IOS: Inter-Operator Scheduler for CNN Acceleration

Yaoyao Ding\textsuperscript{1,2}, Ligeng Zhu\textsuperscript{3}, Zhihao Jia\textsuperscript{4},
Gennady Pekhimenko\textsuperscript{1,2}, Song Han\textsuperscript{3}
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

- More powerful GPUs
- Device under-utilization
- Better device utilization

- More small convs in CNN design.

- GPU peak performance increased

- Intra- and Inter-operator Parallelization

- Sequential Execution
- Inter-Op Parallel Execution

- More convs
- Thread
- Small op & Powerful GPU
- Better device utilization
Background: Wavefront Schedule Policy

Wavefront Schedule Policy: Execute all available operators stage by stage

Stage 1
- Input 384
- Conv [a] 3x3x384
- Conv [c] 3x3x384
- Conv [d] 3x3x768

Stage 2
- Conv [b] 3x3x768
- Concat 1920

Wavefront Schedule

Move Conv [c] from Stage 1 to Stage 2

Stage 1
- Conv [a] 3x3x384
- Conv [b] 3x3x768
- Conv [c] 3x3x384
- Conv [d] 3x3x768

Stage 2
- Concat 1920

A Better Schedule
Wavefront Schedule Policy: Execute all available operators stage by stage

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
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

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)

\[
\text{Latency}[S] = \min \left( \text{Latency}[S - S'] + \text{StageLatency}(S') \right)
\]

- \(S\) is the ops to be scheduled
- \(S'\) can be a last stage of \(S\)
- \(S'\) is a candidate for last stage of \(S\)
- \(S - S'\) are ops remaining to be scheduled in a sub-problem
- Latency of stage \(S'\)

Best schedule’s latency of \(S\)
Parallelization Strategy Selection

\[
\text{Latency}[S] = \min_{S'} \left( \text{Latency}[S - S'] + \text{StageLatency}(S') \right)
\]

Sequential Execution

Concurrent Execution

Concurrently Executed

Profile & Select

Concurrent Execution

General
Sub-optimal performance

Operator Merge

Specialized
Usually better performance
Last Stage Candidates

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

\( S' \) can be a last stage of \( S \) \( \iff \) There is no edge from \( S' \) to \( S - S' \)

Operators \( S \) to be scheduled

\( S' \) can be a last stage of \( S \)

\( S'' \) can \textbf{NOT} be a last stage of \( S \)
Transition Graph and Time Complexity

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

**Vertices:** all valid state \(S\)

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

StageLatency(\(S'\)) = 0.2 ms

A Simple Model

Transition Graph
Transition Graph and Time Complexity

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

**Vertices:** all valid state \( S \)

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

A Simple Model

\[ S_2 = \{a, c\} \]
\[ S_1 = \{a, b, c\} \]
\[ S_4 = \{a, b\} \]
\[ S_5 = \{c\} \]
\[ S_3 = \{a\} \]
\[ S' = \{b, c\} \]
\[ S'' = \{a\} \]

IOS: Find the **shortest** path

\[ S_2' = \{a\} \]
\[ S_1' = \{b, c\} \]

Any path from \( S_1 \) to \( S_6 \) is a schedule
Latency of a system can be defined as:

\[ \text{Latency}[S] = \min_{S'} \left( \text{Latency}[S - S'] + \text{StageLatency}(S') \right) \]

where \( S' \) can be a last stage of \( S \).

**Vertices:** all valid state \( S \)

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

**A Simple Model:**

- \( S_1 = \{a, b, c\} \)
- \( S_2 = \{a, c\} \)
- \( S_3 = \{a\} \)
- \( S_4 = \{a, b\} \)
- \( S_5 = \{c\} \)
- \( S_6 = \{} \)

**Transition Graph**

**Time Complexity** of IOS:

\[ \mathcal{O}\left(\left(\frac{n}{d} + 1\right)^{2d}\right) \]

- \( n \): number of operators
- \( d \): max number of parallelizable ops

**IOS:** Find the **shortest** path
Methodology

- **Benchmarks**
  - Inception V3
  - SqueezeNet
  - Randwire
  - NasNet

- **Environment**
  - NVIDIA Tesla V100

- **Baselines**
  - State-of-the-art Frameworks (cuDNN-based)
  - Different schedules on IOS Runtime

- **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.
IOS: Our method

Under-utilization due to sequential execution

IOS outperforms all frameworks and achieves \textbf{1.1-1.5x} speedup.

Performance is normalized to the best framework.
Comparison of Different Schedules

**IOS Runtime**

- **Sequential:** Run each op sequentially.
- **Wavefront:** Run all available ops stage by stage.
- **IOS-Merge:** IOS with only “operator merge” policy.
- **IOS-Parallel:** IOS with only “parallel execution” policy.
- **IOS-Both:** IOS with both policies

Performance is normalized to the best schedule

**Under-utilize Device**

**Unbalanced Schedule**

**No trade off between parallelization strategies**

**IOS-Both** achieves the best performance
More Active Warps Improve Utilization

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

Sequential Schedule

IOS Schedule

NVIDIA CUPTI profile frequency is every 2.1 ms.
Conclusion

• Sequential execution suffers from **under utilization** problem.

• **Inter-Operator Scheduler (IOS):**
  • Utilize both intra- and **inter-operator parallelism** in CNNs.
  • **Dynamic-programming** explores the schedule space **exhaustively**.
  • Time Complexity: $\mathcal{O} \left( \left(n/d + 1\right)^{2d} \right)$, $d$ is usually small.

• Key Results: **1.1-1.5x** speedup on diverse CNNs.

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

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

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