INTEL® MKL - BACIS LINEAR ALGEBRA SUBROUTINES (BLAS)

Gennady Fedorov - Technical Consulting Engineer

Intel Architecture, Graphics and Software (IAGS)

PRACE workshop, June 2020

Gennady.Fedorov@intel.com

Intel® Math Kernel Library

Linear Algebra
- **BLAS**
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Iterative sparse solvers
- PARDISO*
- Cluster Sparse Solver

FFTs
- Multidimensional
- FFTW interfaces
- Cluster FFT

Neural Networks
- Convolution
- Pooling
- Normalization
- ReLU
- Inner Product

Removed since MKL v.2020

Vector RNGs
- Congruential
- Wichmann-Hill
- Mersenne Twister
- Sobol
- Neiderreiter
- Non-deterministic

Summary Statistics
- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

Vector Math
- **Trigonometric**
- **Hyperbolic**
- **Exponential**
- **Log**
- **Power**
- **Root**

And More
- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver

Benchmarks
- Intel(R) Distribution for LINPACK* Benchmark
- High Performance Computing Linpack Benchmark
- High Performance Conjugate radient Benchmark

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS[1]*

[1] Available only in Intel® Parallel Studio Composer Edition.

BLAS Level 1 Routines

BLAS Level 1 includes routines and functions, which perform vector-vector operations. Table "BLAS Level 1 Routine Groups and Their Data Types" lists the BLAS Level 1 routine and function groups and the data types associated with them.

BLAS Level 1 Routine and Function Groups and Their Data Types

Routine or Function Group	Data Types	Description
?asum	s, d, sc, dz	Sum of vector magnitudes (functions)
?axpy	s, d, c, z	Scalar-vector product (routines)
?copy	s, d, c, z	Copy vector (routines)
?dot	s, d	Dot product (functions)
?sdot	sd, d	Dot product with double precision (functions)
?dotc	c, z	Dot product conjugated (functions)
?dotu	c, z	Dot product unconjugated (functions)
?nrm2	s, d, sc, dz	Vector 2-norm (Euclidean norm) (functions)
?rot	s, d, cs, zd	Plane rotation of points (routines)
?rotg	s, d, c, z	Generate Givens rotation of points (routines)
?rotm	s, d	Modified Givens plane rotation of points (routines)
?rotmg	s, d	Generate modified Givens plane rotation of points (routines)
?scal	s, d, c, z, cs, zd	Vector-scalar product (routines)
?swap	s, d, c, z	Vector-vector swap (routines)
i?amax	s, d, c, z	Index of the maximum absolute value element of a vector (functions)
i?amin	s, d, c, z	Index of the minimum absolute value element of a vector (functions)
?cabs1	s, d	Auxiliary functions, compute the absolute value of a complex number of single or double precision

BLAS Level 2 Routines

BLAS Level 2 Routine Groups and Their Data Types

Routine Groups	Data Types	Description
?gbmv	s, d, c, z	Matrix-vector product using a general band matrix
?gemv	s, d, c, z	Matrix-vector product using a general matrix
?ger	s, d	Rank-1 update of a general matrix
?gerc	c, z	Rank-1 update of a conjugated general matrix
?geru	c, z	Rank-1 update of a general matrix, unconjugated
?hbmv	c, z	Matrix-vector product using a Hermitian band matrix
?hemv	c, z	Matrix-vector product using a Hermitian matrix
?her	c, z	Rank-1 update of a Hermitian matrix
?her2	c, z	Rank-2 update of a Hermitian matrix
?hpmv	c, z	Matrix-vector product using a Hermitian packed matrix
?hpr	c, z	Rank-1 update of a Hermitian packed matrix
?hpr2	c, z	Rank-2 update of a Hermitian packed matrix
?sbmv	s, d	Matrix-vector product using symmetric band matrix
?spmv	s, d	Matrix-vector product using a symmetric packed matrix
?spr	s, d	Rank-1 update of a symmetric packed matrix
?spr2	s, d	Rank-2 update of a symmetric packed matrix
?symv	s, d	Matrix-vector product using a symmetric matrix
?syr	s, d	Rank-1 update of a symmetric matrix
?syr2	s, d	Rank-2 update of a symmetric matrix
?tbmv	s, d, c, z	Matrix-vector product using a triangular band matrix
?tbsv	s, d, c, z	Solution of a linear system of equations with a triangular band matrix
?tpmv	s, d, c, z	Matrix-vector product using a triangular packed matrix

BLAS Level 3 Routines

BLAS Level 3 routines perform matrix-matrix operations. Table "BLAS Level 3 Routine Groups and Their Data Types" lists the BLAS Level 3 routine groups and the data types associated with them.

BLAS Level 3 Routine Groups and Their Data Types

Routine Group	Data Types	Description
?gemm	s, d, c, z	Computes a matrix-matrix product with general matrices.
?hemm	c, z	Computes a matrix-matrix product where one input matrix is Hermitian.
?herk	c, z	Performs a Hermitian rank-k update.
?her2k	c, z	Performs a Hermitian rank-2k update.
?symm	s, d, c, z	Computes a matrix-matrix product where one input matrix is symmetric.
?syrk	s, d, c, z	Performs a symmetric rank-k update.
?syr2k	s, d, c, z	Performs a symmetric rank-2k update.
?trmm	s, d, c, z	Computes a matrix-matrix product where one input matrix is triangular.
?trsm	s, d, c, z	Solves a triangular matrix equation.

BLAS-like Extensions

Routine	Data Types	Description
?axpby	s, d, c, z	Scales two vectors, adds them to one another and stores result in the vector (routines).
?gem2vu	s, d	Two matrix-vector products using a general matrix, real data.
?gem2vc	c, z	Two matrix-vector products using a general matrix, complex data.
?gemmt	s, d, c, z	Computes a matrix-matrix product with general matrices but updates only the upper or lower triangular part of the result matrix.
?gemm3m	c, z	Computes a scalar-matrix-matrix product using matrix multiplications and adds the result to a scalar-matrix product.
?gemm_batch	s, d, c, z	Computes scalar-matrix-matrix products and adds the results to scalar matrix products for groups of general matrices.
?gemm3m_batch	c, z	Computes a scalar-matrix-matrix product using matrix multiplications and adds the result to a scalar-matrix product.
mkl_?imatcopy	s, d, c, z	Performs scaling and in-place transposition/copying of matrices.
mkl_?omatcopy	s, d, c, z	Performs scaling and out-of-place transposition/copying of matrices.
mkl_?omatcopy2	s, d, c, z	Performs two-strided scaling and out-of-place transposition/copying of matrices.
mkl_?omatadd	s, d, c, z	Performs scaling and sum of two matrices including their out-of-place transposition/copying.
?gemm_alloc	s, d	Allocates storage for a packed matrix.
?gemm_pack	s, d	Performs scaling and packing of the matrix into the previously allocated buffer.
?gemm_compute	s, d	Computes a matrix-matrix product with general matrices where one or both input matrices are stored in a packed data structure and adds the result to a scalar-matrix product.
?gemm_free	s, d	Frees the storage previously allocated for the packed matrix.
gemm_*	integer	Computes a matrix-matrix product with general integer matrices.
mkl_jit_create_?gemm	s, d	Creates a handle on a jitter and generates a GEMM kernel that computes a scalar-matrix-matrix product and adds the result to a scalar-matrix product, with general matrices.
mkl_jit_get_?gemm_ptr	s, d	Returns the GEMM kernel previously generated.

Intel MKL BLAS DEMO, GEMM – Agenda

- **General**
- **Direct Call**
- **JIT**
- **Packed API**
- Batch API
- COMPACT API

Requirements

- Intel® Parallel Studio XE 2020 Composer Edition with Intel® C++ Compiler
- Linux* OS supported by Intel® C++ Compiler
- Recommended to have at least 3nd **generation Intel® Core™ processor** (with Intel® AVX2)

Setting the PATH, LIB, and INCLUDE environment variables

Compiler:

source /opt/intel/compilers_and_libraries_2020.1.127/linux/bin/compilervars.sh intel64

module load intel64/19.1up01 // ssh meggie

MKL:

source <mklroot>/bin/mklvars.sh intel64

GEMM – API

C = αA * B + βC Computes a matrix-matrix product with general matrices.

```
void cblas_[s,d,c,z]gemm (
   const CBLAS_LAYOUT layout,
   const CBLAS_TRANSPOSE transa,
   constCBLAS_TRANSPOSE transb,
   const MKL_INT m,
   const MKL_INT n,
   const MKL_INT k,
   const dtype alpha,
   const dtype *a,
   const MKL_INT lda,
   const dtype *b,
   const MKL_INT ldb,
   const dtype beta,
   dtype *c,
   const MKL_INT ldc);
```

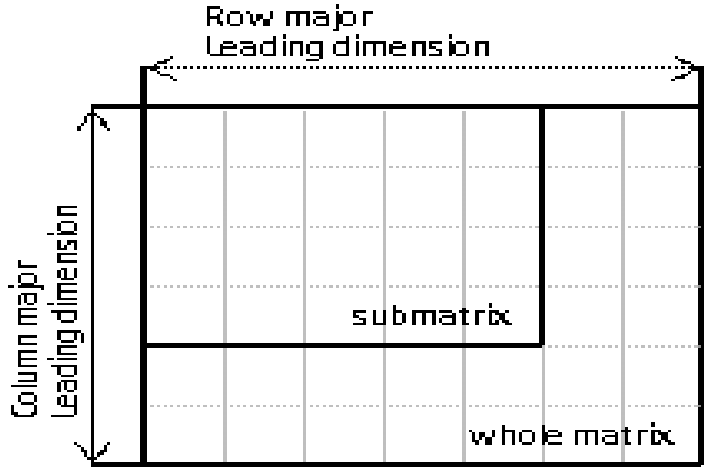
layout- {CblasRowMajor, CblasColMajor}

trans - {CblasNoTrans, CblasTrans, CblasConjTrans}

m = Number of rows in a = Number of rows in c

k = Number of columns in a = Number of rows in b

n = Number of columns in b = Number of columns in c



DEMO, GEMM - General

directory: /workshop/mkl/BLAS**/#1General**

- Review test: gemm.cpp
  - mkl_malloc, mkl_free, dsecnd, cblas_dgemm, mkl_get_version()
- Compiling: icc –mkl gemm.cpp
  - refer to MKL Linker Adviser: https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor
- Export MKL_NUM_THREADS=1
- ./a.out 4000

```
.....
Processor optimization:  Intel(R) Advanced Vector Extensions 512 (Intel(R) AVX-512) enabled processors
        Average time: 1.265071e+00 secs
        GFlop/sec   :   101.18009
```

DEMO, GEMM – General, Verbose

- MKL Verbose mode:
  - set/export MKL_VERBOSE=1 // 0 by default
  - int mkl_verbose (true/false); // False by default
- ./a.out 4000

MKL_VERBOSE Intel(R) MKL 2020.0 Update 1 Product build 20200208 for Intel(R) 64 architecture Intel(R) Advanced Vector Extensions 512 (Intel(R) AVX-512) enabled processors, Lnx 2.40GHz lp64 intel_thread

MKL_VERBOSE
DGEMM(N,N,**4000**,4000,4000,0x7ffe59856990,0x2ae5630d9080,4000,0x2ae56aaec080,4000,0x7ffe59856998,0x2ae5724ff080,4000) **1.32**s **CNR:OFF Dyn:1** FastMM:1 TID:0 **NThr:1**

*Intel® Xeon® Gold 6148 Processor

DEMO, GEMM – General, Scaling

- export MKL_VERBOSE=0
- KMP_AFFINITY to avoid thread migration

  export KMP_AFFINITY=compact,1,0,granularity=fine
- export MKL_NUM_THREADS= #THR, where #THR == 1, 2, 4, 8, 16, 32

  ./a.out 4000

#threads	1	2	4	8	16	32
gflops	100	194	366	677	1072	2018

*Intel® Xeon® Gold 6148 Processor

DEMO, GEMM – General, Conditional Numerical Reproducibility

export MKL_CBWR=**VALUE**

MKL_CBWR_AUTO	2	CNR mode uses the standard ISA-based dispatching model while ensuring fixed cache sizes, deterministic reductions, and static scheduling
		CNR mode uses the branch for the following ISA:
MKL_CBWR_COMPATIBLE	3	Intel® Streaming SIMD Extensions 2 (Intel® SSE2) without rcpps/rsqrtps instructions
MKL_CBWR_SSE2	4	Intel SSE2
MKL_CBWR_SSE3	5	DEPRECATED. Intel® Streaming SIMD Extensions 3 (Intel® SSE3). This setting is kept for backward compatibility and is equivalent to MKL_CBWR_SSE2.
MKL_CBWR_SSSE3	6	Supplemental Streaming SIMD Extensions 3 (SSSE3)
MKL_CBWR_SSE4_1	7	Intel® Streaming SIMD Extensions 4-1 (SSE4-1)
MKL_CBWR_SSE4_2	8	Intel® Streaming SIMD Extensions 4-2 (SSE4-2)
MKL_CBWR_AVX	9	Intel® Advanced Vector Extensions (Intel® AVX)
MKL_CBWR_AVX2	10	Intel® Advanced Vector Extensions 2 (Intel® AVX2)
MKL_CBWR_AVX512_MIC	11	Intel® Advanced Vector Extensions 512 (Intel® AVX-512) on Intel® Xeon Phi™ processors
MKL_CBWR_AVX512	12	Intel AVX-512 on Intel® Xeon® processors

DEMO, GEMM – General, Conditional Numerical Reproducibility

review and launch run_cnr.sh

./a.out 4000 (./run_cnr.sh)

- export MKL_CBWR=COMPATIBLE GFlops = ~11.4
- export MKL_CBWR=SSE4_2 GFlops = ~14
- export MKL_CBWR=AVX GFlops = ~28
- export MKL_CBWR=AVX2 GFlops = ~53
- export MKL_CBWR=AVX512 GFlops = ~ 100

*Intel® Xeon® Gold 6148 Processor

DEMO, GEMM – Direct Call

Small Sizes (M, N, K < 20)

Challenges: High function call overheads, low vectorization, low parallelization

Solutions: Batch API, Compact API and MKL_DIRECT_CALL

MKL_DIRECT_CALL

- Improves performance for small sizes (M, N, K < 20)
- Skips error checking and function call overheads
- Enabled for several functions
  - BLAS: gemm, gemm3m, syrk, trsm, axpy, dot
  - LAPACK: potrf, getrf, getrs, getri, geqrf

DEMO, GEMM – MKL_DIRECT_CALL

- Define the preprocessor macro MKL_DIRECT_CALL
  - If threading is not required, use MKL_DIRECT_CALL_SEQ
- Improves the performance of small sizes (M, N, K < 20)
  - Instead of calling a library function, a C implementation may be used
  - Starting from Intel MKL 2018.1, compiler intrinsics kernels for DGEMM Intel AVX2+
- Intel MKL can avoid some of the overheads
  - No error checking
  - No MKL_VERBOSE support
  - No CNR (Conditional Numerical Reproducibility) support
- Minimal modification is required, just add the preprocessor macro and the header file:

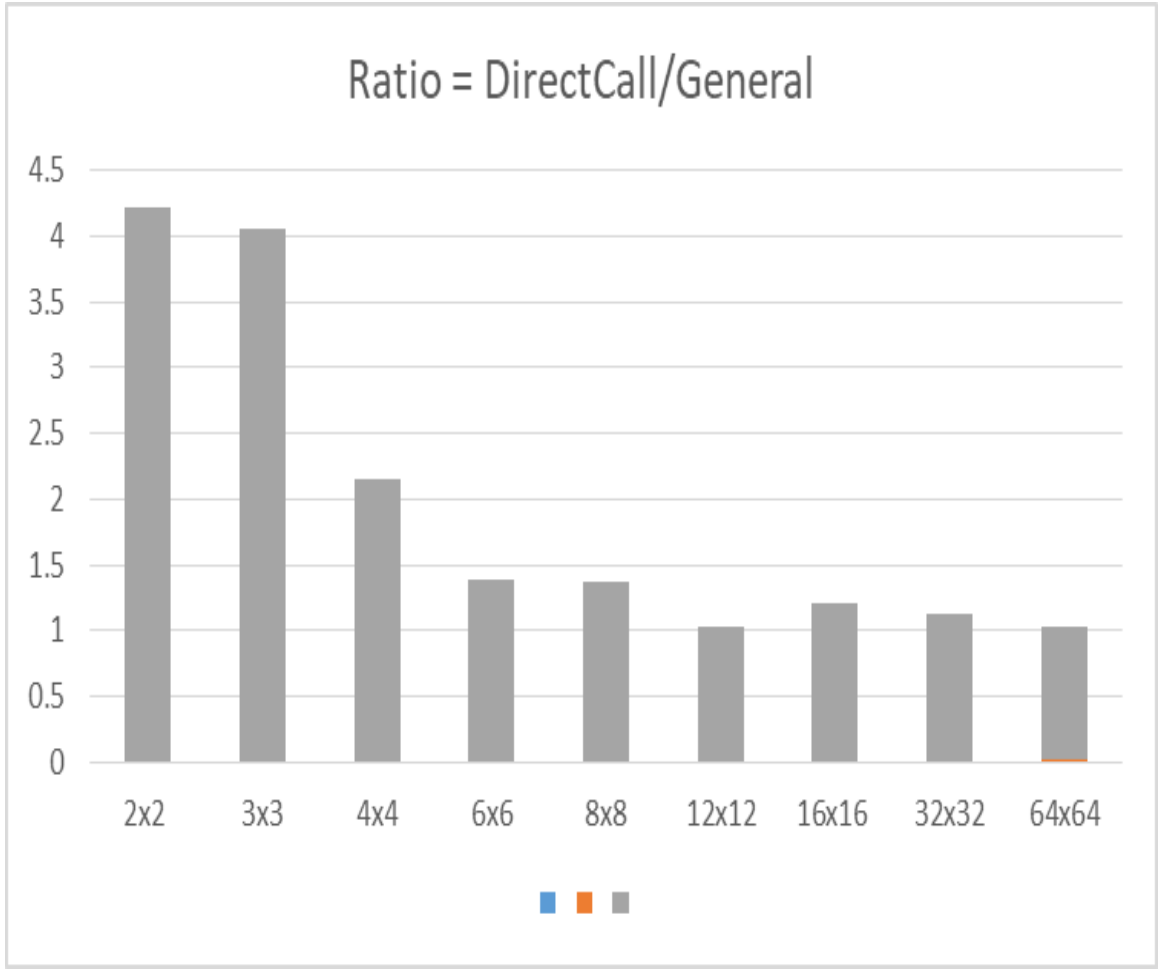
```
// compile with: icc -DMKL_DIRECT_CALL …
#include <mkl.h>
void main(void) {
    dgemm(…);
}
```

```
! compile with: ifort -DMKL_DIRECT_CALL -fpp …
#     include “mkl_direct_call.fi”
      program DGEMM_MAIN
      DGEMM(…)
```

DEMO, GEMM – Direct Call

- Directory: ~ /mkl/2BLAS/#2DirectCall
- Review test: dgemm_small.cpp
- Review makefile
  - CCFLAGS = –DMKL_DIRECT_CALL –std=c99
  - make : Compiling with and without CCFLAGS
- ./rundirect.sh

*Intel® Xeon® Gold 6148 Processor



DEMO, GEMM – JIT

- Introduced in MKL v.2019
- Language supported: C (CBLAS interface only) and Fortran (same function name as C API)
- All architectures supported, by default pointer to standard GEMM is returned
- JIT only for AVX2, AVX512 and M, N, K ≤ 16
- MKL_DIRECT_CALL_JIT
- Limitations: Verbose and CNR features are not supported

DEMO, GEMM – JIT API

- **Types**

```
typedef enum {MKL_JIT_ERROR, MKL_JIT_SUCCESS, MKL_NO_JIT}
mkl_jit_status_t;

typedef (*{s,d}gemm_jit_kernel_t)(void*, FP_TYPE*, FP_TYPE*, FP_TYPE*)
```

- **Functions:**

```
mkl_jit_status_t mkl_jit_create_{s,d}gemm(void** jitter, MKL_LAYOUT
layout, MKL_TRANSPOSE transa, MKL_TRANSPOSE transb, MKL_INT m, MKL_INT n, MKL_INT k,
FP_TYPE alpha, MKL_INT lda, MKL_INT ldb, FP_TYPE beta, MKL_INT ldc)

{s,d}gemm_jit_kernel_t mkl_jit_get_{s,d}gemm_ptr(void* jitter)

mkl_jit_status_t mkl_jit_destroy(void* jitter)
```

DEMO, GEMM – JIT EXAMPLE

```
int main() {
    MKL_INT m = 10, n = 5, k = 12, lda = 32, ldb = 32, ldc = 32;
    MKL_TRANSPOSE transa = MKL_NOTRANS, transb = MKL_TRANS;
    MKL_LAYOUT layout = MKL_COL_MAJOR;
    float alpha = 2.0, beta = 1.0;
    float *a, *b, *c;
    void* jitter_s_10_5_12;

    // allocate and initialize matrices
    mkl_jit_status_t status = mkl_jit_create_sgemm(&jitter_s_10_5_12, layout, transa, transb, m, n, k,
                                                   alpha, lda, ldb, beta, ldc);
    if (MKL_JIT_ERROR == status) {
        printf("Creation jitter failed\n");
        return 1;
    }
    sgemm_jit_kernel_t sgemm_10_5_12 = mkl_jit_get_sgemm_ptr(jitter_s_10_5_12);

    sgemm_10_5_12(jitter_s_10_5_12, a, b, c);

    mkl_jit_destroy(jitter_s_10_5_12);
    // free matrices
    return 0;
}
```

perform C = alpha * A x B + beta*C

DEMO, GEMM-JIT (JIT KERNEL vs GENERAL)

Directory: /workshop/mkl/2BLAS**/#3JIT**

Building:

make

#Run and record the Executions times:

./run.sh

Note: be aware that CPU supports AVX2 and or AVX-512 ISA:

cat /proc/cpuinfo | grep avx2 (avx512)

```
[gfedorov@sk110 #3_JIT]$ ./run.sh
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 2x2 <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [2 x 2], SGEMM Execution Time == 1.881272e-07
        [2 x 2], JIT_SGEMM Execution Time == 3.911555e-08
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 3x3 <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [3 x 3], SGEMM Execution Time == 1.909211e-07
        [3 x 3], JIT_SGEMM Execution Time == 4.097819e-08
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 4x4 <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [4 x 4], SGEMM Execution Time == 1.927838e-07
        [4 x 4], JIT_SGEMM Execution Time == 4.004687e-08
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 6x6 <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [6 x 6], SGEMM Execution Time == 1.974404e-07
        [6 x 6], JIT_SGEMM Execution Time == 5.029142e-08
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 8x8 <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [8 x 8], SGEMM Execution Time == 2.076849e-07
        [8 x 8], JIT_SGEMM Execution Time == 4.656613e-08
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 12x12 <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [12 x 12], SGEMM Execution Time == 2.328306e-07
        [12 x 12], JIT_SGEMM Execution Time == 6.798655e-08
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 16x16  <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [16 x 16], SGEMM Execution Time == 2.468005e-07
        [16 x 16], JIT_SGEMM Execution Time == 8.009374e-08
>>>>>>>>>>>>>>>>>>>>>>>>>>>> 20x20  <<<<<<<<<<<<<<<<<<<<<<<<<<<<
        [20 x 20], SGEMM Execution Time == 3.483146e-07
        [20 x 20], JIT_SGEMM Execution Time == 1.639128e-07
[gfedorov@sk110 #3_JIT]$
```

DEMO, GEMM – DIRECT_CALL_JIT

MKL_DIRECT_CALL_JIT :

- make jitdirect

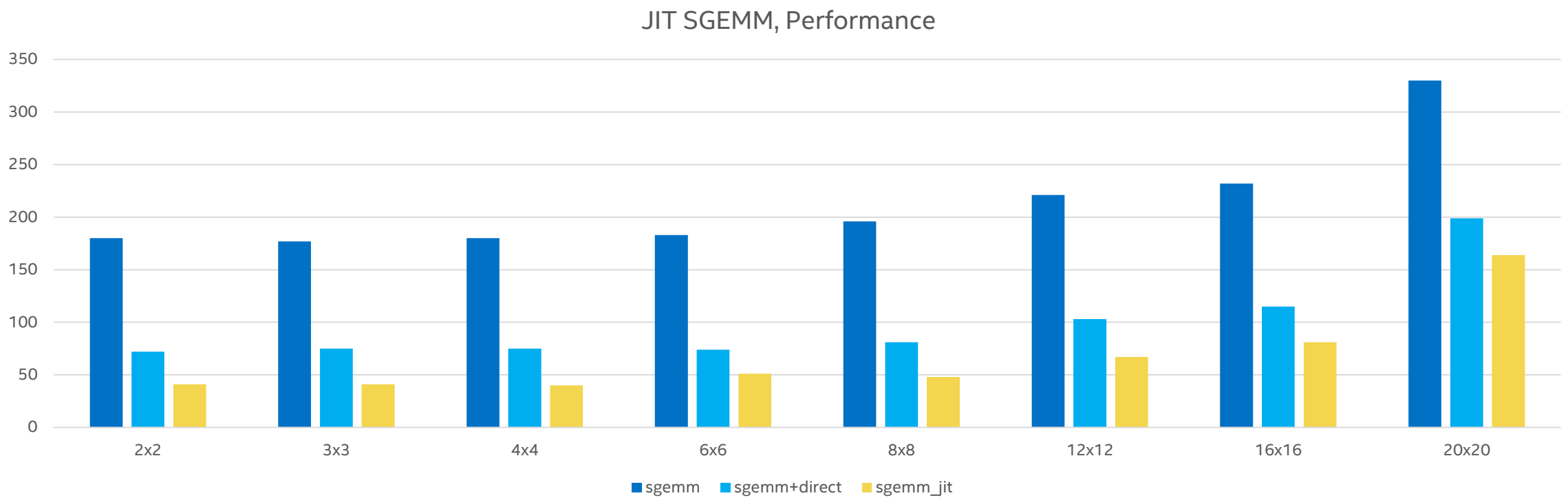
```
icc -DMKL_DIRECT_CALL_JIT -std=c99 -I${MKL_INCL} small_gemm.c -o jit_direct.out  -Wl,--
start-group  ${2019}/mkl/lib/intel64/libmkl_intel_lp64.a
${2019}/mkl/lib/intel64/libmkl_intel_thread.a ${2019}/mkl/lib/intel64/libmkl_core.a  -Wl,--
end-group -liomp5 -lpthread -lm -ldl
```

- run: ./jitdirect.out <size>, size = {2, 3, 4, 6, 8, 12, 16, 20}
- Compare the performance results with previous calls

Conclusions? Do you see something like this?

```
[4 x 4], SGEMM       Execution Time  ==  1.473993e-06
[4 x 4], SGEMM JIT   Execution Time  ==  1.335982e-06
[4 x 4], JIT_SGEMM Execution Time  ==  1.204957e-06
```

DEMO, GEMM-JIT, Performance



Configuration Info – SW Versions: Intel® Math Kernel Library (Intel® MKL) 2020. Hardware: Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz,192 GB RAM (12x16GB DDR4-2666). 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. Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

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® MKL, GEMM – Packed API

- Amortize copy (pack) operation over multiple GEMM calls with the same input matrix
- Copy (pack) the data once and reuse it in many GEMM calls
- Improves the performance for medium or skewed sizes (M or N < 500) with input matrix reuse

$$C^1 = alpha \cdot op(A^1) \cdot op(B^1) + beta \cdot C^1$$

$$C^2 = alpha \cdot op(A^1) \cdot op(B^2) + beta \cdot C^2$$

$$C^3 = alpha \cdot op(A^1) \cdot op(B^3) + beta \cdot C^3$$

Input matrix A^1 is shared between three GEMM calls

Intel® MKL, GEMM – Packed API (cont'd)

Code modifications are required to transform GEMM calls into GEMM_PACK + GEMM_COMPUTE
Three SGEMM calls with shared A matrix is computed with the packed APIs below:

```
#include <mkl.h>

float *Ap;
Ap = sgemm_alloc("A", &m, &n, &k);

// transform A into packed format
sgemm_pack("A", "T", &m, &n, &k, &alpha, A, &lda, Ap);

// SGEMM computations are performed using the packed A matrix: Ap
sgemm_compute("P", "N", &m, &n, &k, Ap, &lda, B1, &ldb1, &beta, C1, &ldc1);
sgemm_compute("P", "N", &m, &n, &k, Ap, &lda, B2, &ldb2, &beta, C2, &ldc2);
sgemm_compute("P", "N", &m, &n, &k, Ap, &lda, B3, &ldb3, &beta, C3, &ldc3);

// release the memory for Ap
sgemm_free(Ap);
```

DEMO, GEMM – Packed API (cont'd)

directory: /workshop/mkl/BLAS/#4Packed

- Review test: gemm_packed_benchmark.cpp
- Building: icc **-mkl** gemm_packed_benchmark.cpp
- run : **./a.out 100**

Expected Output:

```
~/workshop/mkl/BLAS/#4Packed$ ./a.out 100
        the problem solve: C(m,n) = A(m,k)*B(k,n) ....
.... Testing problems: M x K x N == 100, 10240, 10240 ......
  ..  Classical API, Execution == 1.431301, sec
  ... Packed API, Execution == 0.992116, sec
  .... Packed API faster in   == 1.442676 times, sgemm = 163.749570 Gflops, sgemm_compute = 236.237528 GFlops
```

Intel® MKL, GEMM – Batch API (cont'd)

- Improves performance for small-medium sizes (M, N, K < 500)
- Groups several independent function calls together
- Enabled for gemm, gemm3m and trsm BLAS functions

```
#include <mkl.h>
int group_count = 2;
// Create arrays size of group_count to store GEMM
arguments
CBLAS_TRANSPOSE transA[] = {CblasNoTrans, CblasNoTrans};
CBLAS_TRANSPOSE transB[] = {CblasTrans, CblasNoTrans};
MKL_INT   m[] = {4, 3};
MKL_INT   k[] = {4, 6};
MKL_INT   n[] = {8, 3};
MKL_INT   lda[] = {4, 6};
MKL_INT   ldb[] = {4, 6};
MKL_INT   ldc[] = {8, 3};
double    alpha[] = {1.0, 1.0};
double    beta[] = {0.0, 2.0};
MKL_INT size_per_grp[] = {20, 30};

// Call cblas_dgemm_batch to perform GEMM operations
cblas_dgemm_batch(CblasRowMajor, transA, transB, m, n, k,
                  alpha, a_array, lda, b_array, ldb,
                  beta, c_array, ldc, group_count,
size_per_group);
```

Intel® MKL, GEMM – Compact API (cont'd)

- Non-standard BLAS API that requires some code modification
- Improves performance significantly for a **large group** of **same-size** small matrices (M, N, K < 20)
- Intel MKL utility functions to transform matrices between column/row major and compact layout:

```
#include <mkl.h>

// query the optimal format for the architecture
MKL_COMPACT_PACK compact_format = mkl_get_format_compact();

// allocate memory for compact layout
a_size = mkl_dget_size_compact(lda, k, compact_format, num_matrix);
b_size = mkl_dget_size_compact(ldb, n, compact_format, num_matrix);
c_size = mkl_dget_size_compact(ldc, n, compact_format, num_matrix);

// transform the data into the compact format
mkl_dgepack_compact(layout, m, k, a_array, lda, a_c, lda, compact_format, num_matrix);
mkl_dgepack_compact(layout, k, n, b_array, ldb, b_c, ldb, compact_format, num_matrix);
mkl_dgepack_compact(layout, m, n, c_array, ldc, c_c, ldc, compact_format, num_matrix);

// multiple dgemm operations on compact data layout
mkl_dgemm_compact(layout, transa, transb, m, n, k, alpha, a_c, lda, b_c, ldb, beta, c_c, ldc, compact_format, num_matrix);

// transform from compact format to standard BLAS format
mkl_dgeunpack_compact(layout, m, n, c_array, ldc, c_c, ldc, compact_format, num_matrix);
```

Intel MKL Resources

Intel® MKL website:

– https://software.intel.com/en-us/intel-mkl

Intel MKL forum:

– https://software.intel.com/en-us/forums/intel-math-kernel-library

Intel® MKL benchmarks:

– https://software.intel.com/en-us/intel-mkl/benchmarks#

Intel® MKL link line advisor:

– http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/

Legal Disclaimer & Optimization Notice

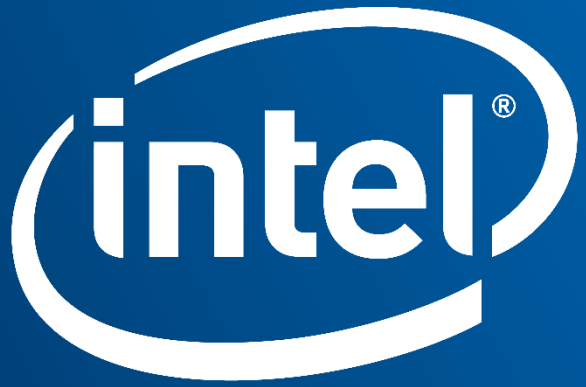
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING 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



Software