13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2022)

Optimizing Half-precision Winograd Algorithm on ARM Many-core Processors

Dedong Xie
University of Toronto

Zhen Jia
Amazon Web Services

Zili Zhang
Peking University

Xin Jin
Peking University

Presenter：Dedong Xie

Date：2022.08.24

CONTENTS

Introduction

Introduction

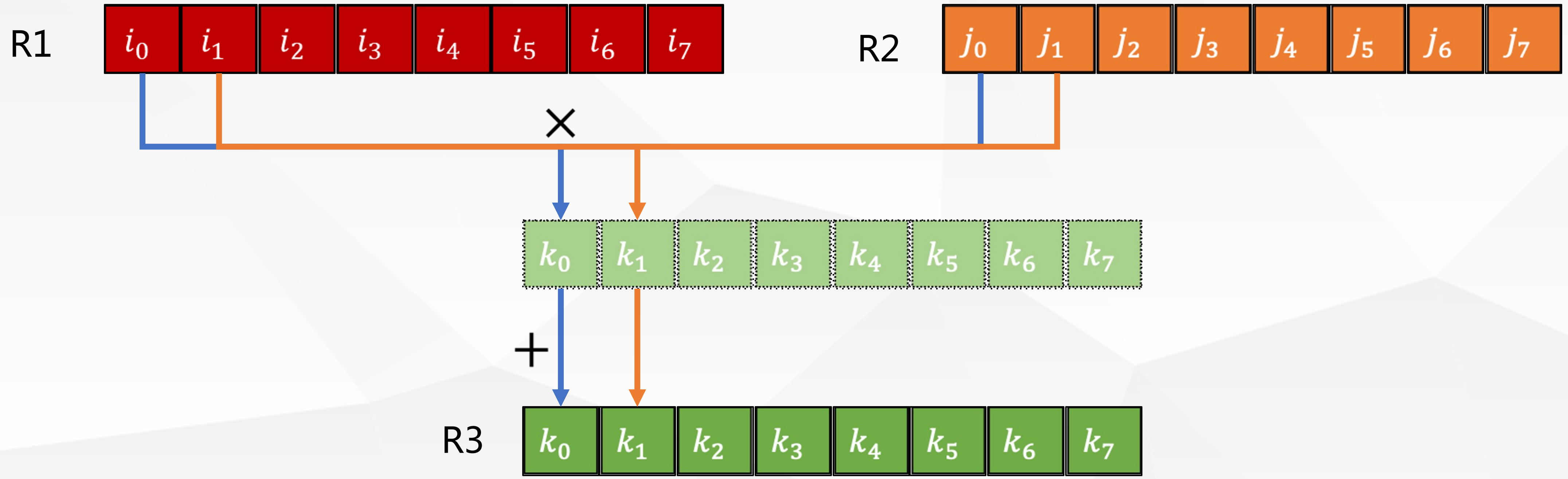
- Convolutional Neural Networks
  - Successful in recognition and recommendation
  - E.g., VGG, ResNet
- CPU
  - Vector instructions
  - High performance
  - GPU shortage
  - Attractive choice for certain use cases (e.g., inference)

Graviton CPU

- Fast, efficient, better price performance
- ARM NEON SIMD ISA
- Opportunity for optimizations for CNN

ARM NEON Vector Instructions

- Each Vector register of size 128bits, contains 8 lanes of FP16 data, compute 8 lanes in one instruction
- Example: Fused Multiply-Add, FMLA R3, R1, R2

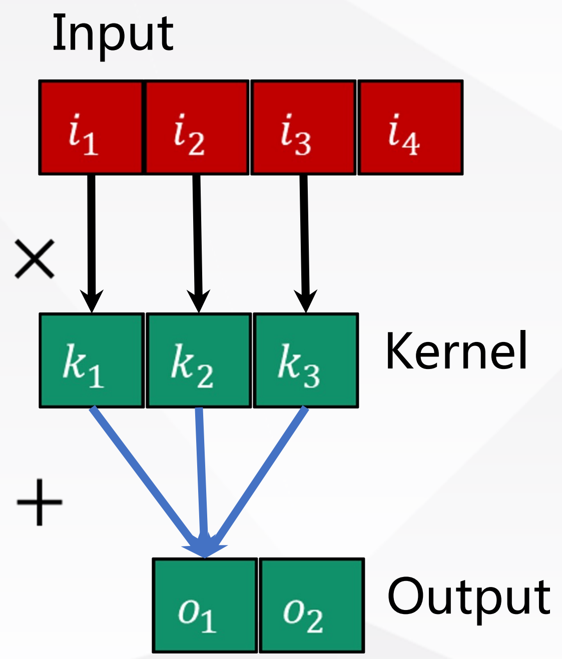


R1
i_0 i_1 i_2 i_3 i_4 i_5 i_6 i_7
R2
j_0 j_1 j_2 j_3 j_4 j_5 j_6 j_7
$\times$
k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7
$+$
R3
k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7

Convolution

- Input: i_1, i_2, i_3, i_4 Kernel: k_1, k_2, k_3 Output: o_1, o_2
- Direct Convolution:
- $o_1 = i_1 k_1 + i_2 k_2 + i_3 k_3, o_2 = i_2 k_1 + i_3 k_2 + i_4 k_3$

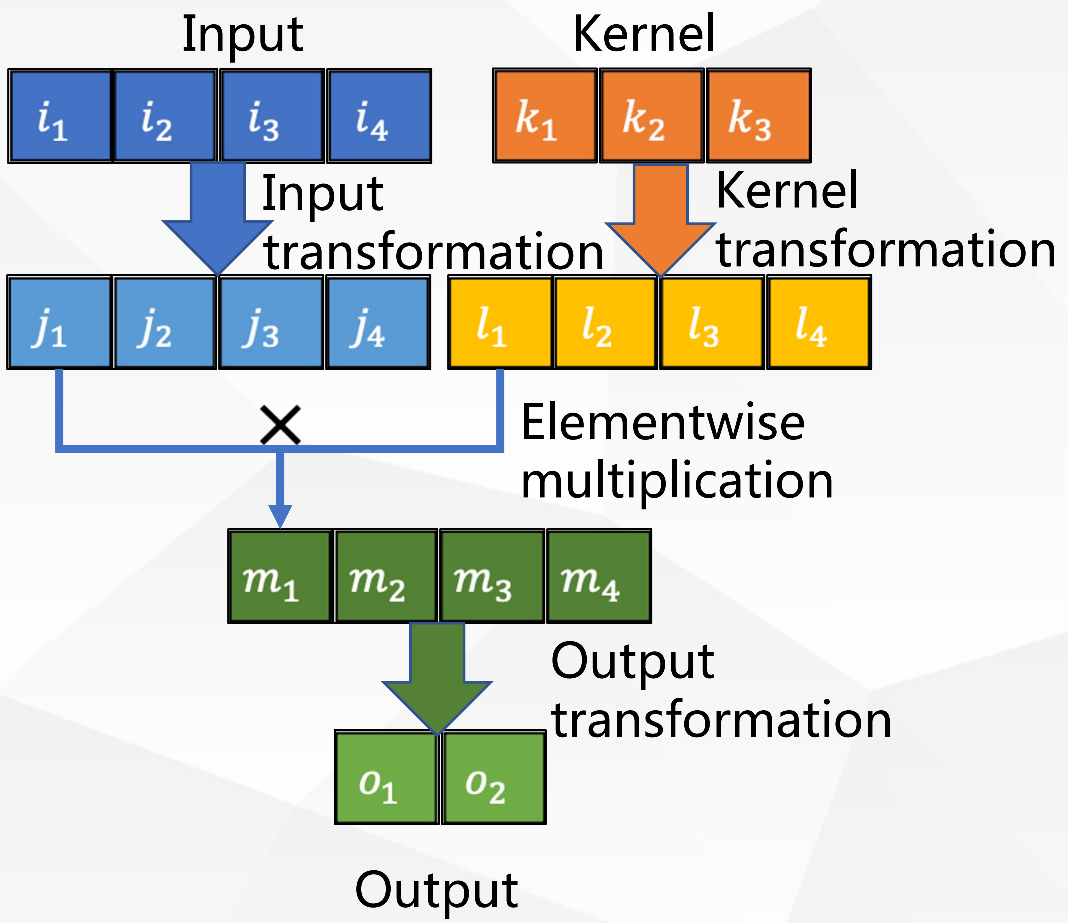
- A total of mr multiplications needed for F(m,r)



Convolution F(2,3)

Winograd Algorithm for Convolution

- Take intermediate values
- A total of m+r+1 multiplications needed for F(m,r)



$$j_1 = i_1 - i_3, j_2 = i_2 + i_3,$$
$$j_3 = i_3 - i_2, j_4 = i_2 - i_4$$
$$l_1 = k_1, l_2 = \frac{k_1 + k_2 + k_3}{2},$$
$$l_3 = \frac{k_1 - k_2 + k_3}{2}, l_4 = k_3$$
$$m_1 = j_1 l_1, m_2 = j_2 l_2,$$
$$m_3 = j_3 l_3, m_4 = j_4 l_4$$
$$\begin{aligned} o_1 &= m_1 + m_2 + m_3 \\ &= k_1 i_1 - k_1 i_3 + k_2 i_2 + k_1 i_3 + k_3 i_3 \\ &= k_1 i_1 + k_2 i_2 + k_3 i_3 \end{aligned}$$
$$\begin{aligned} o_2 &= m_2 - m_3 - m_4 \\ &= k_1 i_2 + k_3 i_2 + k_2 i_3 - k_3 i_2 + k_3 i_4 \\ &= k_1 i_2 + k_2 i_3 + k_3 i_4 \end{aligned}$$

Winograd Algorithm

Input Transformation → Elementwise Multiplication → Output Transformation

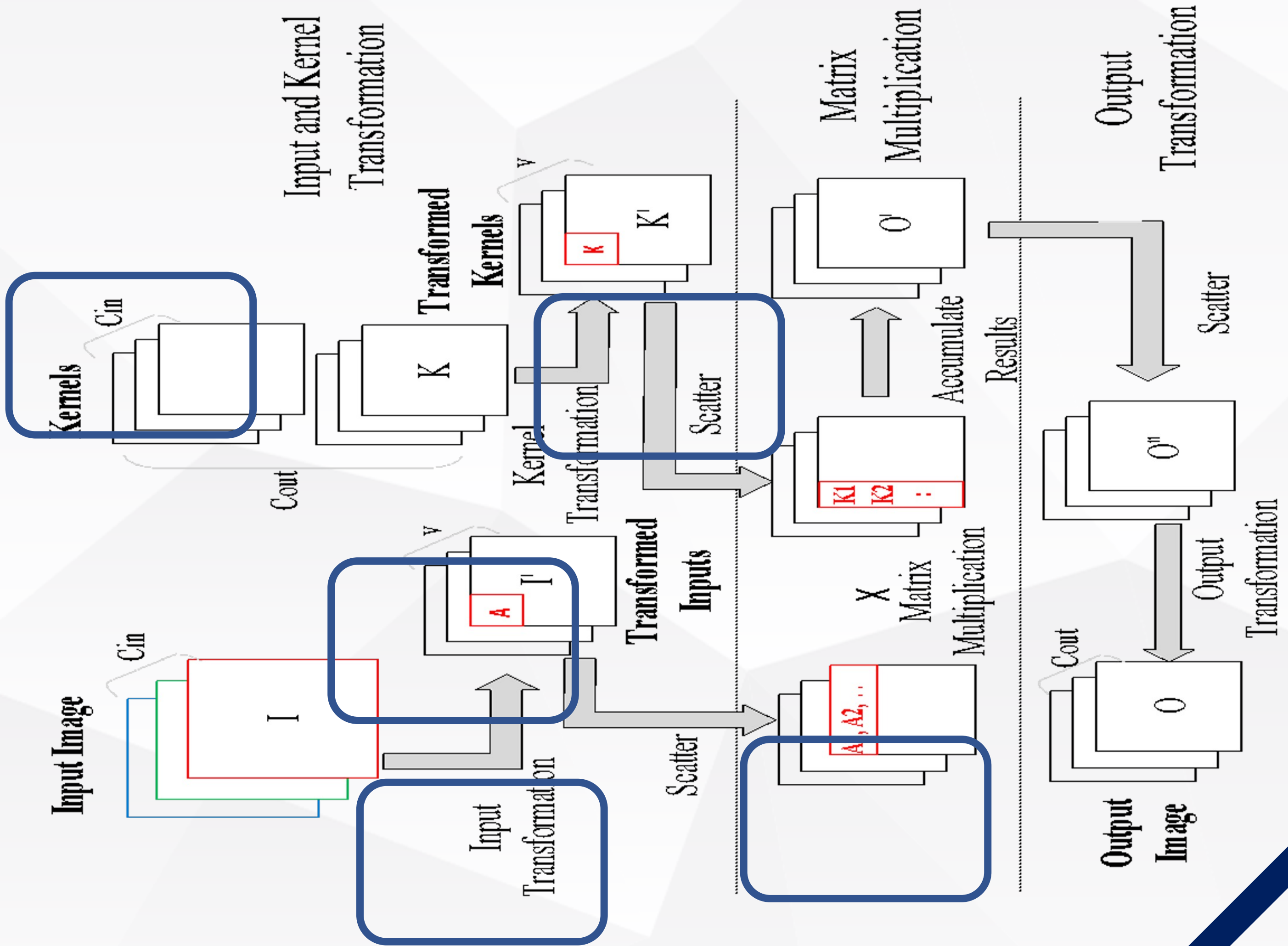
$$O = C[AI \odot BK]$$

Design

HAWC: Design

We present HAWC, Half-precision Winograd algorithm convolution for ARM many-core processors.

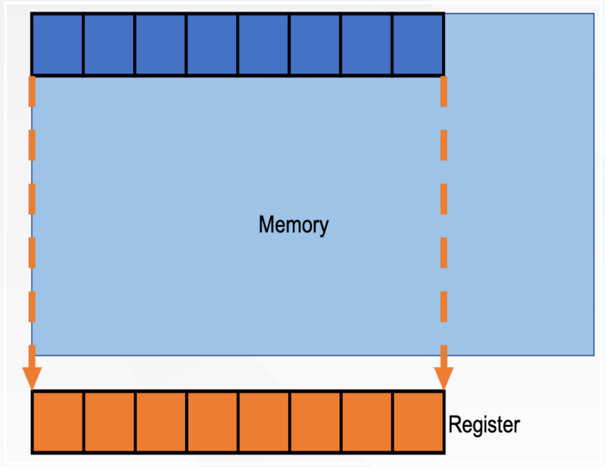
Circled parts are where we apply specific optimizations



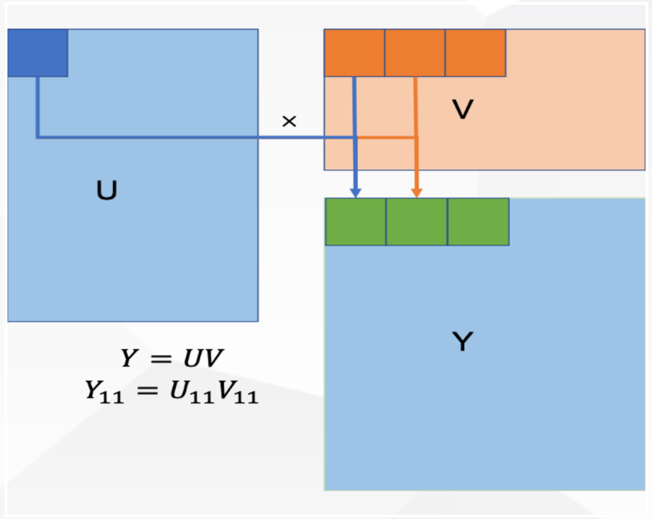
HAWC: Main Components

- Data Layout
- GEMM Kernel Generator
- Scatter Store
- Parallelization

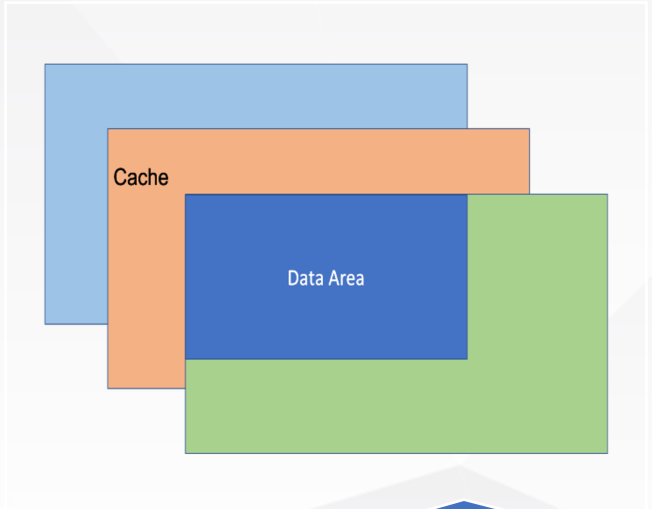
HAWC: Data Layout



Apply vectorization



Maximize data re-use



Reduce access overhead

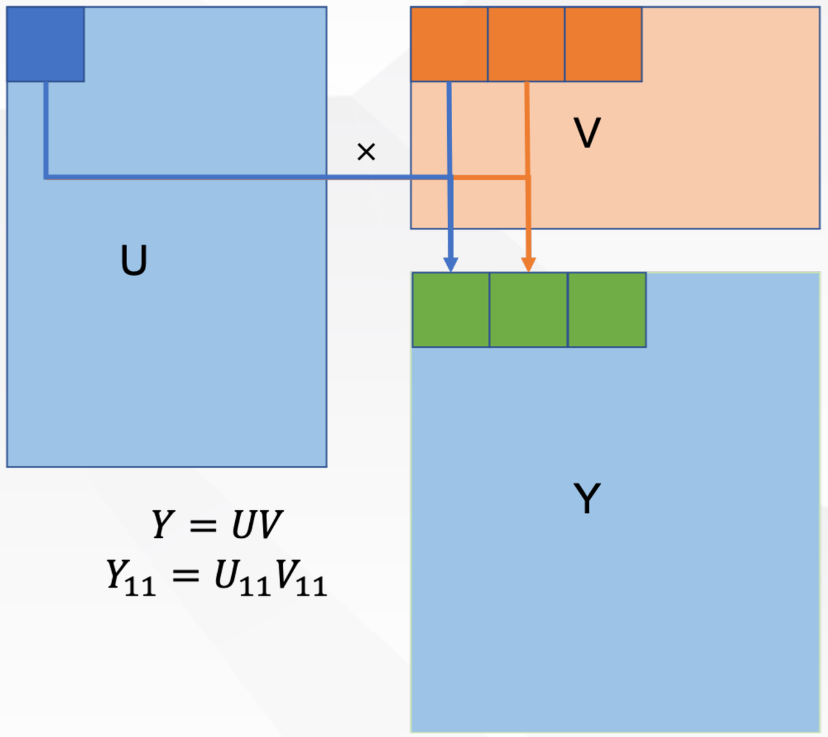
HAWC: Optimizations for Transformations

- Pre-defined transformation codelets for input, kernel, and output transformations
- Use of NEON intrinsics
- Use of C++ template
- Scattered store for matrix multiplication

```
#include <arm_neon.h>

template <long_t M, long_t R, long_t OS, long_t IS>
inline __attribute__((always_inline))
typename std::enable_if<(M + R - 1) == 4>::type
transform_image(float16x8_t* __restrict out, float16x8_t* __restrict in) {
    out[0]              = vsubq_f16(in[0], in[IS * 2]);
    out[OS * 1]         = vaddq_f16(in[IS], in[IS * 2]);
    out[OS * 2]         = vsubq_f16(in[IS * 2], in[IS]);
    out[OS * 3]         = vsubq_f16(in[IS * 3], in[IS]);
}
```

HAWC: GEMM Kernel Generator

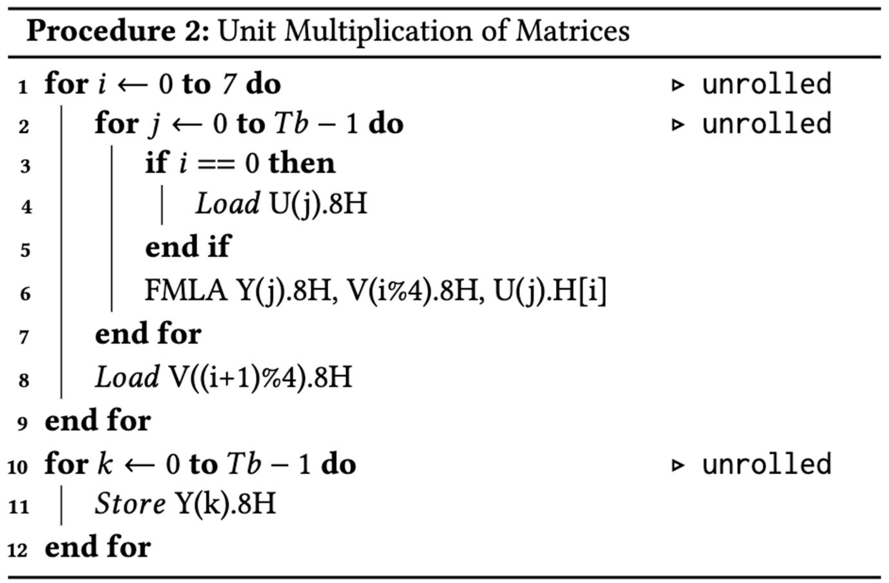


$$Y = UV$$
$$Y_{11} = U_{11}V_{11}$$

Fast

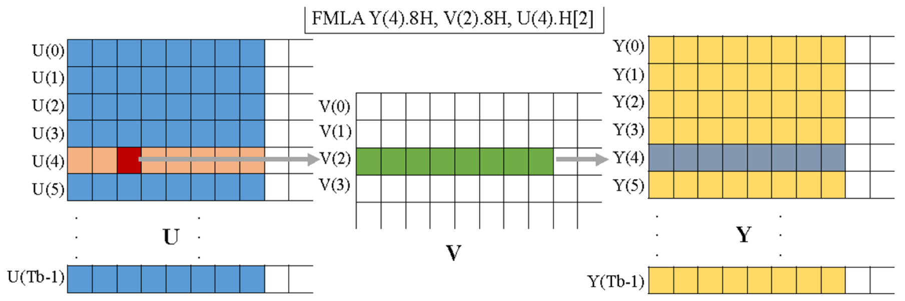
Flexible

Adaptive



Procedure 2: Unit Multiplication of Matrices

```
for $i \leftarrow 0$ to $7$ do                          ▷ unrolled
    for $j \leftarrow 0$ to $Tb-1$ do                  ▷ unrolled
        if $i == 0$ then
            Load U(j).8H
        end if
        FMLA Y(j).8H, V(i%4).8H, U(j).H[i]
    end for
    Load V((i+1)%4).8H
end for
for $k \leftarrow 0$ to $Tb-1$ do                      ▷ unrolled
    Store Y(k).8H
end for
```



🟦 Transformed inputs in register

🟩 Vector register being multiplied

🟨 Registers accumulate products

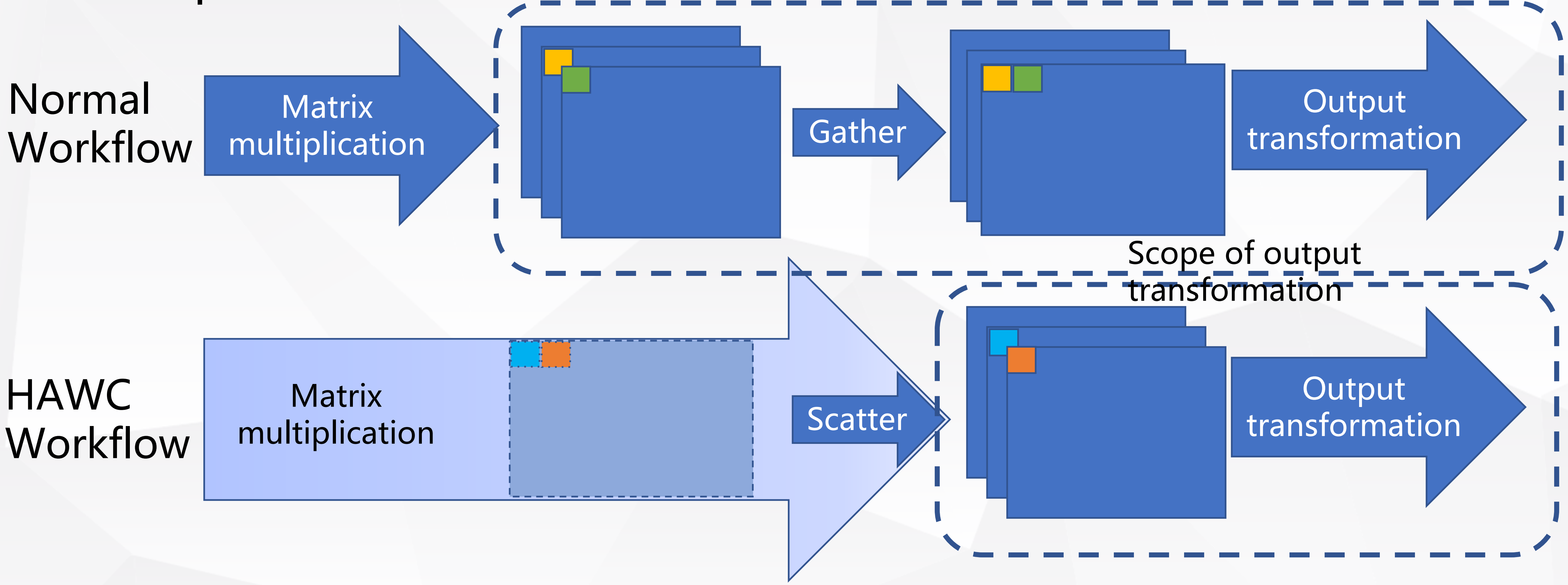
🟧 Vector register contains the broadcasted element

🟥 Broadcasted element as the third operand of FMLA

⬛ Register accumulates results from this FMLA

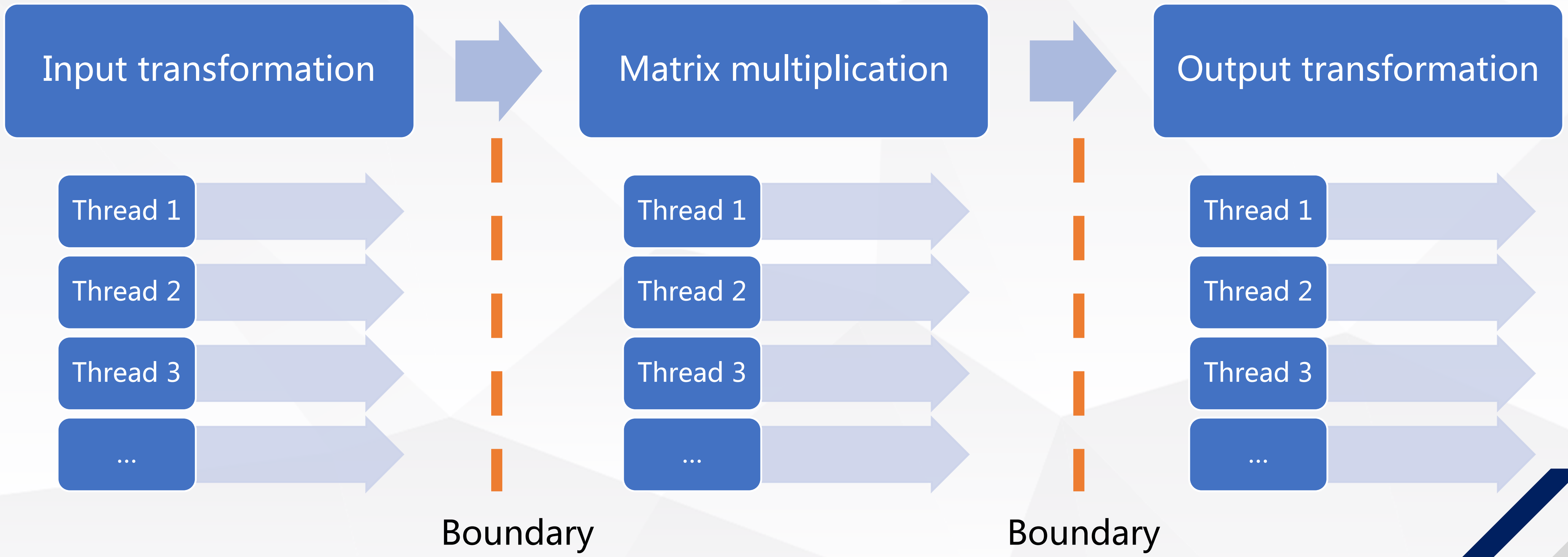
HAWC: Scattered Store

- After matrix multiplication, an inverse transformation is needed to rebuild elementwise multiplied results to apply output transformation.



HAWC: Parallelization

- Minimal parallel scheduler, parallel in each stage



Implementation

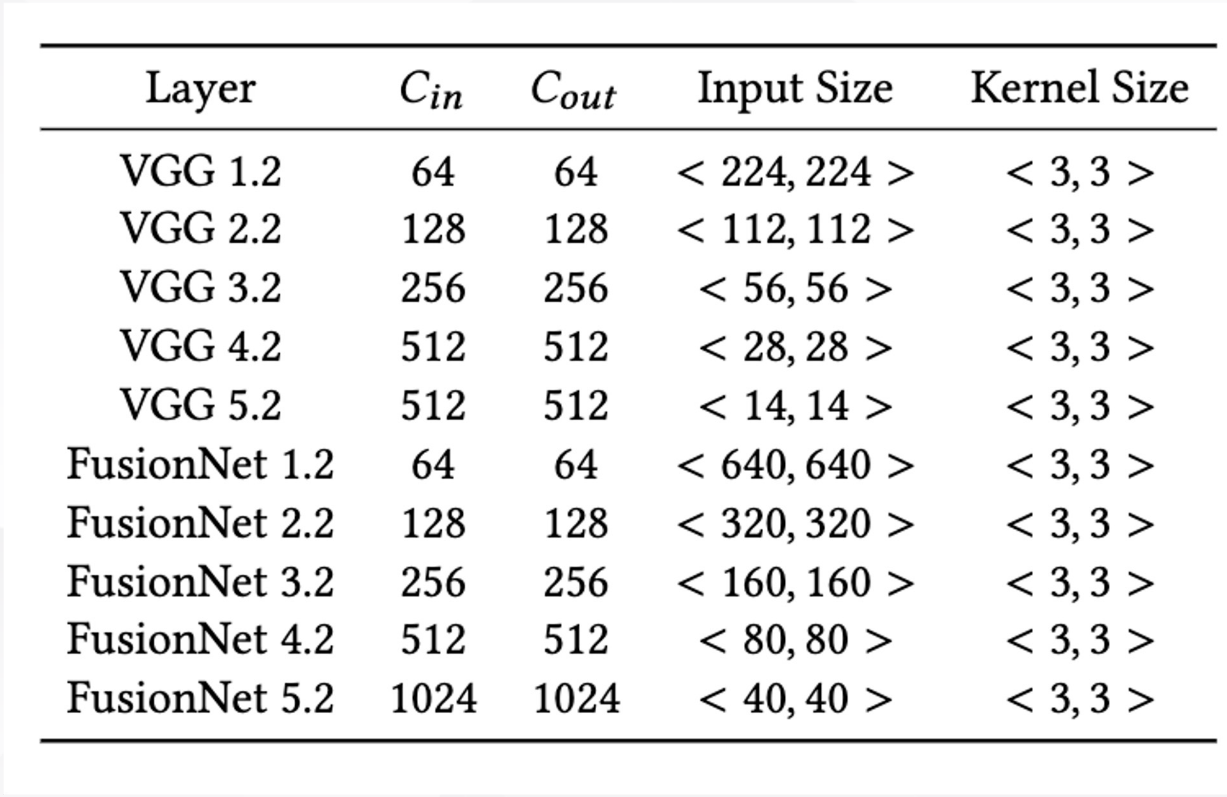
- Implemented in C++
- Target ARM CPU with FP16 ASIMD support
- Rely on ARM Compiler Toolchain
- Compile using GCC(g++)
- Build with Make

Experiments
&
Analysis

Experiments Setup

- Amazon EC2 m6g.metal instance
- Graviton 2 64 cores
- Ubuntu 18.04

- Compare latency on representative layers of CNN models
- Compare with NCNN and MNN



Layer	C_{in}	C_{out}	Input Size	Kernel Size
VGG 1.2	64	64	$< 224, 224 >$	$< 3, 3 >$
VGG 2.2	128	128	$< 112, 112 >$	$< 3, 3 >$
VGG 3.2	256	256	$< 56, 56 >$	$< 3, 3 >$
VGG 4.2	512	512	$< 28, 28 >$	$< 3, 3 >$
VGG 5.2	512	512	$< 14, 14 >$	$< 3, 3 >$
FusionNet 1.2	64	64	$< 640, 640 >$	$< 3, 3 >$
FusionNet 2.2	128	128	$< 320, 320 >$	$< 3, 3 >$
FusionNet 3.2	256	256	$< 160, 160 >$	$< 3, 3 >$
FusionNet 4.2	512	512	$< 80, 80 >$	$< 3, 3 >$
FusionNet 5.2	1024	1024	$< 40, 40 >$	$< 3, 3 >$

NCNN

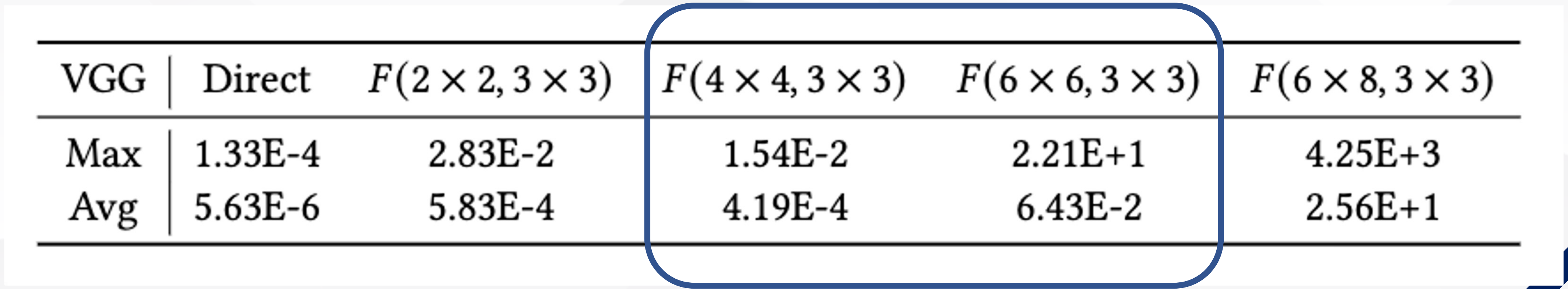
MNN

Source:
https://github.com/Tencent/ncnn

Source:
https://github.com/alibaba/MNN

Accuracy

- Winograd algorithm has mathematical instability
- For F(m,r), higher m will yield less operations with lower accuracy (m: Hyper parameter r: Kernel size)
- Calculate maximum element error and average element error
- Average error of less than E-2 will not influence stability*

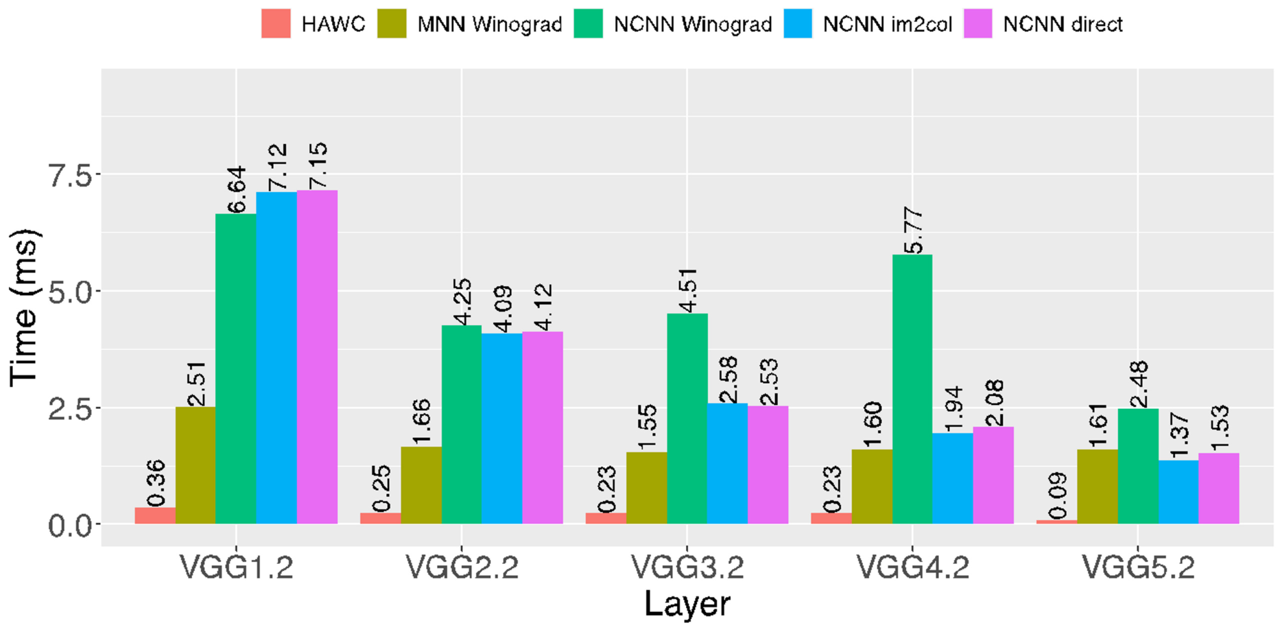


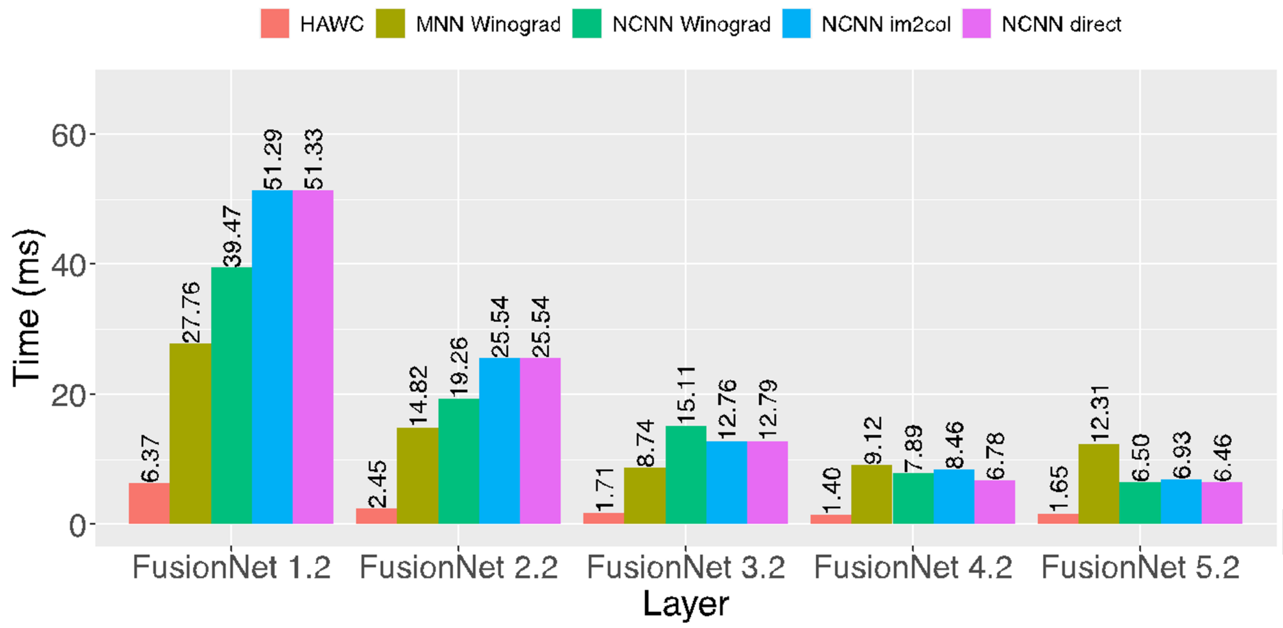
VGG	Direct	$F(2\times 2, 3\times 3)$	$F(4\times 4, 3\times 3)$	$F(6\times 6, 3\times 3)$	$F(6\times 8, 3\times 3)$
Max	1.33E-4	2.83E-2	1.54E-2	2.21E+1	4.25E+3
Avg	5.63E-6	5.83E-4	4.19E-4	6.43E-2	2.56E+1

*: Gupta et al. 2015. Deep Learning with Limited Numerical Precision. ICML' 15.

Performance

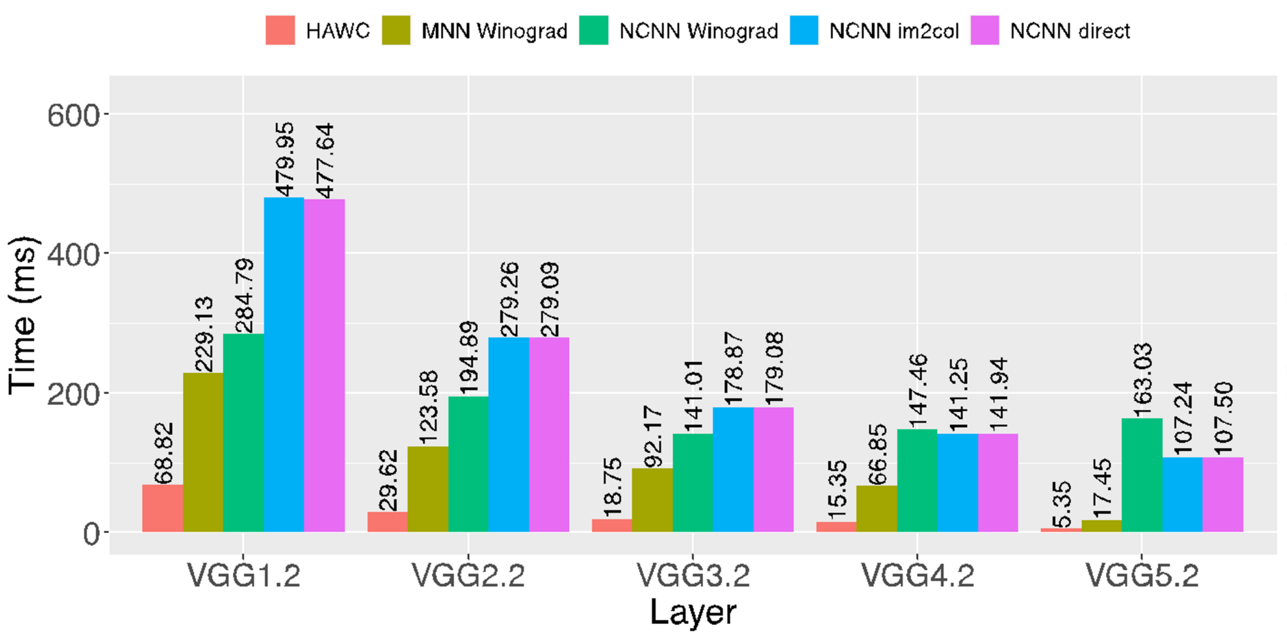
- On average: 10.74× speedup
- Up to: 27.56× speedup

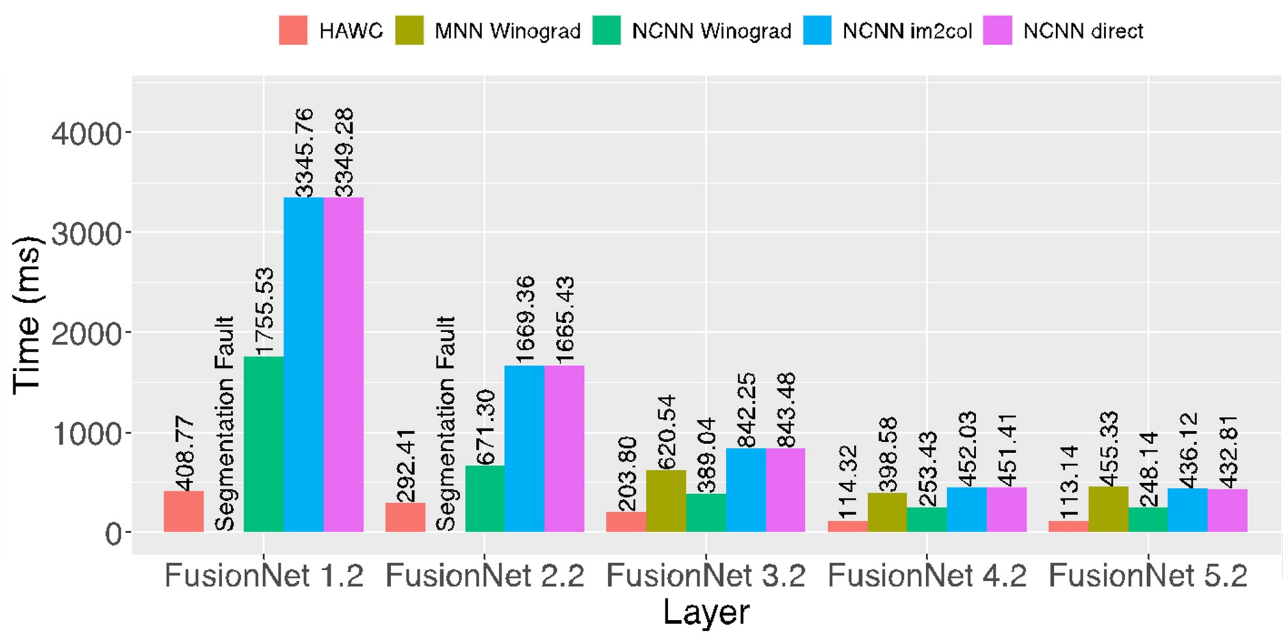




Multi-Batch Performance

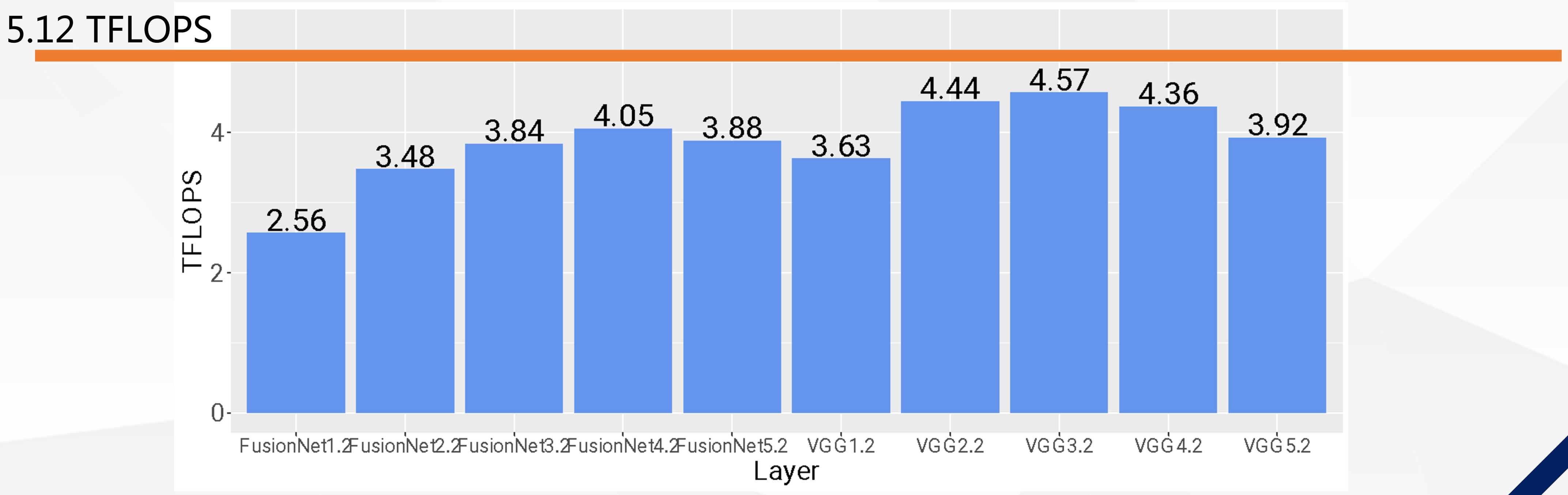
- On average: 5.45× speedup
- Up to: 30.47× speedup





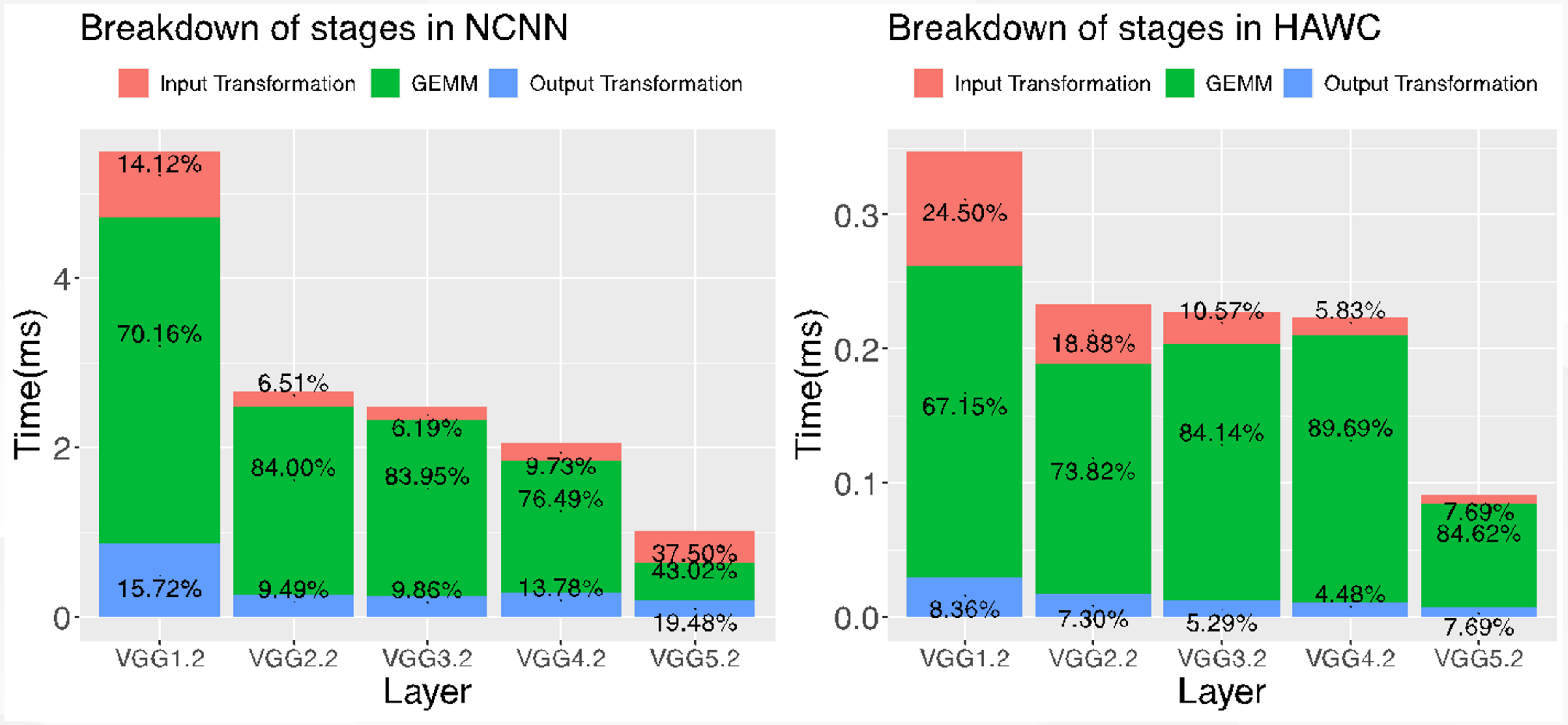
GEMM Performance

Achieves ~70%-90% of theoretical maximum TFLOPS



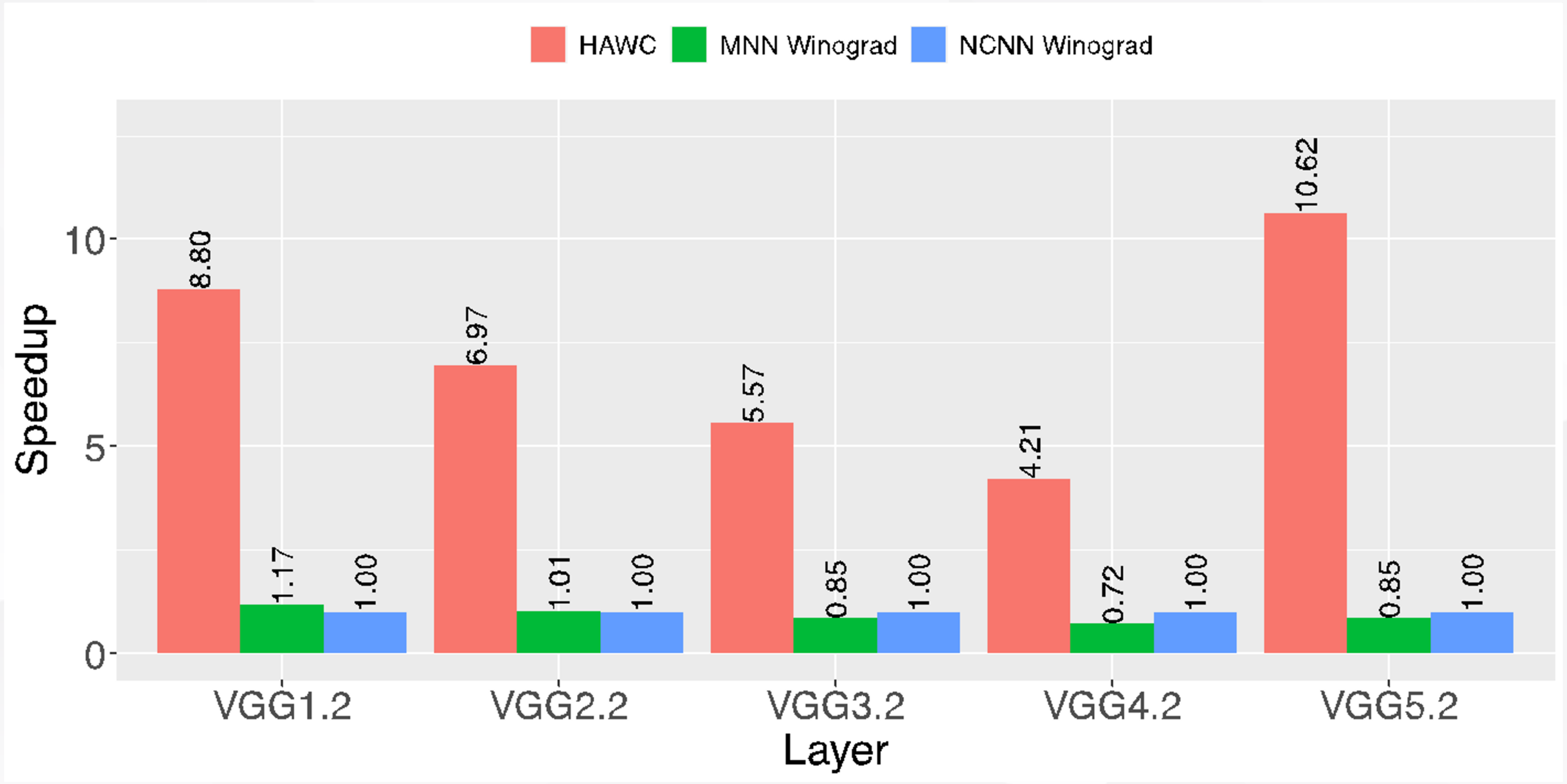
Computation Time Breakdown

Our design of scattered store saves time used by output transformation



Case Study: Graviton 3

- AWS Graviton 3 instance is released in May 2022, with new features.



Conclusion

Contributions

HAWC

- Efficient implementation of FP16 Winograd convolution optimized for ARM many-core processors.

Design

- Apply various optimizations.
- A custom JIT-compiled matrix multiplication kernel for Winograd convolution for ARM NEON ISA.

Performance

- HAWC achieves on average 10.74× and up to 27.56× speedup by experiments.

Future work

- Autotune selection of GEMM parameters
- Longer vector registers: 256bits, 512bits,...
- Different data type: BF16, INT8,...

13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2022)
August 24th, 2022

Thank you

- Thank you for listening!
- Questions and comments?