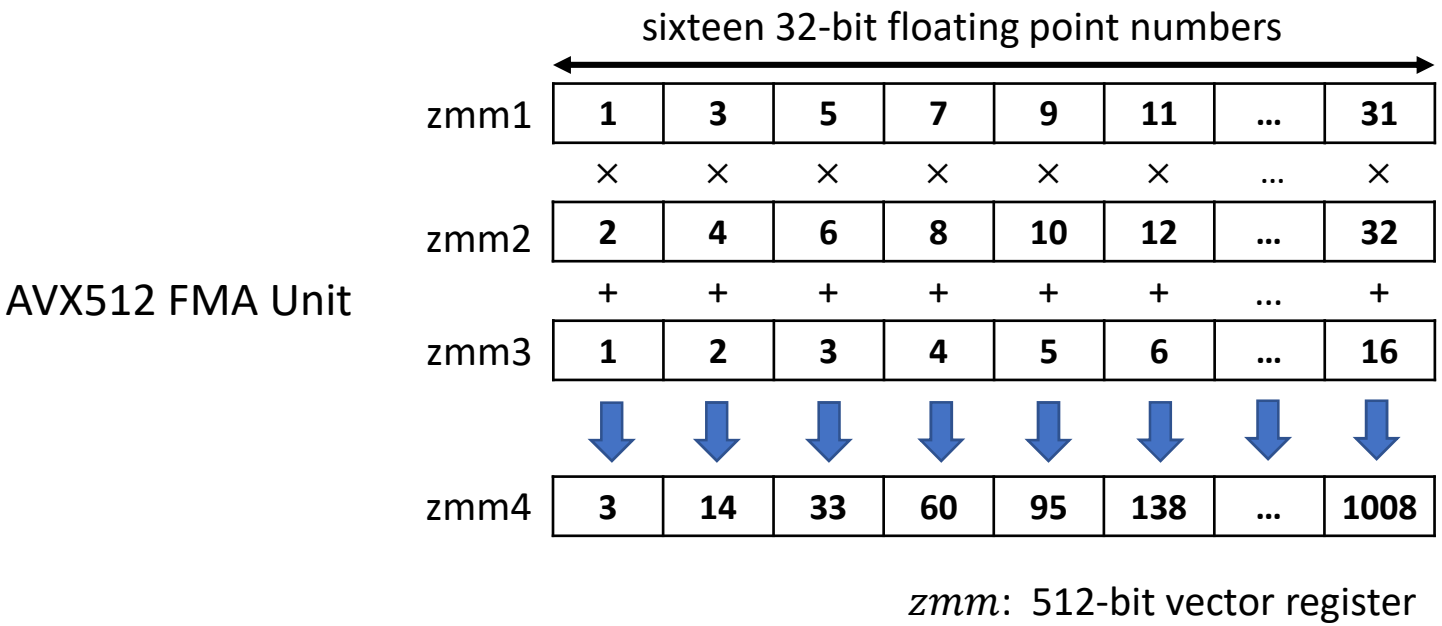
An Efficient Multicore CPU Implementation for Convolution-Pooling Computation in CNN

Hiroki Kataoka, Kohei Yamashita, Yasuaki Ito, Koji Nakano
Hiroshima University

Akihiko Kasagi, Tsuguchika Tabaru
Fujitsu Laboratories Ltd.

Contribution

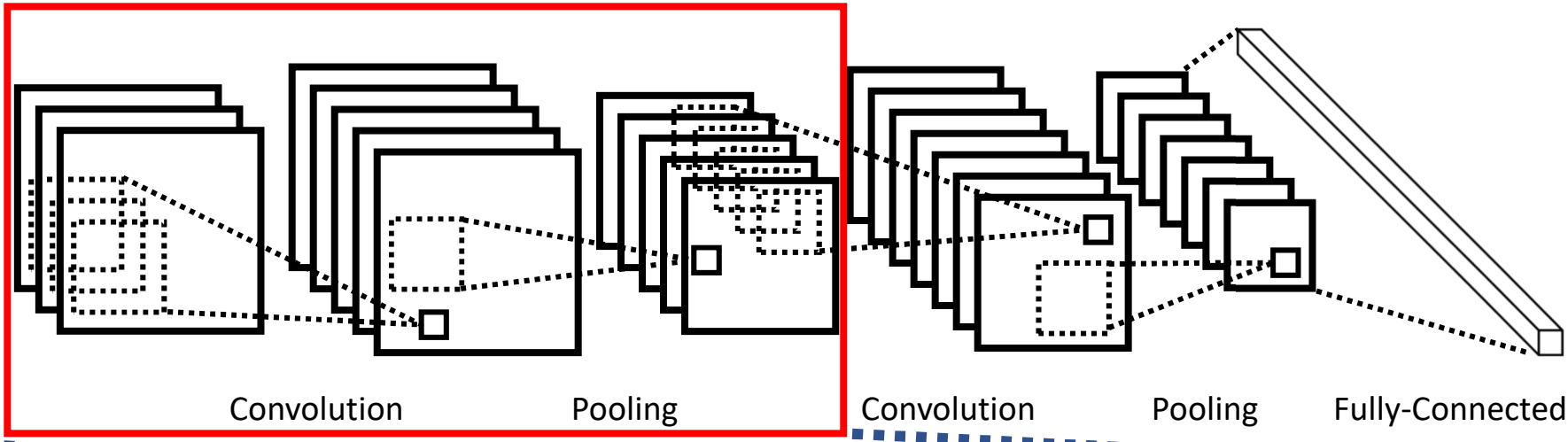
- We present an efficient implementation of the convolutional-pooling in the multicore CPU with AVX-512 FMA units.
  - We use convolution interchange to reduce the computational cost.
  - Our proposed method is up to 2.82 times faster than the multiple convolution and then pooling by DNNL.
  - We incorporate the proposed implementation into TensorFlow to perform them as a TensorFlow operation.



Convolutional Neural Network

Convolutional Neural Network (CNN) is commonly used for deep learning
CNN consists of some convolution operations and pooling operations

Our techniques accelerate



input

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

Convolution kernel

1	0	2
0	3	0
4	0	5

Convolution (Stride=1)

33	61	73	40
69	116	131	71
109	176	191	103
59	73	79	59

Pooling Kernel (average pooling)

0.25	**0.25**
0.25	**0.25**

Convolution (Stride=2)

output

69.75	78.75
104.25	108

Fused Filter algorithm (proposed)

The idea of the fused-filter method is to perform only one convolution to compute the convolution-pooling.

◆ Naïve algorithm

input

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

Convolution kernel

1	0	2
0	3	0
4	0	5

Convolution (Stride=1)

33	61	73	40
69	116	131	71
109	176	191	103
59	73	79	59

Pooling Kernel

0.25	**0.25**
0.25	**0.25**

Convolution (Stride=2)

output

69.75	78.75
104.25	108

◆ Fused Filter algorithm

input

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

FusedFilter

0.25	0.25	0.5	0.5
0.25	1	1.25	0.5
1	1.75	2	1.25
1	1	1.25	1.25

Convolution kernel

1	0	2
0	3	0
4	0	5

$\otimes$

Pooling Kernel

0.25	0.25
0.25	0.25

Convolution(Stride=2)

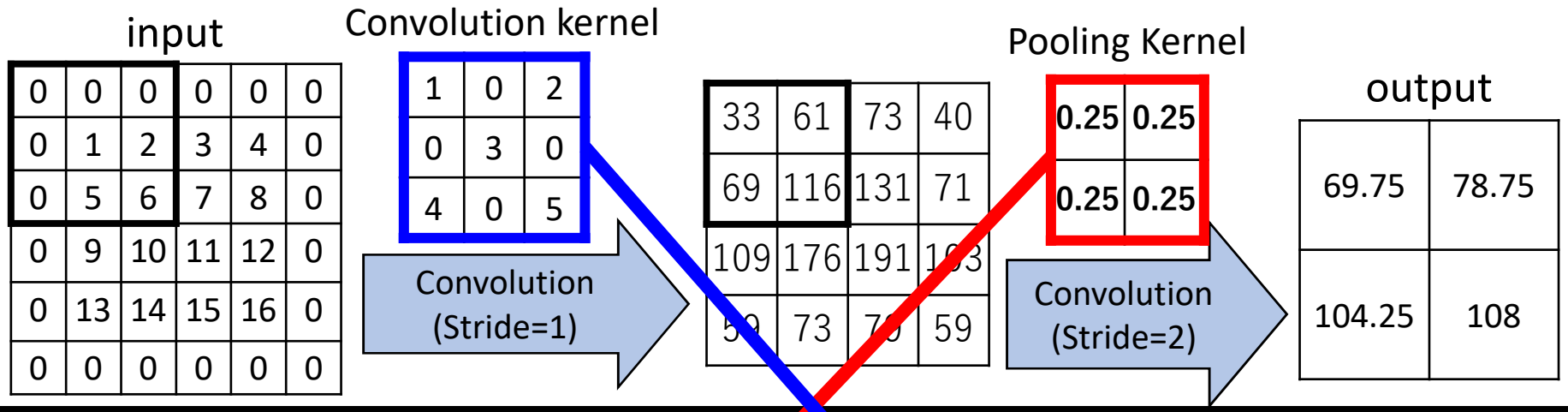
output

69.75	78.75
104.25	108

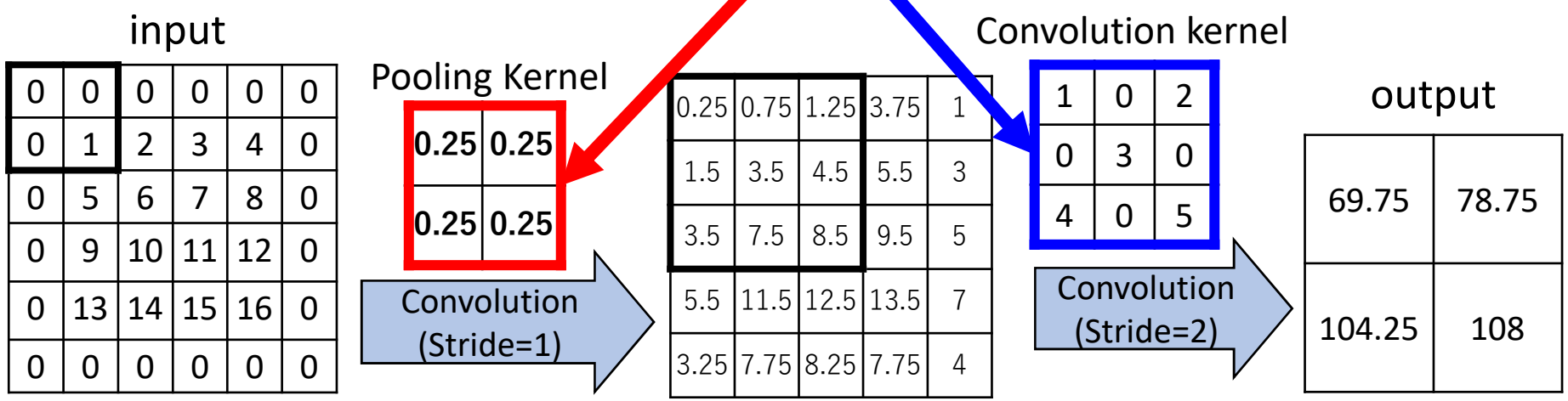
Direct Sum algorithm (proposed)

The idea of the direct-sum method is to use the convolution interchange technique to implement the convolution-pooling.

◆ Naïve algorithm



◆ Direct Sum algorithm



Direct Sum algorithm (proposed)

Instead of the 2x2 pooling, we perform pairwise-sum twice.

◆ Naïve algorithm

input

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

Convolution kernel

1	0	2
0	3	0
4	0	5

Convolution (Stride=1)

33	61	73	40
69	116	131	71
109	176	191	103
59	73	79	59

Pooling Kernel

0.25	**0.25**
0.25	**0.25**

Convolution (Stride=2)

output

69.75	78.75
104.25	108

◆ Direct Sum algorithm

input

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

0	0	0	0	0
1	3	5	7	4
5	11	13	15	8
9	19	21	23	12
13	27	29	31	16
0	0	0	0	0

1	3	5	7	4
6	14	18	22	8
14	30	33	38	12
22	46	50	54	16
13	27	29	31	16

Convolution kernel

1	0	2
0	3	0
4	0	5

Convolution (Stride=2)

279	315
417	432

$\times \frac{1}{4}$

output

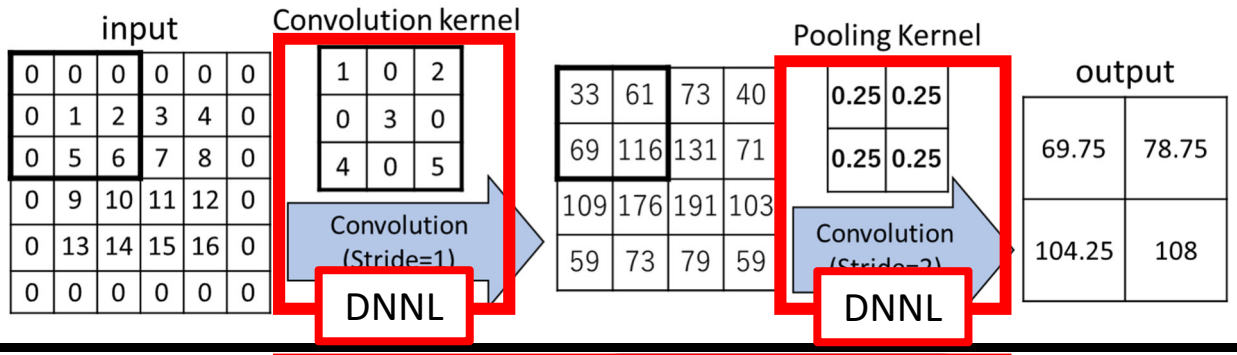
69.75	78.75
104.25	108

Proposed implementation

We adopt DNNL functions to perform convolution and pooling since the functions are well-optimized.
In our performance evaluation, we compare the following three algorithms.

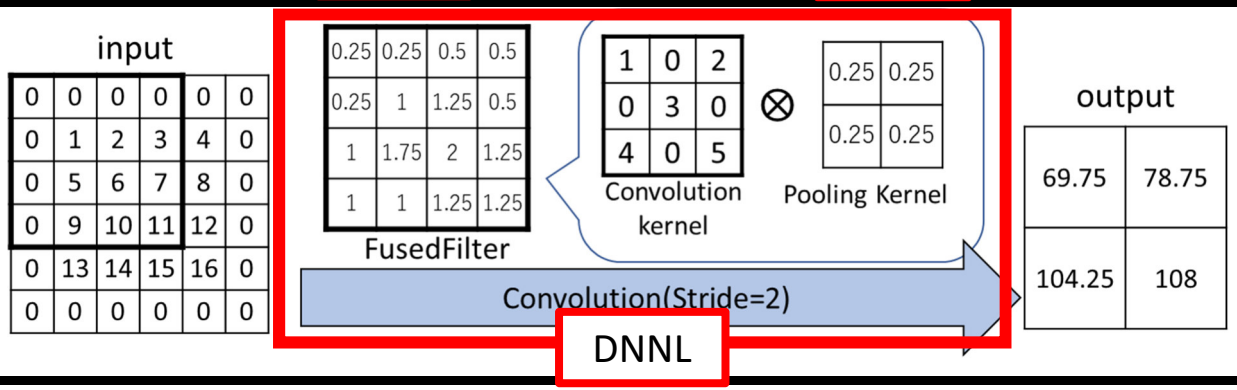
Naïve algorithm

Convolution and then pooling



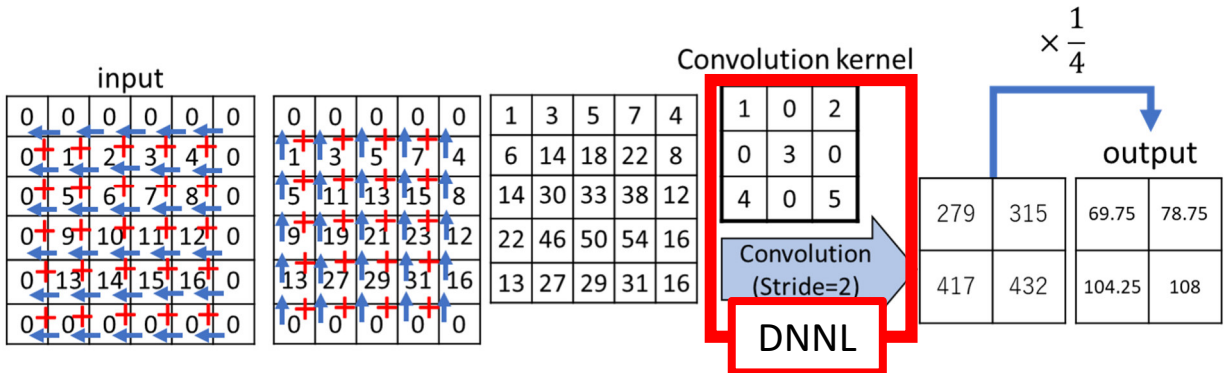
Fused Filter algorithm

Convolution with the fused filter



Direct Sum algorithm

Average pooling implemented by pairwise-sums in horizontal and vertical directions, and then convolution



Performance Evaluation

CPU: Core i9-7980XE (18 physical cores)
Maximum clock frequency: 4.4GHz(single core), 3.4GHz(18 cores)
OpenMP: version 5.0, **DNNL:** version 1.1.0

Computation time [ms]

input: 18×18, kernel: 3×3, pooling size: 2×2, batch: 32

		input channel, output channel				
		32,32	64,64	128,128	256,256	512,512
naïve (DNNL)	convolution	0.133	0.272	0.816	3.750	11.007
	pooling	0.018	0.032	0.067	0.152	0.309
	total	**0.151**	**0.304**	**0.883**	**3.902**	**11.316**
proposed (fused-filter)	convolution	**0.100**	**0.210**	**0.572**	**2.055**	**8.849**
	speed-up	1.510	1.448	1.544	1.899	1.279
proposed (direct-sum)	direct-sum	0.022	0.030	0.066	0.222	0.707
	convolution	0.064	0.166	0.363	1.162	4.940
	total	**0.086**	**0.196**	**0.429**	**1.384**	**5.647**
	speed-up	1.756	1.551	2.058	2.820	2.004

TensorFlow Incorporation

- We incorporate the proposed implementation into TensorFlow to perform them as a TensorFlow operation.

Computation time [ms]

input: 18×18, kernel: 3×3, pooling size: 2×2, batch: 32

input channel, output channel	32, 32	64, 64	128, 128	256, 256	512, 512
naïve	0.8296	1.6692	3.4670	9.1944	25.1610
incorporation	0.7047	1.1260	2.0238	4.0289	12.1624
speed-up	1.1772	1.4824	1.7131	2.2821	2.0687

CPU: Core i9-7980XE (18 physical cores)
Maximum clock frequency: 4.4GHz(single core), 3.4GHz(18 cores)
OpenMP: version 5.0, **DNNL:** version 1.1.0

Conclusion

- We presented an efficient implementation of the convolutional-pooling in the multicore CPU with AVX-512 FMA units.
  - Convolution interchange to reduce the computational cost.
  - Our proposed method is up to 2.82 times faster than the multiple convolution and then pooling by DNNL.
  - We incorporate the proposed implementation into TensorFlow to perform them as a TensorFlow operation.

Contact: yasuaki@cs.hiroshima-u.ac.jp