

EPFL

Duplo: Lifting Redundant Memory Accesses of Deep Neural Networks for GPU Tensor Cores

The 53rd IEEE/ACM International Symposium on Microarchitecture
Oct. 20, 2020

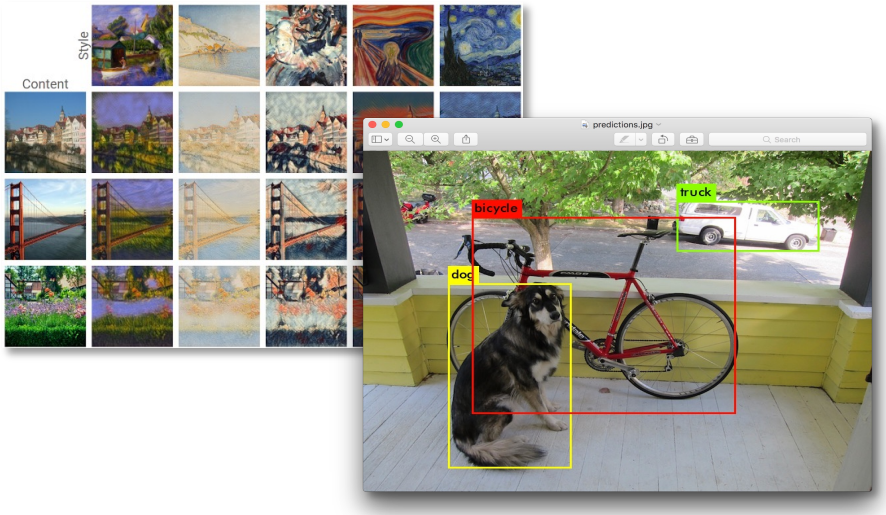
Hyeonjin Kim[1], Sungwoo Ahn[1], Yunho Oh[2]
Bogil Kim[1], Won Woo Ro[1], and William J. Song[1]

[1]School of Electrical and Electronic Engineering, Yonsei University
[2]EcoCloud, École Polytechnique Fédérale de Lausanne (EPFL)

Outline

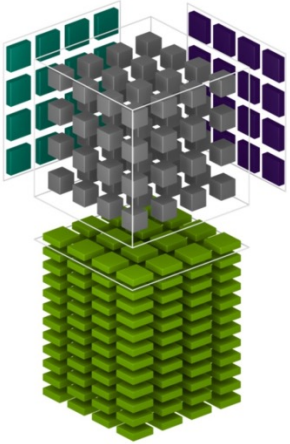
- **Introduction**
- Background and Motivation
- Duplo Architecture
- Evaluation Results
- Conclusion

GPUs for Deep Learning

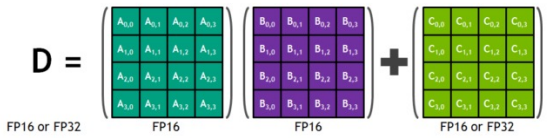


- **Convolution** is one of the fundamental operations in DNNs.
- ResNet, GAN, and YOLO spend **more than 99% of runtime** on executing convolutions.

- GPUs serve as de facto hardware for deep learning, and **tensor cores** in recent NVIDIA GPUs offer **greater operational intensity**.

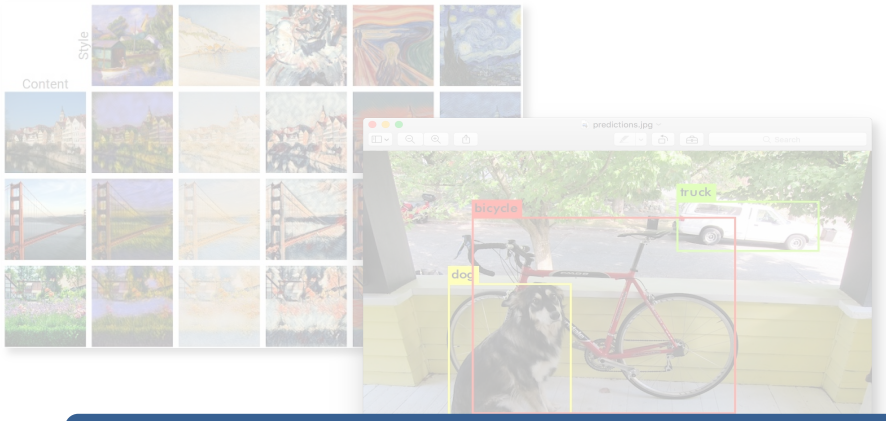






- NVIDIA, “Volta Tensor Core GPU Achieves New AI Performance Milestones,” 2018.
- TensorFlow, “Artistic Style Transfer with TensorFlow Lite” [Online] https://www.tensorflow.org/
- Redmon et al., “YOLO9000: Better, Faster, Stronger,” 2016.

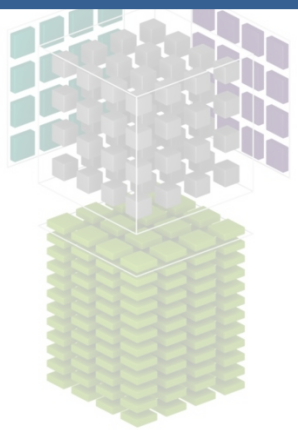
GPU for Deep Learning



Convolution is one of the fundamental operations in DNNs. ResNet, GAN, and YOLO spend **more than 99% of runtime** on

Accelerating convolutions is essential to speed up DNNs.

GPUs serve as de facto hardware for deep learning, and **tensor cores** in recent NVIDIA GPUs offer **greater operational intensity**.







- NVIDIA, “Volta Tensor Core GPU Achieves New AI Performance Milestones,” 2018.
- TensorFlow, “Artistic Style Transfer with TensorFlow Lite” [Online] https://www.tensorflow.org/
- Redmon et al., “YOLO9000: Better, Faster, Stronger,” 2016.

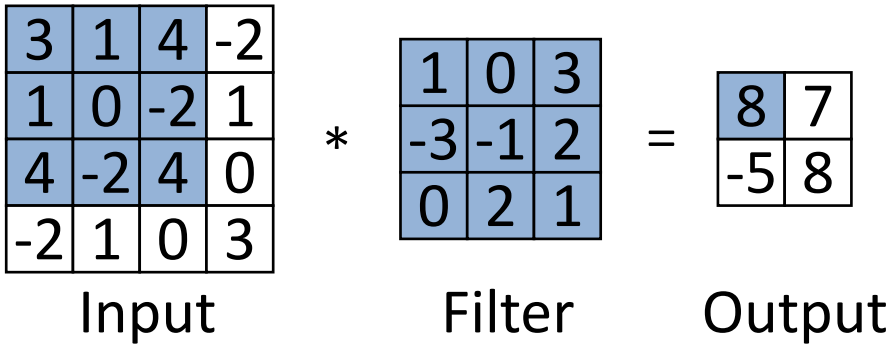
Outline

- Introduction
- **Background and Motivation**
- Duplo Architecture
- Evaluation Results
- Conclusion

Convolution in DNN

- **Direct convolution:** naive representation of a convolution algorithm forms **deeply nested convolution loops** in a program code.

Striding filters across multi-dimensional data

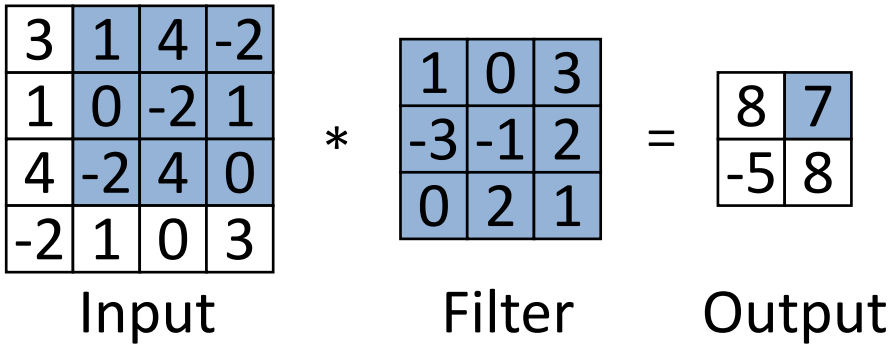



```
// deeply nested convolution loops
for(input_batch)
  for(filter_batch)
    for(input_channel)
      for(input_width)
        for(input_height)
          for(filter_width)
            for(filter_channel)
              output += input*filter;
```

Convolution in DNN

- **Direct convolution:** naive representation of a convolution algorithm forms **deeply nested convolution loops** in a program code.

Striding filters across multi-dimensional data

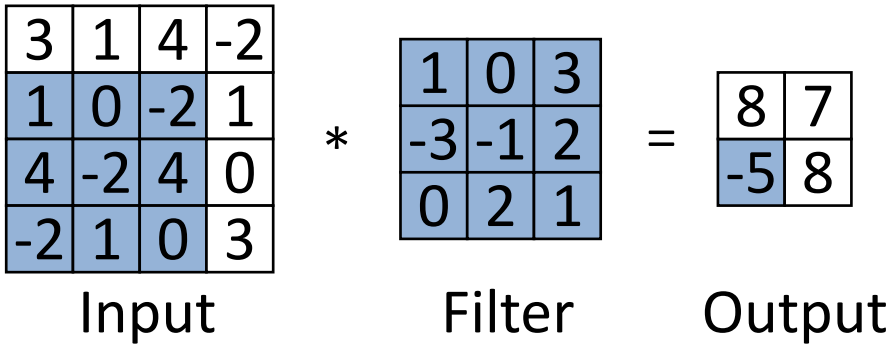



```
// deeply nested convolution loops
for(input_batch)
  for(filter_batch)
    for(input_channel)
      for(input_width)
        for(input_height)
          for(filter_width)
            for(filter_channel)
              output += input*filter;
```

Convolution in DNN

- **Direct convolution:** naive representation of a convolution algorithm forms **deeply nested convolution loops** in a program code.

Striding filters across multi-dimensional data

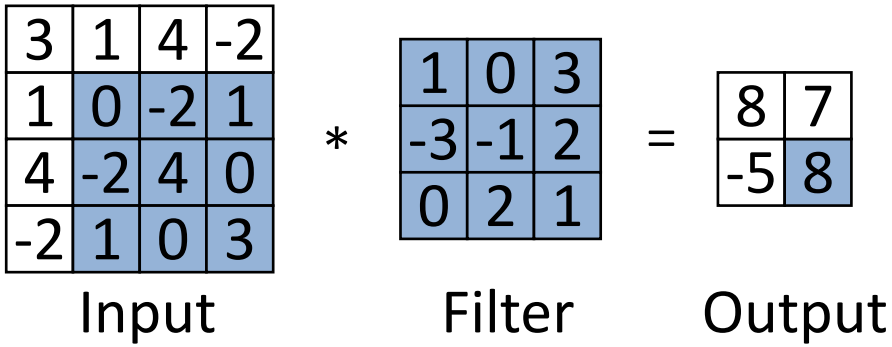



```
// deeply nested convolution loops
for(input_batch)
  for(filter_batch)
    for(input_channel)
      for(input_width)
        for(input_height)
          for(filter_width)
            for(filter_channel)
              output += input*filter;
```

Convolution in DNN

- **Direct convolution:** naive representation of a convolution algorithm forms **deeply nested convolution loops** in a program code.

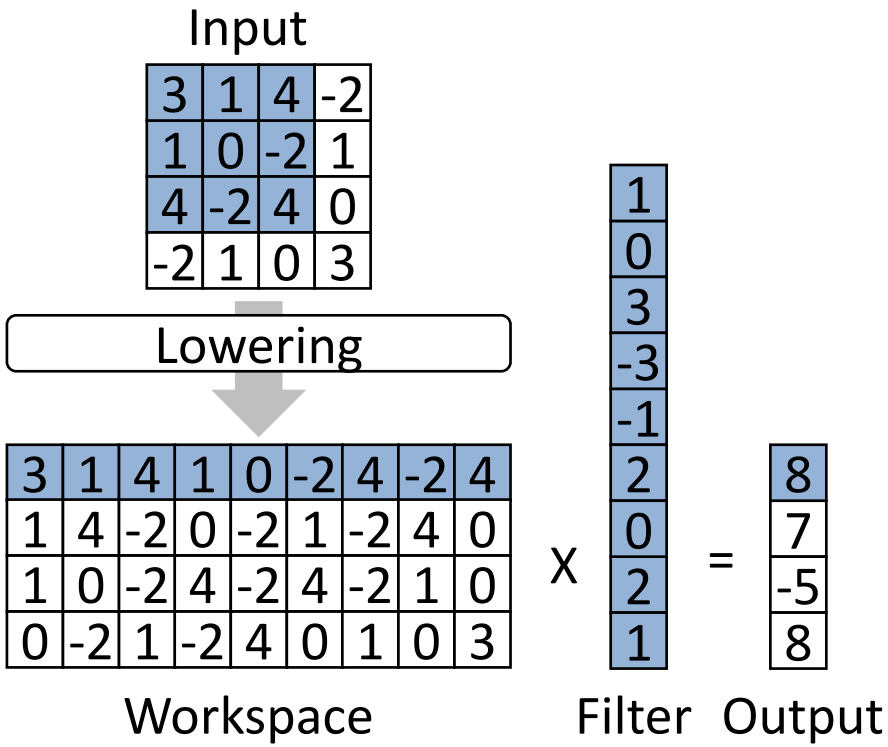
Striding filters across multi-dimensional data




```
// deeply nested convolution loops
for(input_batch)
  for(filter_batch)
    for(input_channel)
      for(input_width)
        for(input_height)
          for(filter_width)
            for(filter_channel)
              output += input*filter;
```

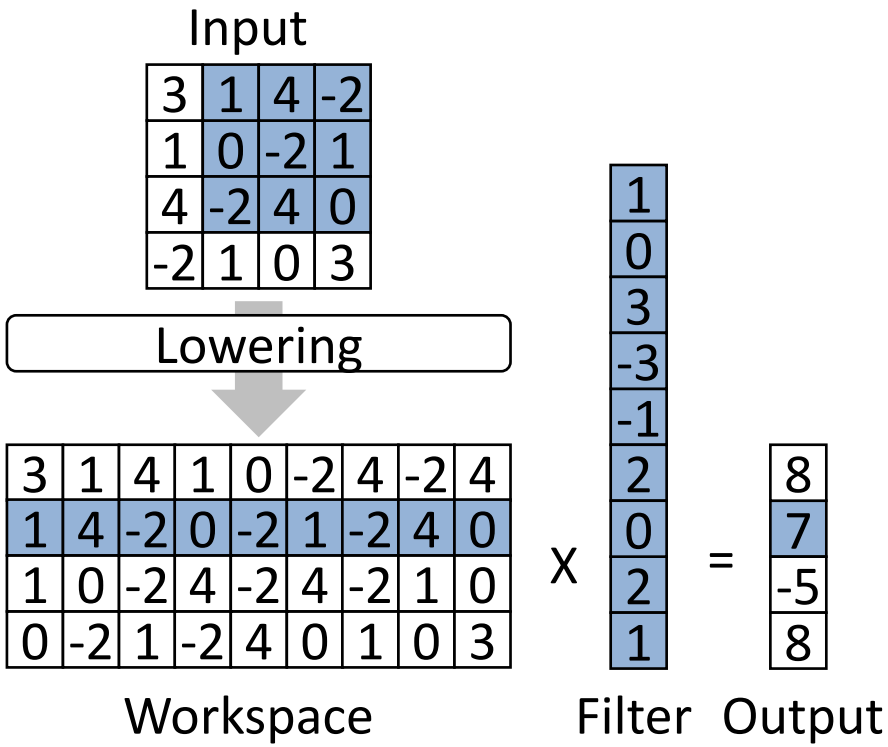
Lowering the Convolution

- **Lowering the convolution** transforms the deeply nested convolution loops into **matrix multiplication**.
- **GEMM-based convolution:** matrix multiplication can be rapidly calculated by using a GEMM library.



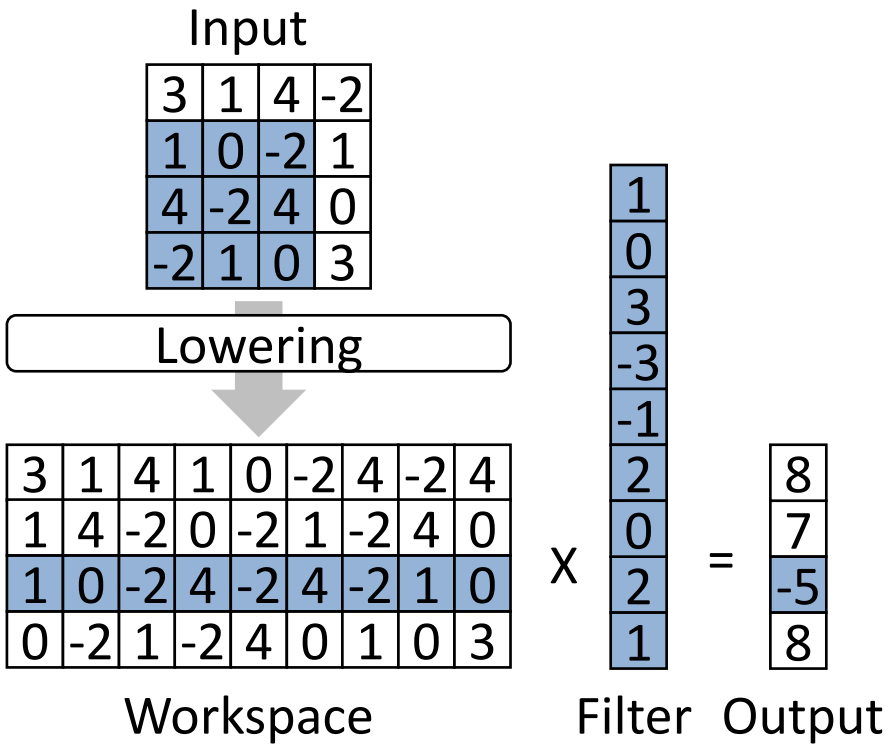
Lowering the Convolution

- **Lowering the convolution** transforms the deeply nested convolution loops into **matrix multiplication**.
- **GEMM-based convolution:** matrix multiplication can be rapidly calculated by using a GEMM library.



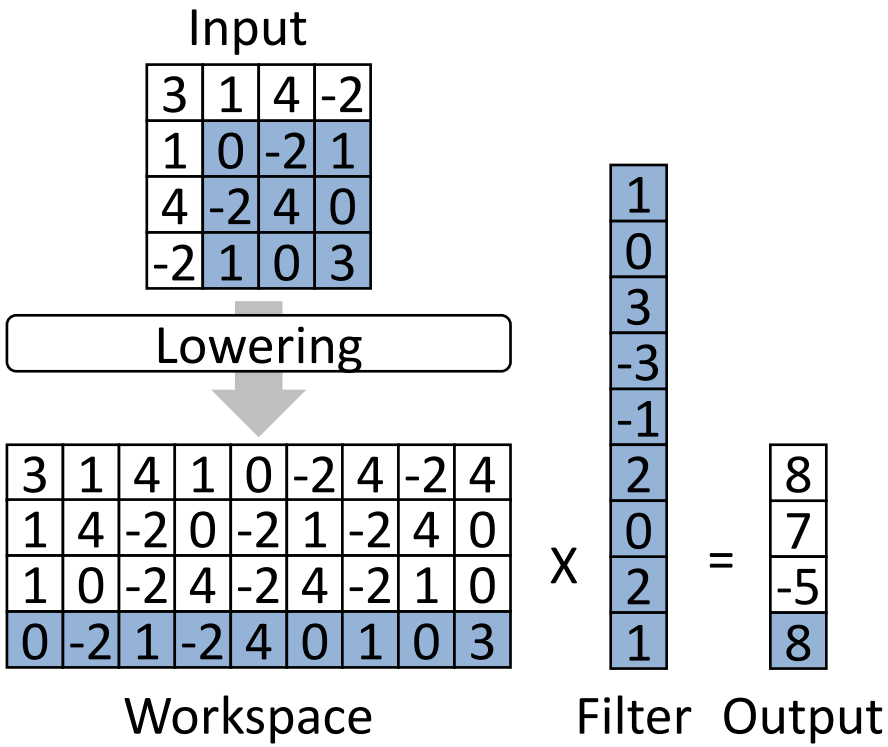
Lowering the Convolution

- **Lowering the convolution** transforms the deeply nested convolution loops into **matrix multiplication**.
- **GEMM-based convolution:** matrix multiplication can be rapidly calculated by using a GEMM library.



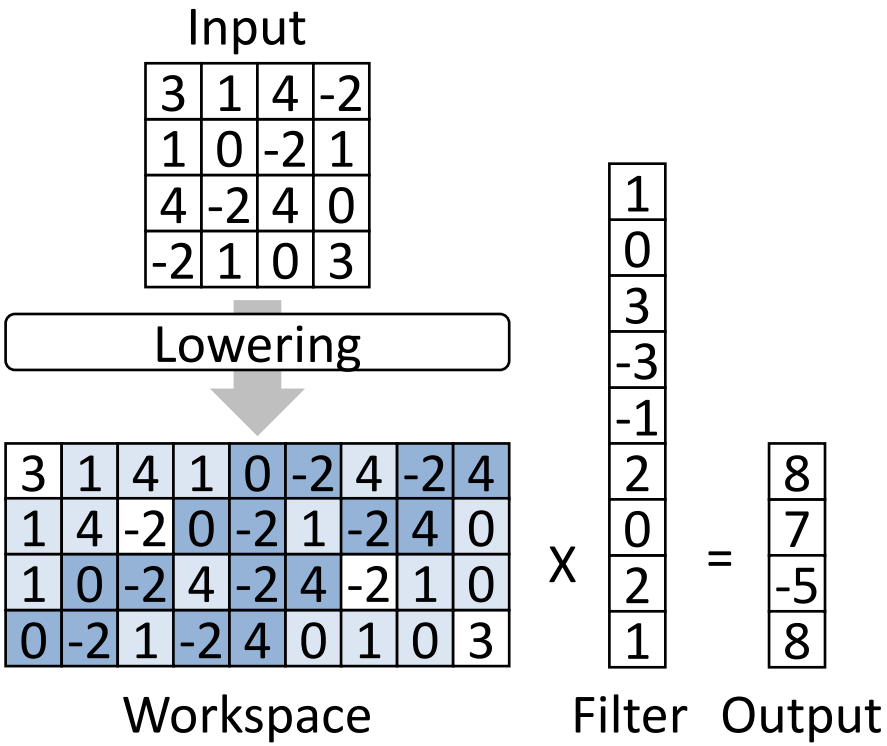
Lowering the Convolution

- **Lowering the convolution** transforms the deeply nested convolution loops into **matrix multiplication**.
- **GEMM-based convolution:** matrix multiplication can be rapidly calculated by using a GEMM library.



Data Duplication in the GEMM Workspace

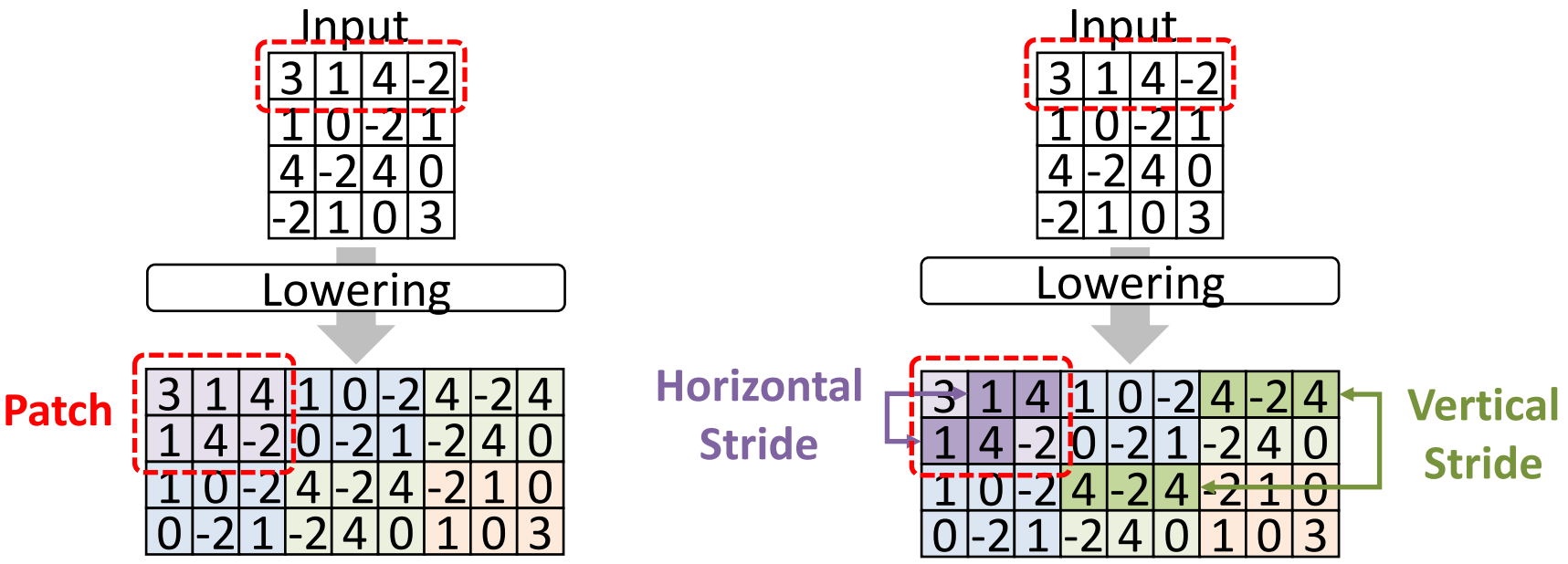
- Lowering produces **multiple duplicate copies of input data**.
- **Redundant memory accesses** are generated to load the replicates of same data located at different memory addresses.



Large workspace is filled with many duplicate data!

Sources of Data Duplication

- Striding of a filter creates a grid pattern of data duplication.
- **Horizontal stride** of the filter replicates data **within a patch**.
- **Vertical stride** replicates data **across patches**.



Data Duplication Detection Mechanism

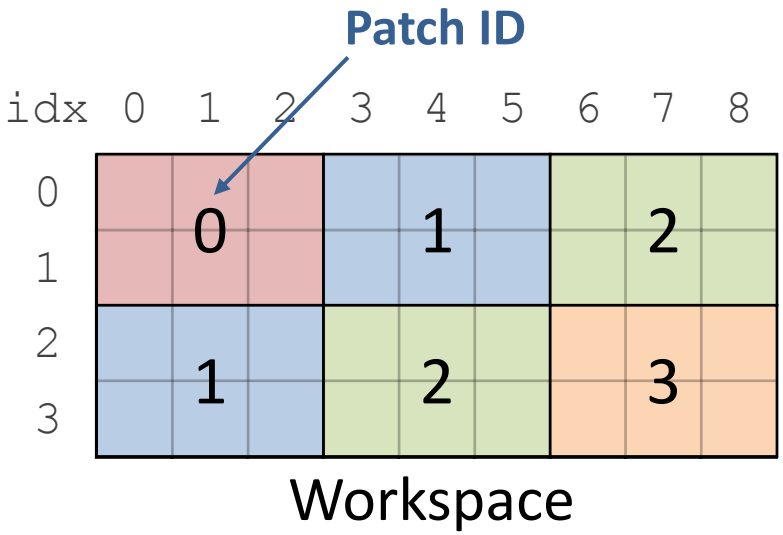
idx	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	9	10	11	12	13	14	15	16	17
2	18	19	20	21	22	23	24	25	26
3	27	28	29	30	31	32	33	34	35

Workspace

Step 1: Workspace row & col ID calculation

```
worksp_row_idx = array_idx
/ (filter_width * filter_height)
 worksp_col_idx = array_idx
% (filter_width * filter_height)
```

Data Duplication Detection Mechanism



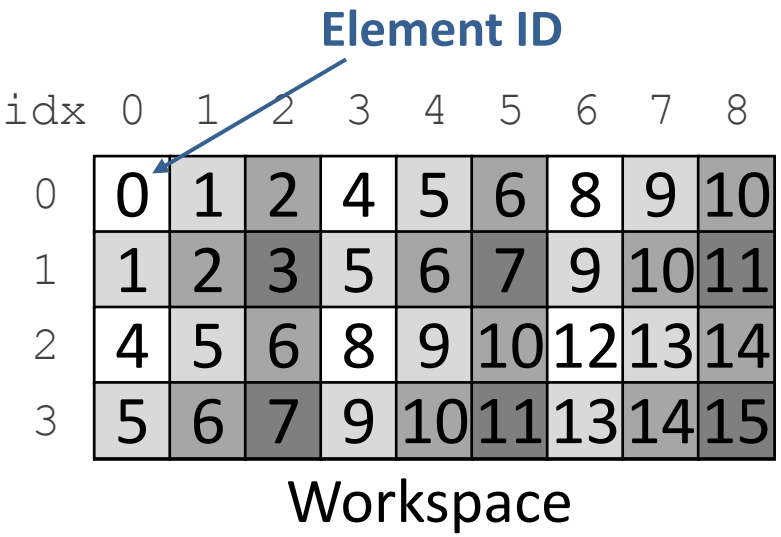
Step 1: Workspace row & col ID calculation

```
worksp_row_idx = array_idx
/ (filter_width * filter_height)
 worksp_col_idx = array_idx
% (filter_width * filter_height)
```

Step 2: Patch ID calculation

```
patch_row = worksp_row_idx /
output_height
patch_col = worksp_col_idx /
filter_width
patch_id = patch_row_idx +
patch_col_idx
```

Data Duplication Detection Mechanism



Workspace

Step 1: Workspace row & col ID calculation

```
worksp_row_idx = array_idx
/ (filter_width * filter_height)
 worksp_col_idx = array_idx
% (filter_width * filter_height)
```

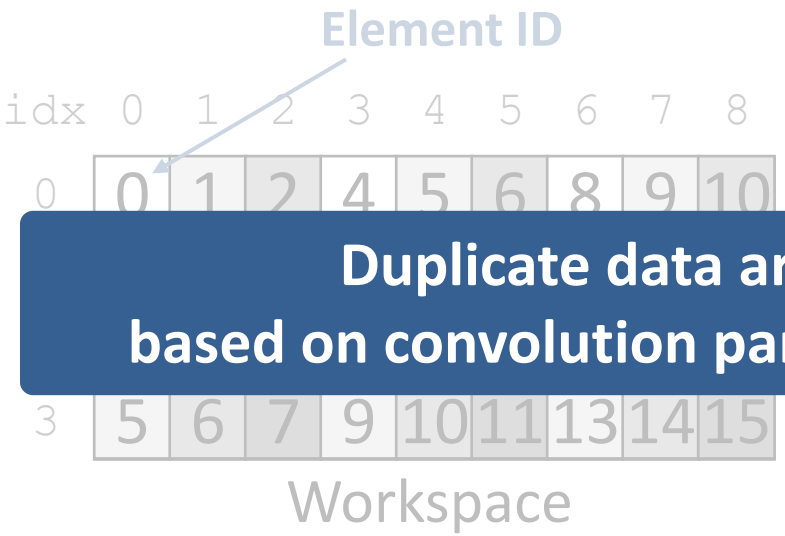
Step 2: Patch ID calculation

```
patch_row = worksp_row_idx /
output_height
patch_col = worksp_col_idx /
filter_width
patch_id = patch_row_idx +
patch_col_idx
```

Step 3: Unique element ID calculation

```
element_id = worksp_row_idx %
output_width + worksp_col_idx %
filter_width + offset
offset = patch_id * input_width
batch_id = worksp_row_idx /
(output_width * output_height)
```

Data Duplication Detection Mechanism



Duplicate data are uniquely identifiable based on convolution parameters and data dimensions.

Step 1: Workspace row & col ID calculation

```
worksp_row_idx = array_idx
/ (filter_width * filter_height)
 worksp_col_idx = array_idx
% (filter_width * filter_height)
```

Step 2: Patch ID calculation

```
patch_id = patch_row_idx +
patch_col_idx
```

Step 3: Unique element ID calculation

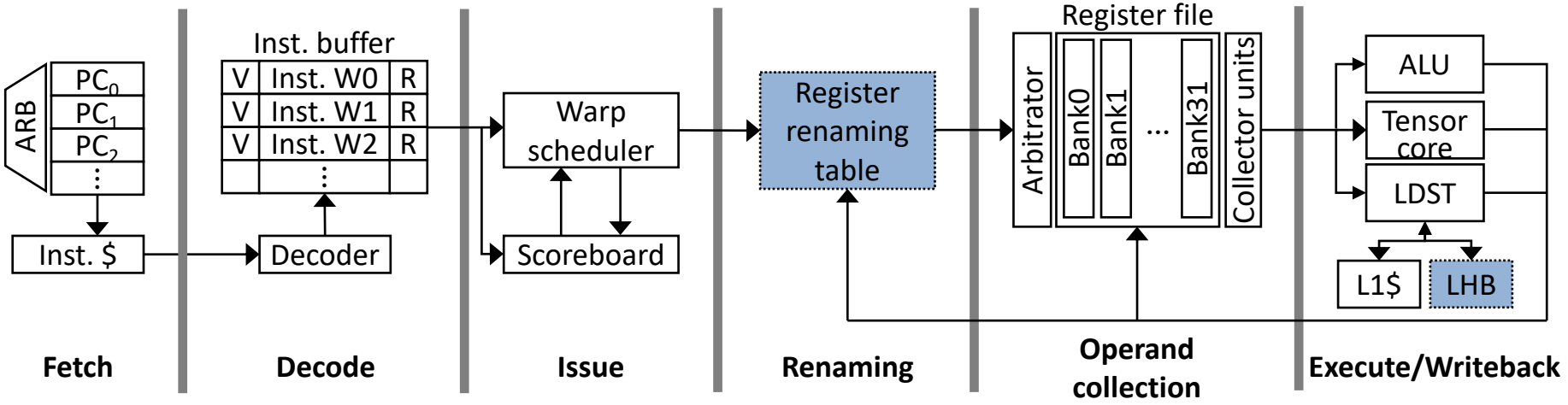
```
element_id = worksp_row_idx %
output_width + worksp_col_idx %
filter_width + offset
offset = patch_id * input_width
batch_id = worksp_row_idx /
(output_width * output_height)
```

Outline

- Introduction
- Background and Motivation
- **Duplo Architecture**
- Evaluation Results
- Conclusion

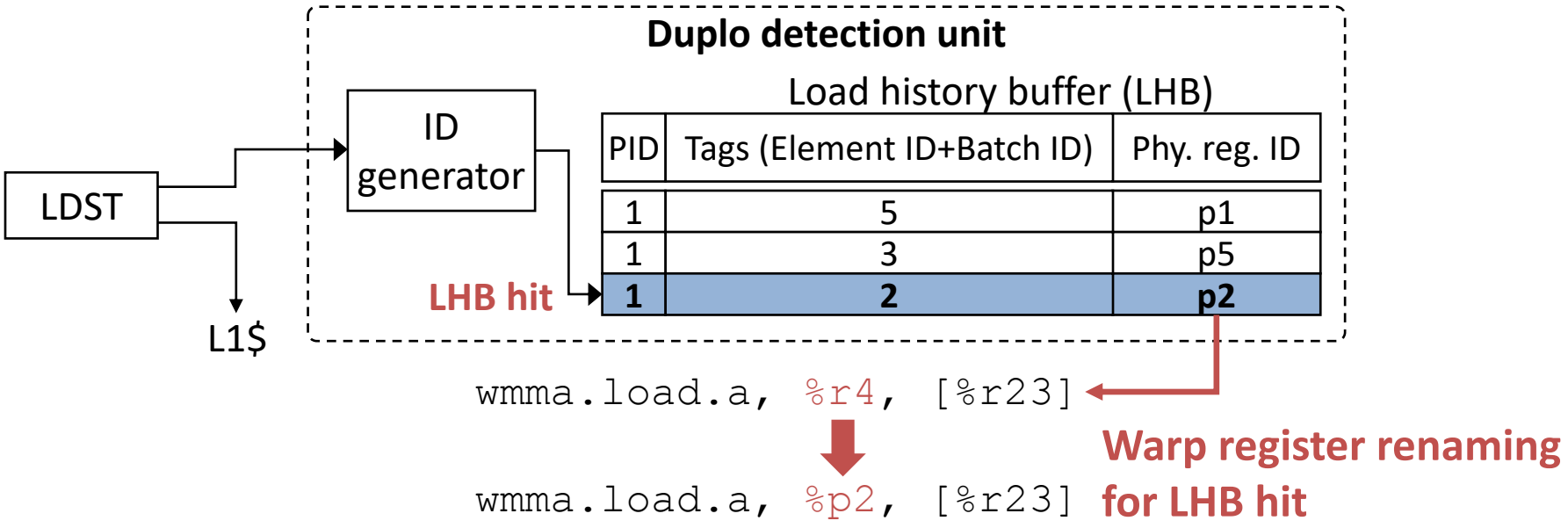
Duplo Architecture Pipeline

- Duplo appends a small **load history buffer (LHB)** to LDST unit to trace the history of workspace data and their locations in RF.
- On an LHB hit, **a warp register is renamed** as the one containing the **identical values** instead of issuing a memory request to fetch the duplicate copy of data.



Tracing Tensor Core Loads and Data Duplication

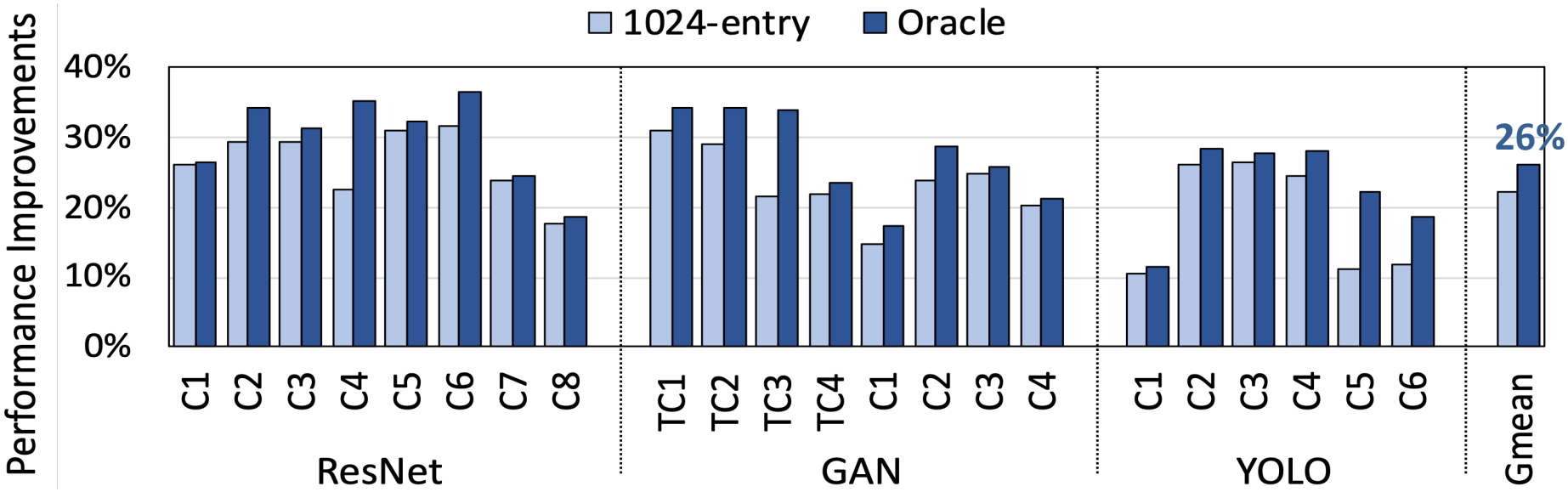
- For a tensor-core load of workspace data, Duplo **looks up the LHB** if the identical data were already loaded.
- Duplo **renames the warp register** on a LHB hit.
- When there is no matching entry in LHB, data is fetched in conventional way from memory heirarchy.



Outline

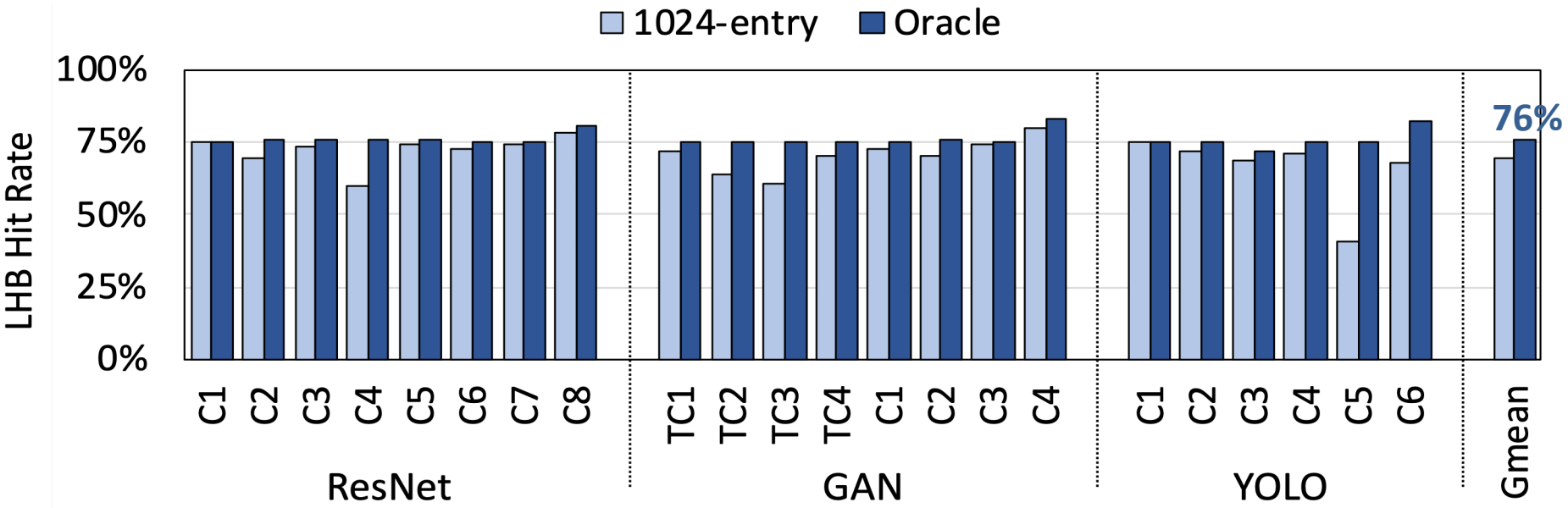
- Introduction
- Background and Motivation
- Duplo Architecture
- **Evaluation Results**
- Conclusion

Performance Improvements



- GPUs are known to hide memory latencies well, but Duplo is able to turn the **elimination of tensor-core-load instructions** into **performance improvements**.
- The oracle case (i.e., infinite-sized LHB) shows **26% speedup**.

Elimination of Redundant Tensor-Core Loads



- Duplo can **remove up to 76%** of redundant tensor-core-load instructions using the detection mechanism and LHB.
- About 3/4 of memory loads turn into simpler register renaming!
- With aggressive elimination of tensor-core loads, Duplo achieves **performance improvement** and **energy saving**.

Outline

- Introduction
- Background and Motivation
- Duplo Architecture
- Evaluation Results
- **Conclusion**

Conclusion

- Duplo saves a good portion of **off-chip DRAM transactions** and cache accesses and **turns them into in-SM register renaming**.
- With aggressive eliminations of tensor-core-load instructions, Duplo achieves **performance improvements** and **energy saving**.

Contact

- If you have any questions, please contact **Hyeonjin Kim** via hyeonjin_kim@yonsei.ac.kr. Thank you!

This presentation and recording belong to the authors.
No distribution is allowed without the authors' permission.