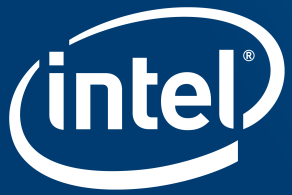


MATRIX Multiplication with Quantized matrices

Intel® Math Kernel Library (Intel® MKL) Team - Presenter: Murat Efe Guney

Workshop on Batched, Reproducible, and Reduced Precision BLAS

Georgia Tech, Atlanta

February 24, 2017

Acknowledgements

Benoit Jacob - Google*

Greg Henry and Peter Tang - Intel®

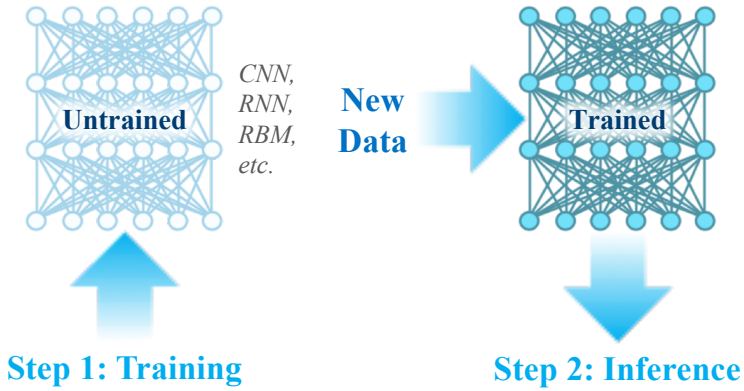
Why integer matrix multiplication?

Deep-learning applications rely on integer matrix-matrix multiplication

- GEMMLOWP*: speech/face recognition
- KALDI* speech recognition
- Inference using 8/16bit integer weights
- Open question: training with integers

Intel® Instruction Set Architecture (ISA) extensions for integer matrix operations

- AVX512_4VNNIW: Vector instructions for deep learning enhanced word variable precision
- https://software.intel.com/en-us/isa-extensions



Quantizing with integers

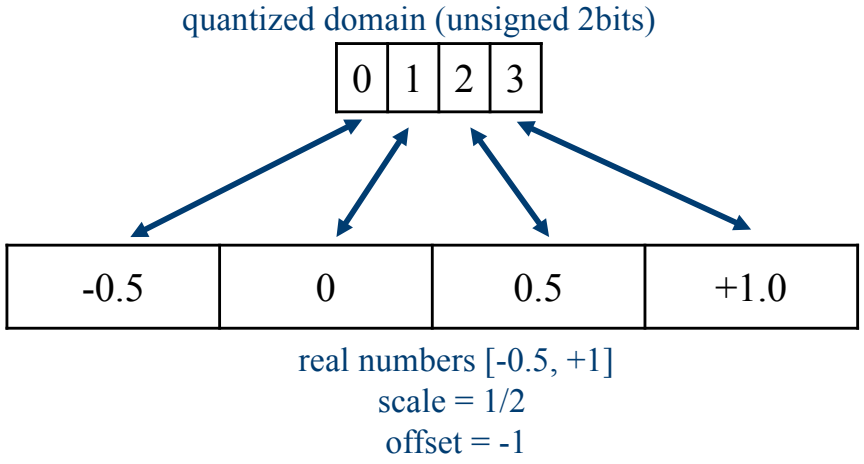
Integers instead of floating points for lower power/bandwidth and higher throughput

Integers to represent a real number range [a, b]

- real = **scale** * (integer + **offset**)

Why do we need an offset?

- Unsigned integers to represent signed real number domains, for e.g., [-0.5, +1]



Quantized matrices

The real number range [a, b] is known in advance

- Choose **scale** and **offset** values for each matrix accordingly
- Sacrifice dynamic range

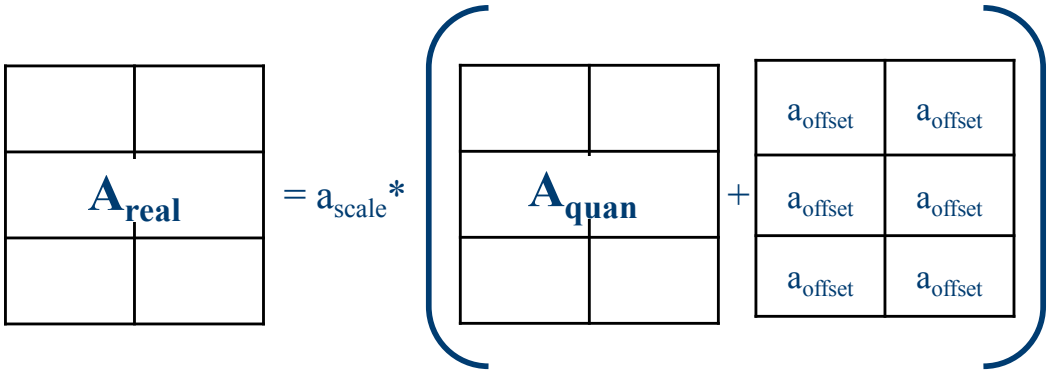
Zero point (0) must be represented perfectly in the quantized domain

- Zero-padding of the signal
- **offset** value represents the zero point

Scope of **offset** and **scale**:

- **offset** is a signed integer typically being the same precision as quantized domain
- **scale** is an arbitrary real number

$$real = scale * (integer + offset)$$



$$a_{offset}\ v_m\ v_n^T$$

Quantized matrix operations

Standard GEMM operation:

- $C = alpha\ A\ B + beta\ C$

For neural networks, we need a GEMM-like operation:

- Extra requirement: add bias values to rows/columns of the output matrix
- $C_{real} = A_{real}B_{real} + b\ v_n^T$, or $C_{real} = A_{real}B_{real} + v_m b^T$, where v: vector of all ones; b: bias vector

First, let's look at the $\mathbf{C}_{real}=\mathbf{A}_{real}\mathbf{B}_{real}$ part:

- $c_{scale}\left(C_{quan} + \tilde{c_{offset}} v_m\ v_n^T\right) = a_{scale}\left(A_{quan} + a_{offset} v_m\ v_k^T\right) b_{scale}\left(B_{quan} + b_{offset} v_k\ v_n^T\right)$
- $C_{quan} = \frac{a_{scale} b_{scale}}{c_{scale}}\left(A_{quan} + a_{offset} v_m\ v_k^T\right)\left(B_{quan} + b_{offset} v_k\ v_n^T\right) + c_{offset} v_m\ v_n^T$ where $(c_{offset} = -\tilde{c_{offset}})$

Next, add the bias term:

- $C_{quan} = \frac{a_{scale} b_{scale}}{c_{scale}}\left(A_{quan} + a_{offset} v_m\ v_k^T\right)\left(B_{quan} + b_{offset} v_k\ v_n^T\right) + c_{offset} v_m\ v_n^T + (b\ v_n^T\ or\ v_m b^T)$

Quantized matrix-matrix multiplication

Finally, add the beta * C term and incorporate the c_{offset} to bias term

- $C_{quan} = \underbrace{\frac{a_{scale} b_{scale}}{c_{scale}}}_{alpha} (A_{quan} + a_{offset} v_m v_k^T)(B_{quan} + b_{offset} v_k v_n^T) + beta\ C_{quan} + (b\ v_n^T\ or\ v_m b^T)$

Argument	BLAS Type	CBLAS Type	Description
layout	N/A	CBLAS_LAYOUT	Row-major or column-major storage
transa	char*	CBLAS_TRANSPOSE	op(A)
transb	char*	CBLAS_TRANSPOSE	op(B)
biasc	**char***	**CBLAS_OFFSET**	**C bias is applied to rows or columns or a fixed offset for the entire matrix**
m	MKL_INT*	MKL_INT	First dimension of C matrix (number of rows for column major)
n	MKL_INT*	MKL_INT	Second dimension of C matrix (number of columns for column major)
k	MKL_INT*	MKL_INT	Common dimension of A and B matrices
alpha	double*	double	Alpha scalar multiplication
A	MKL_INT16*	MKL_INT16*	Pointer to input matrix A
lda	MKL_INT*	MKL_INT	Leading dimension for A matrix
oa	**MKL_INT16***	**MKL_INT16**	**Scalar offset value for A matrix**
B	MKL_INT16*	MKL_INT16*	Pointer to input matrix B
ldb	MKL_INT*	MKL_INT	Leading dimension for B
ob	**MKL_INT16***	**MKL_INT16**	**Scalar offset value for the B matrix**
beta	double*	double	Scalar scaling of the input/output C matrix
C	MKL_INT32*	MKL_INT32*	Pointer to the C matrix
ldc	MKL_INT*	MKL_INT	Leading dimension for the C matrix
bc	**MKL_INT32***	**MKL_INT32***	**Vector storing bias/offsets for C matrix.**

Function syntax and naming convention

$$C = alpha\left(op(A) + a_{offset} v_m \, v_k^T\right)\left(op(B) + b_{offset} v_k \, v_n^T\right) + beta\, C + \left(b \, v_n^T \; or \; v_m b^T\right)$$

GEMM_{S,U}{b1}{S,U}{b2}{S,U}{b3} (char* transa, char* transb, char* biasc, MKL_INT* m, MKL_INT* n, MKL_INT* k, double* alpha, MKL_[U]INT{b1}* A, MKL_INT{b1}* oa, MKL_INT* lda, MKL_[U]INT{b2}* B, MKL_INT{b2}* ob, MKL_INT* ldb, double* beta, MKL_[U]INT{b3}* C, MKL_INT* ldc, MKL_INT{b3}* bc)

- Offset/bias are same types as the corresponding matrix elements
- alpha and beta are double precision values
- Bias for C (bc) can be a scalar or a vector based on the value of offsetc
  - biasc = “F”, sizeof(bc) = 1
  - biasc = “R”, sizeof(bc) = num_cols(C)
  - biasc = “C”, sizeof(bc) = num_rows(C)

Implementation notes

Results from double-precision multiplications round to the nearest

- $C_{quan} = \underbrace{alpha\left(A_{quan} + a_{offset} v_m\, v_k^T\right)\left(B_{quan} + b_{offset} v_k\, v_n^T\right)}_{X} + \underbrace{beta\, C_{quan}}_{Y} + \underbrace{\left(b\, v_n^T \; or \; v_m b^T\right)}_{Z}$
- X, Y and Z are the partial results stored in double-precision
- C_{quan}= round_to_nearest(X + Y + Z);
- Open question: do we need alternative rounding modes (for e.g., stochastic rounding)?

Results may not be identical for X + Y + Z

- Enforcing the ordering ((X+Y) + Z) provides bitwise identical results

Additional implementation notes

Computation of the **X** term is susceptible to overflow/underflow

- $C_{quan} = \underbrace{alpha\left(A_{quan} + a_{offset} v_m\, v_k^T\right)\left(B_{quan} + b_{offset} v_k\, v_n^T\right)}_{\text{X}} + \underbrace{beta\, C_{quan}}_{\text{Y}} + \underbrace{(b\, v_n^T \; or \; v_m b^T)}_{\text{Z}}$
- Currently X term is expanded as:
- $X = A_{quan}B_{quan} + a_{offset} v_m\, v_k^T B_{quan} + b_{offset} A_{quan} v_k\, v_n^T + a_{offset} b_{offset} v_m\, v_k^T v_k\, v_n^T$
- $A_{quan}B_{quan}$ is like a regular matrix multiplication with input matrix precision
- This approach allows effectively utilizing input matrix precision for all offset values
- The order of integer addition is important to prevent overflows/underflows

What happens in the event of overflow/underflow?

- Overflow/underflow is highly undesirable for application developers
- Intel® MKL implementations saturate, which may lead to non-reproducible results

Integer GEMM implementations in Intel® MKL

Two variants are available in Intel® MKL 2018 Beta

- GEMM_S16S16S32: Input matrices A/B are 16-bit signed integer, input/output matrix C is 32bit signed integer
- GEMM_S16S16S16: All matrices A/B/C are 16-bit signed integer
- All scaling factors are double-precision (likely to be changed to single-precision for 16-bit output)
- Internal summation is with at least 32-bit signed integers
- Loosely follows XBLAS naming convention (missing the internal summation precision and abbreviations)
- Fixed-point matrix multiplication is a subset of the functionality (set offset values to 0)

Only saturation variants are implemented

More optimizations are coming for Intel® MKL 2018 Gold release

Future considerations

Rounding modes other than round-to-nearest

- Stochastic rounding may be required for the training (S Gupta et. al, ICML, 2015)

Alternative representations (for e.g., flex-point)

- Adjust scaling/offset values inside integer GEMM

Fuse activation functions with the integer GEMM functionality

- tanh, ReLU, etc…
- Partial results are already in double-precision

A flexible API that allows fusing operations for best performance

- Currently needed: *round_function*(*activation_function*(*double*(GEMM + bias)))
- In the future: $f_n(\ldots f_3(f_2(f_1(f_0(\text{GEMM}))))$

Scaling factors as fixed-points?

Are FORTRAN interfaces needed?

Summary

Intel® MKL 2018 Beta will provide two GEMM variants for quantized matrices

Bias is fused into the matrix-multiply for improved performance

Saturate instead of over-flowing or under-flowing

Reproducible results due to the integer computations - as long as there is no saturation

Additional variants with different precisions may be introduced based on the hardware support

Operation fusing is important for best performance

FORTRAN APIs are less relevant for the machine learning domain

Legal Disclaimer & Optimization Notice

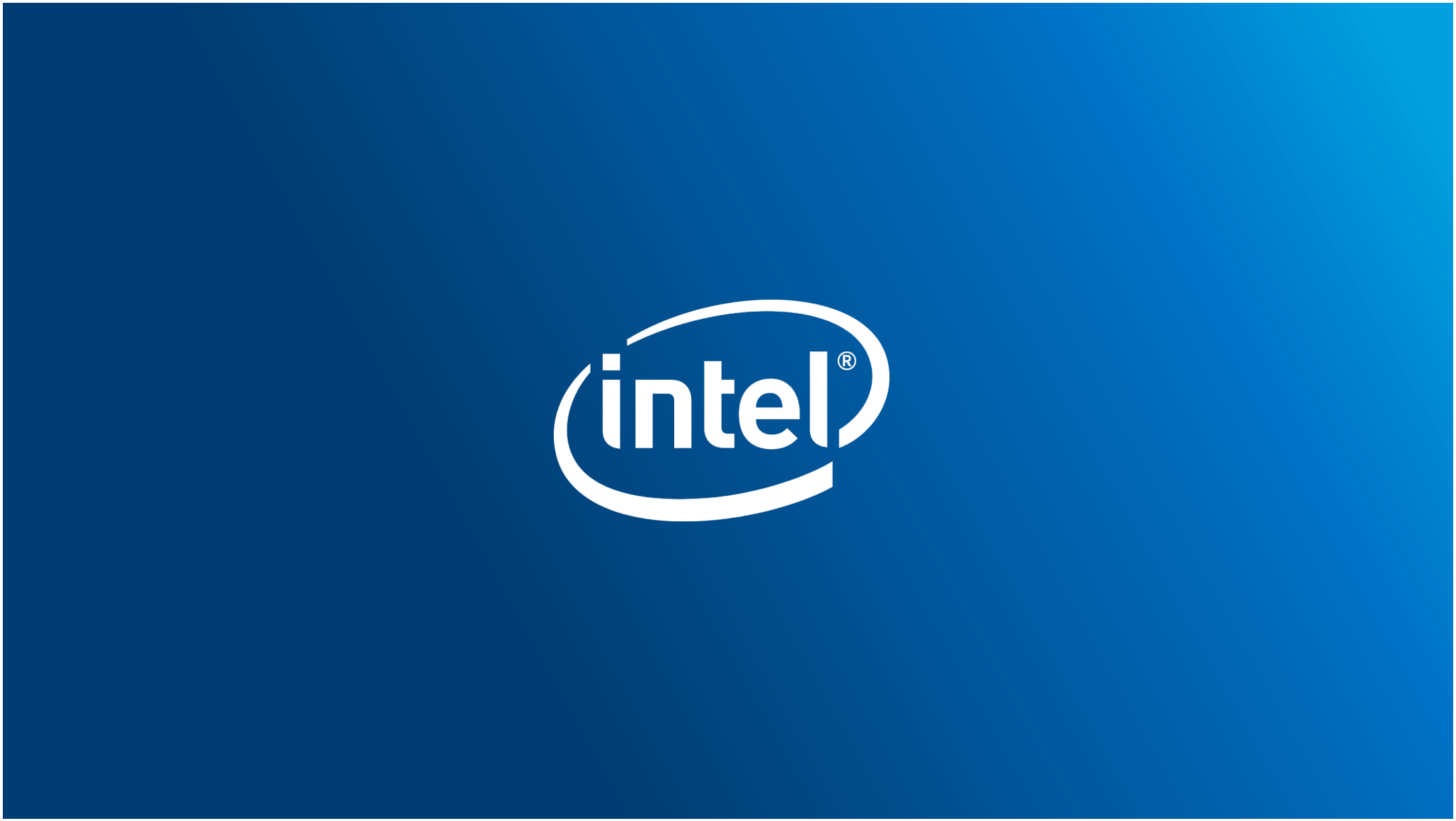
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. 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.

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


intel®