}$: max number of parallelizable ops

Methodology

- **Benchmarks**
  - Inception V3 } Expert Designed
  - SqueezeNet
  - Randwire } Neural Architecture Search
  - NasNet

- **Baselines**
  - State-of-the-art Frameworks (cuDNN-based)

    TensorFlow   TensorRT   TASO

  - Different schedules on IOS Runtime

- **Environment**

NVIDIA Tesla V100

NVIDIA CUDA 10.2   cuDNN 7.6.5

- **IOS Implementation**
  - cuDNN kernels
  - CUDA Stream

Comparison of cuDNN-based Frameworks

Tensorflow: A popular machine learning framework.

Tensorflow-XLA: TensorFlow with compilation optimization.

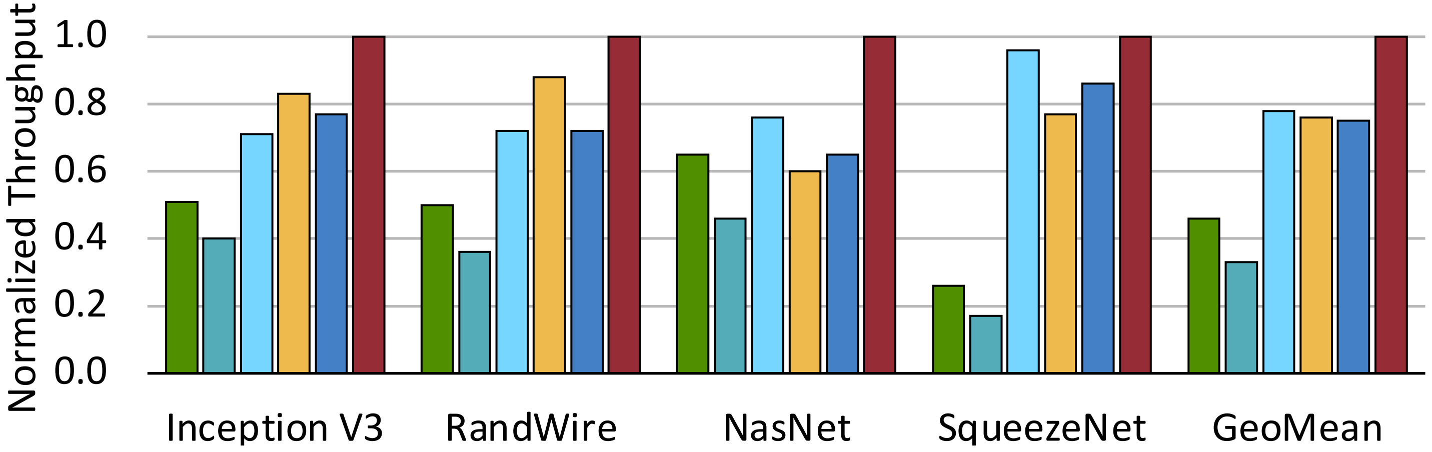
TASO: Transformation-based optimizer.

TVM-cuDNN: TVM backed with cuDNN convolution kernel.

TensorRT: NVIDIA high-performance inference engine.

IOS: Our method

Under-utilization due to sequential execution



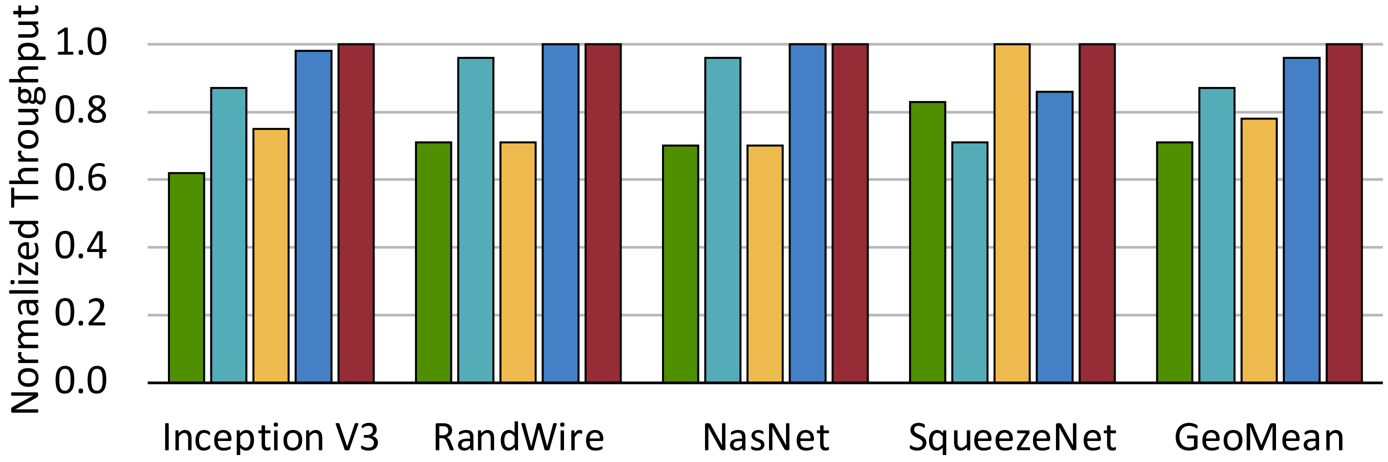
Performance is normalized to the best framework

IOS outperforms all frameworks and achieves **1.1-1.5x** speedup.

Comparison of Different Schedules

IOS Runtime

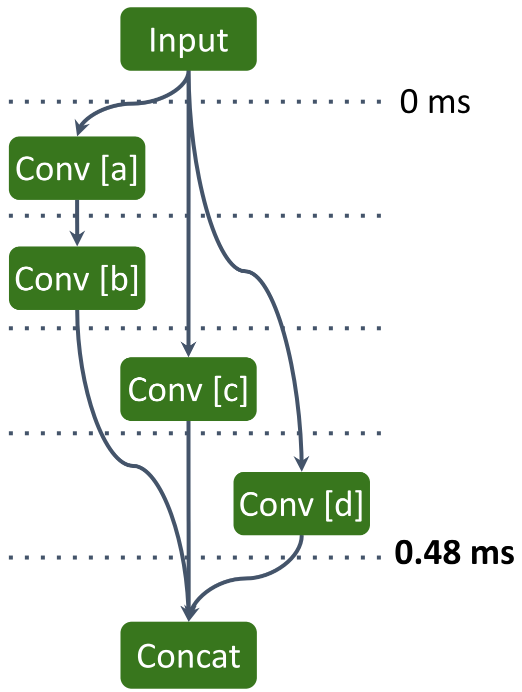
- **Sequential:** Run each op sequentially. — *Under-utilize Device*
- **Wavefront:** Run all available ops stage by stage. — *Unbalanced Schedule*
- **IOS-Merge:** IOS with only "operator merge" policy. — *No trade off between parallelization strategies*
- **IOS-Parallel:** IOS with only "parallel execution" policy.
- **IOS-Both:** IOS with both policies



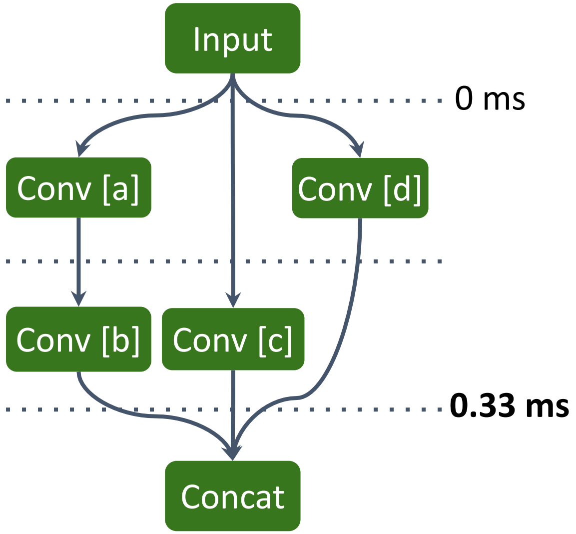
Performance is normalized to the best schedule

IOS-Both achieves the best performance

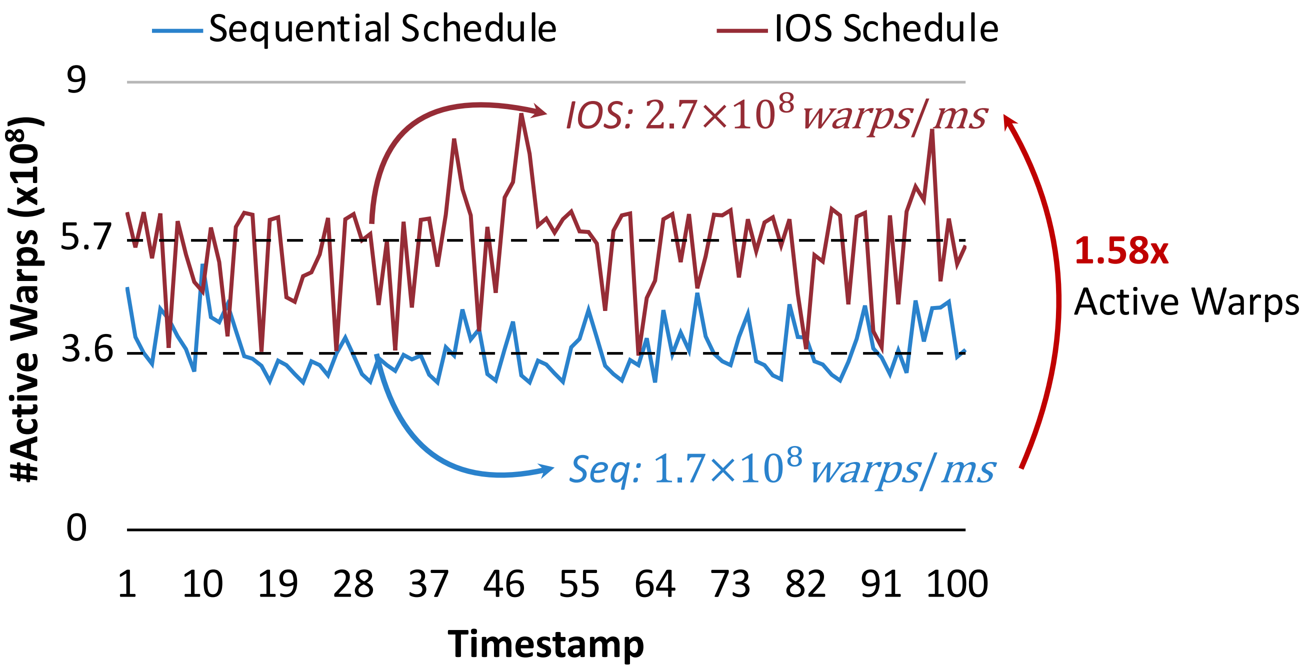
More Active Warps Improve Utilization



Sequential Schedule



IOS Schedule



NVIDIA CUPTI profile frequency is every 2.1 ms.

More active warps ➡ More eligible warps to execute at each cycle ➡ Higher Device-Utilization

Conclusion

- Sequential execution suffers from **under utilization** problem.
- **I**nter-**O**perator **S**cheduler (**IOS**):
  - Utilize both intra- and **inter-operator parallelism** in CNNs.
  - **Dynamic-programming** explores the schedule space **exhaustively**.
  - Time Complexity: $\mathcal{O}\left((n/d+1)^{2d}\right)$, d is usually small.
- Key Results: **1.1-1.5x** speedup on diverse CNNs.

https://github.com/mit-han-lab/inter-operator-scheduler

We provide scripts to reproduce results in every figure and table!