
AIDC
INTEL AI DEVCON 2018

UNDERSTANDING THE NEW VECTOR NEURAL NETWORK INSTRUCTIONS

Evarist Fomenko
Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN)
05/24/2018

OUTLINE

- Reduced precision inference
- VNNI instructions
- 8-bit convolution implementation

OUTLINE

- Reduced precision inference
- VNNI instructions
- 8-bit convolution implementation

REDUCED PRECISION INFERENCE

Data types for CNNs:

- Training: fp32, fp16, bfloat16, int16, …
- Inference: fp32, fp16, int8, …

Int8 vs fp32:

- Better performance (instruction throughput)
- Lower memory consumption (higher bandwidth, better cache usage)
- Acceptable accuracy loss

PERFORMANCE INT8 VS FP32

For Intel® processors with AVX512_BW support

- Standalone convolutions: ~1.5x (see next slide)
- Topology level

Topology	Batch size	FP32 img/s	INT8 img/s	Speed-up
ResNet-50	448	368	517	1.40x
SSD/VGG16	448	29	46	1.59x

Measured with Intel® Optimization for Caffe* (source: https://github.com/intel/caffe/wiki/Introduction-of-Accuracy-Calibration-Tool-for-8-Bit-Inference)

With new VNNI instructions the performance is expected to be higher

INT8 INFERENCE TECHNIQUES

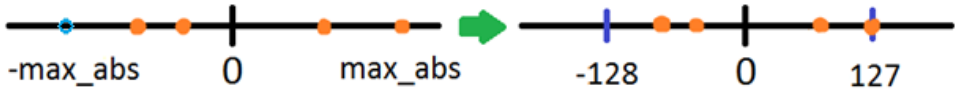
Int8 has lower dynamic range compare to fp32

Type	Dynamic range
fp32	$-3.4*10^{38}$ .. $3.4*10^{38}$
int8	0 .. 255 (-128 .. 127)

Quantization:

- $v \leftarrow \alpha \cdot i_8 + \beta$ (biased)

- $v \leftarrow \alpha \cdot i_8$ (symmetric)

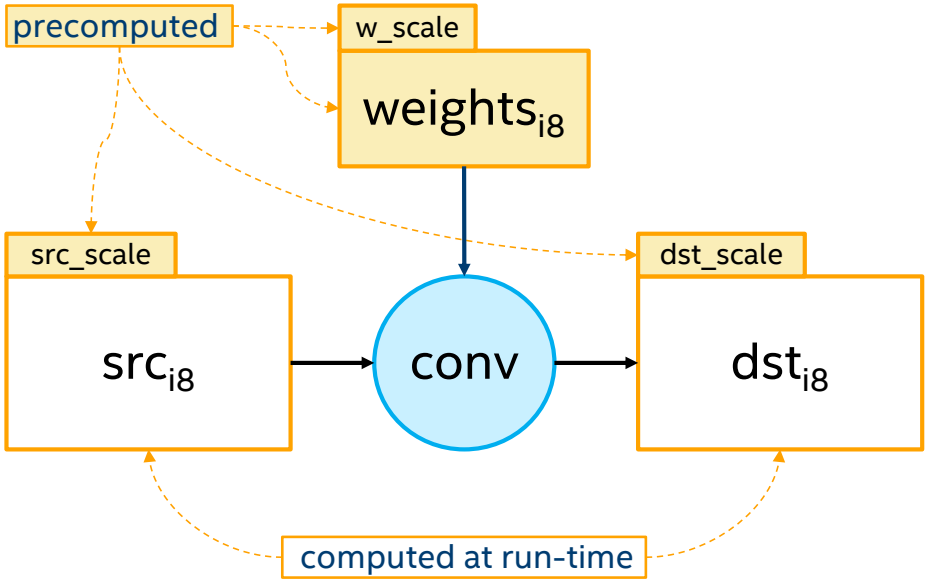


Typically there is a single scale $\alpha \sim \frac{\max(\{|v|\})}{128}$ per tensor $\{v\}$.

INT8 CONVOLUTION

Assumptions:

- Symmetric quantization
- Scales are precomputed
- Single scale per tensor



Computations:

1. $\gamma \leftarrow \frac{src\_scale \cdot w\_scale}{dst\_scale}$
2. $dst_{i8} \leftarrow \gamma \cdot conv_{i32}(src_{i8}, weights_{i8})$

OUTLINE

- Reduced precision inference
- VNNI instructions
- 8-bit convolution implementation

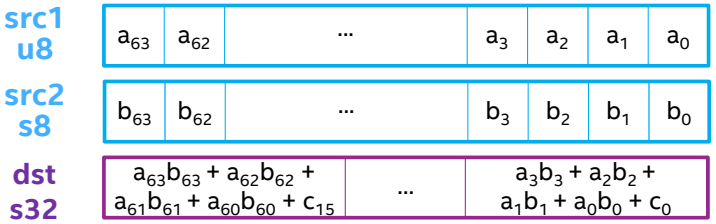
THE GOAL

$$conv_{i32}(src_{i8}, weights_{i8})$$

Key component: int8-multiplication with int32 accumulation

U8/S8 TO S32 MAC

```
for i = 0 .. 15
    for j = 0 .. 3
        dst32[i] += src1u8[4i + j] * src2s8[4i + j]
```



AVX-512 BW	VNNI

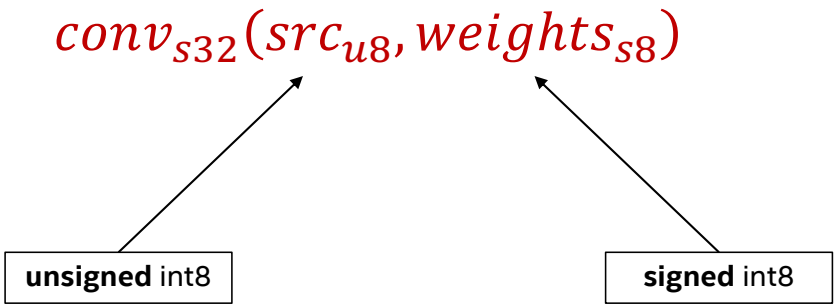
`vpmaddubsw zmm_s16, src1_u8, src2_s8`
`vpmaddwd zmm_s16, zmm_s16, [16 x 1_16]`
`vpaddd dst_s32, dst_s32, zmm_s16`
	`vpdpbusd dst_s32, src1_u8, src2_s8`

• 64 multiply-adds in 3 instructions • potential intermediate saturation after `vpmaddubsw` `saturate<s16>(u8*s8 + u8*s8)` • **1.33x** throughput of F32 equivalent (4x `vfmadd231ps`)	• 64 multiply-adds in 1 instruction • **no** potential intermediate saturation • src2 (s_8) operand can be broadcasted from memory • **4x** throughput of F32 equivalent (4x `vfmadd231ps`)

OUTLINE

- Reduced precision inference
- VNNI instructions
- 8-bit convolution implementation

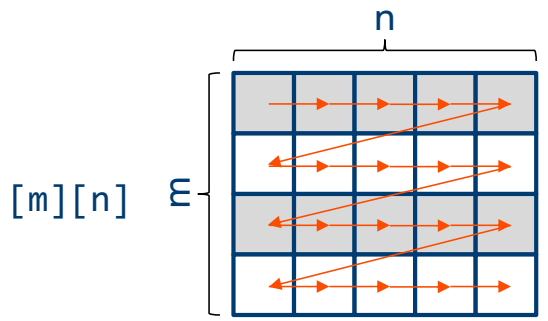
THE GOAL



CONVOLUTION AND LAYOUT

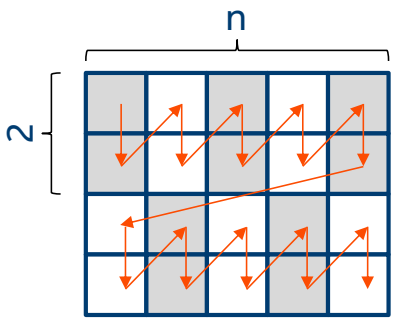
Naïve layout: nhwc / hwio

src_{u8} [mb][ih][iw][ic]
wei_{s8} [kh][kw][ic][oc]
---
dst_{s32} [mb][oh][ow] [oc]



Optimized layout: nhwc / hwIo4i

src_{u8} [mb][ih][iw][ic/4] [4ic]
wei_{s8} [kh][kw][ic/4][oc/16][16oc][4ic]
---
dst_{s32} [mb][oh][ow] [oc/16][16oc]



OPTIMIZED CONVOLUTION

Optimized layout: nhwc / hwIo4i

```
src_u8  [mb][ih][iw][ic/4]              [4ic]
wei_s8      [kh][kw][ic/4][oc/16][16oc][4ic]
-------------------------------------------
dst_s32 [mb][oh][ow]       [oc/16][16oc]
```

Innermost loop:

```
  16 x [4ic] <-- vpbroadcastd s[4ic]
  dst_s32    <-- vpdpbusd 16 x [4ic], [16oc][4ic]
```

Parallelization: mb, oh

```
. . .
vpbroadcastd zmm21, dword ptr [r11]
vpbroadcastd zmm22, dword ptr [r11+0x40]
vpbroadcastd zmm23, dword ptr [r11+0x80]
vpbroadcastd zmm24, dword ptr [r11+0xc0]
vmovups  zmm31, zmmword ptr [r12]
vpdpbusd zmm1, zmm21, zmm31
vpdpbusd zmm2, zmm22, zmm31
vpdpbusd zmm3, zmm23, zmm31
vpdpbusd zmm4, zmm24, zmm31
vmovups  zmm31, zmmword ptr [r12+0x2400]
vpdpbusd zmm6, zmm21, zmm31
vpdpbusd zmm7, zmm22, zmm31
vpdpbusd zmm8, zmm23, zmm31
vpdpbusd zmm9, zmm24, zmm31
. . .
```

IMPLEMENTATION IN INTEL MKL-DNN

Optimizations that are not shown:

- Even more blocking (weights format is OIhw**4i**16o**4i**)
- More parallelism (mb, oc/16, oh)
- Reuse sources and weights
- And more…

Check https://github.com/intel/mkl-dnn:

src/cpu/jit_avx512_core_u8s8s32x_conv*.cpp

LIMITATIONS

Intel MKL-DNN supports:

$$dst_{dt} \leftarrow \gamma \cdot conv_{s32}(src_{u8}, weights_{s8}), dt \in \{f_{32}, s_{32}, s_8, u_8\}$$

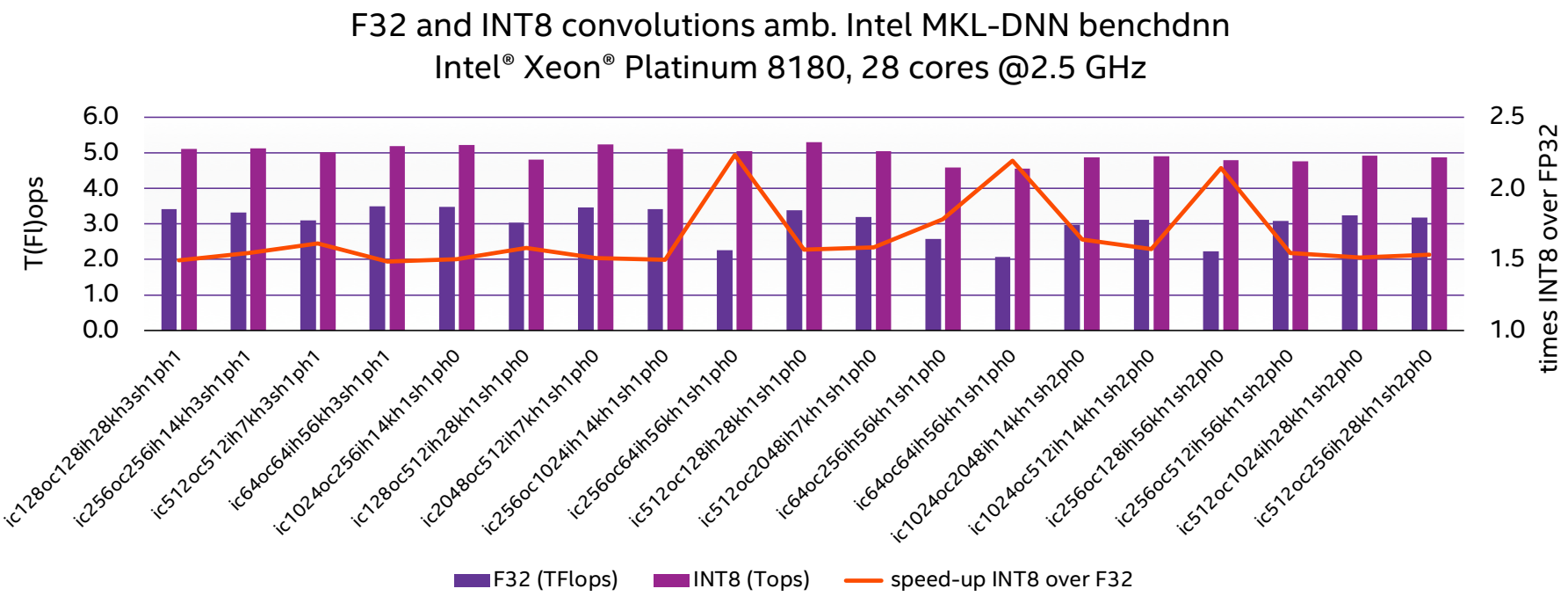
Source data must be non-negative

- Convolution after ReLU
  – all but first in AlexNet, ResNet-50, Inception-v3
- (Optional) implementation w/ compensation: $src_{s8} \rightarrow src'_{u8} - 128$

Number of channels must be a multiple of 16

- Quite common
  – all but first in AlexNet, ResNet-50, Inception-v3
- (Optional) implementation w/ padding

CONVOLUTIONS PERFORMANCE IN RESNET-50 BATCH 50

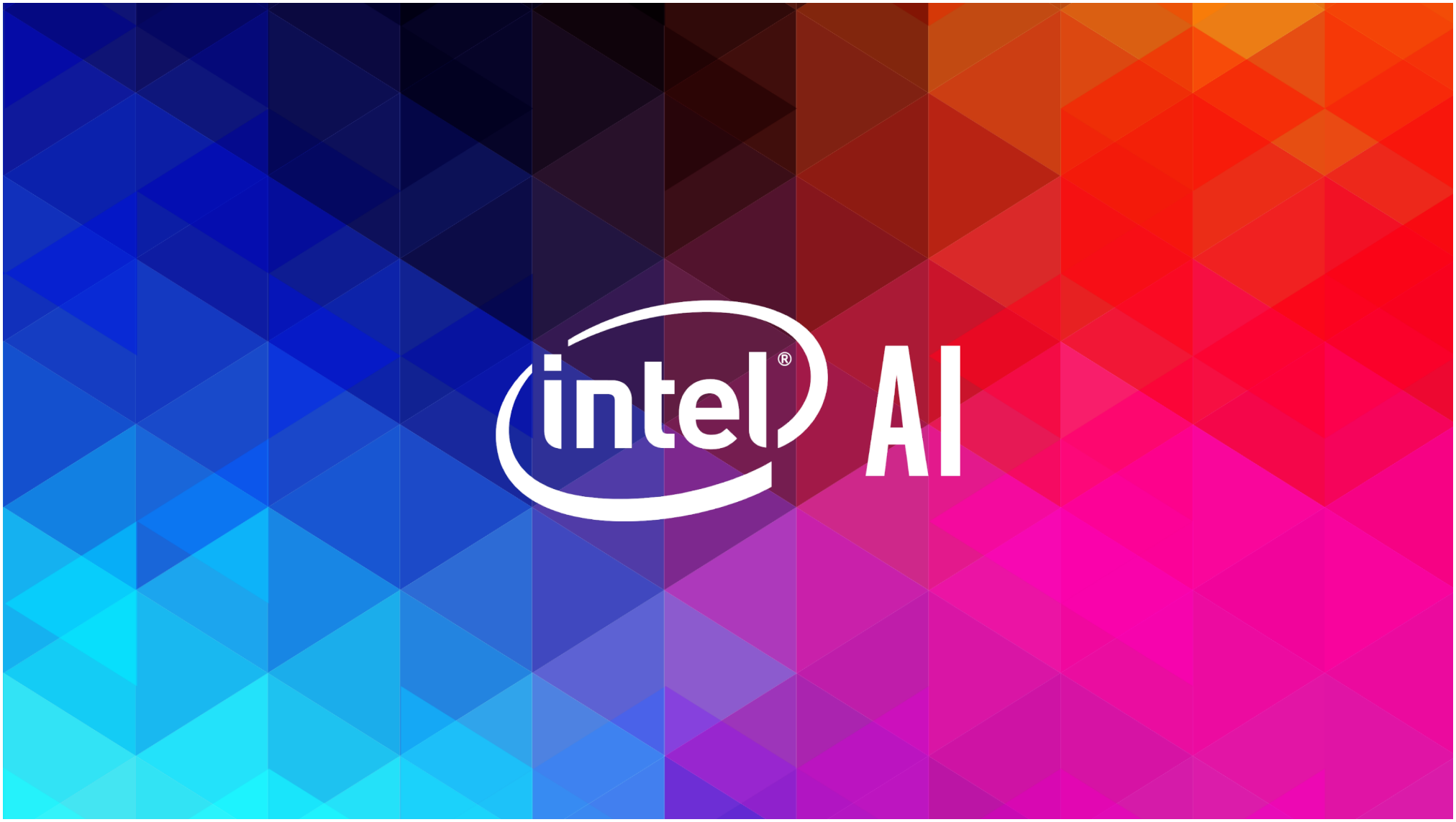


Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Configuration: Intel(R) Xeon(R) Platinum 8180 CPU @2.50 GHz (1 socket, 28 cores), HT disabled, Turbo enabled, 96 GB DDR4-2666 RAM, Red Hat Enterprise Linux Server release 7.2 (3.10.0-327.62.4.el7.x86_64). Intel MKL-DNN version v0.14. Environment variables: KMP_AFFINITY='granularity=fine, compact', OMP_NUM_THREADS=28. Performance measured with 'benchdnn --conv --mode=PERF --cfg=f32 ${prb} --cfg=u8s8u8s32 ${prb}'.

Optimization notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit: http://www.intel.com/reinformance. Source: Intel measured or estimated as of April 2018.


intel
AI

BACKUP

NEW VNNI INSTRUCTIONS (CONT'D)

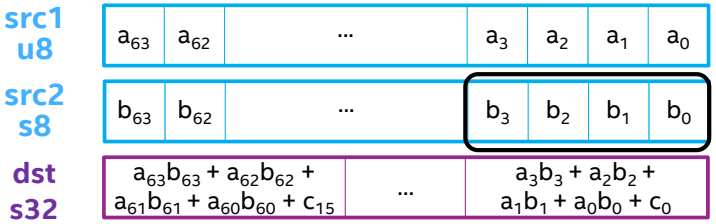
VPDPBUSD(S)

```
vpdpbusd  zmm0, zmm1, zmm2/m512/m32bcst
```

$s_{32} \leftarrow s_{32} + \sum_0^3 u_8 * s_8$

```
vpdpbusds zmm0, zmm1, zmm2/m512/m32bcst
```

$s_{32} \leftarrow \text{saturate}\langle s_{32}\rangle(s_{32} + \sum_0^3 u_8 * s_8)$



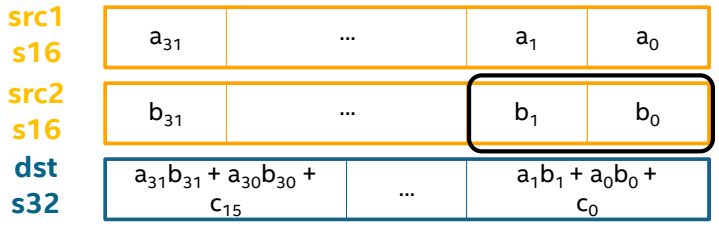
VPDPWSSD(S)

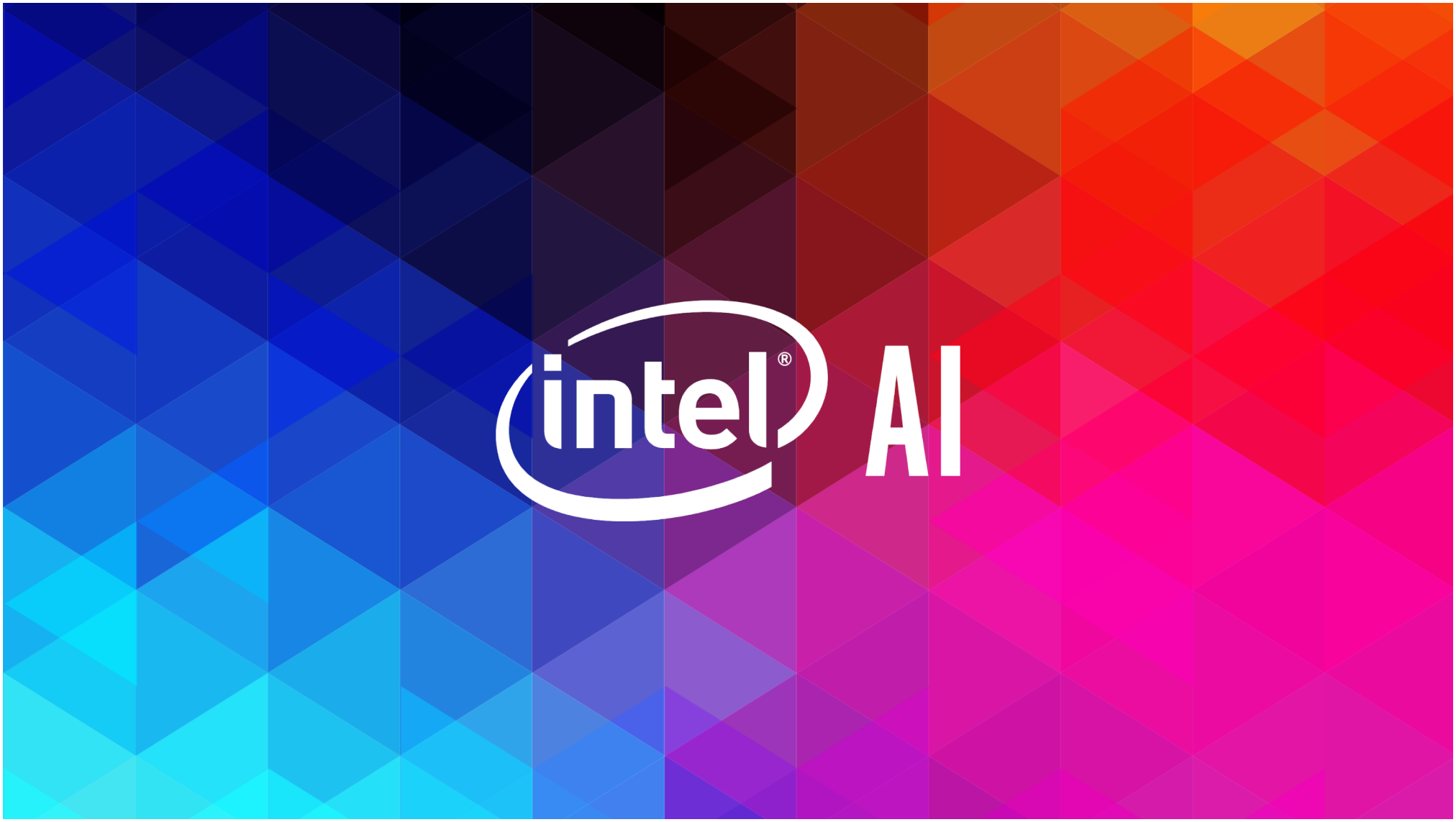
```
vpdpwssd  zmm0, zmm1, zmm2/m512/m32bcst
```

$s_{32} \leftarrow s_{32} + \sum_0^1 s_{16} * s_{16}$

```
vpdpwssds zmm0, zmm1, zmm2/m512/m32bcst
```

$s_{32} \leftarrow \text{saturate}\langle s_{32}\rangle(s_{32} + \sum_0^1 s_{16} * s_{16})$




intel
AI