Enhancements: Just-in-Time Compilation, Packed GEMM APIs, and Integer GEMMs

Fastest and most used math library for Intel®-based systems[1]

Speaker: Alexander Kalinkin

Contributing authors: Peter Caday, Kazushige Goto, Louise Huot, Sarah Knepper, Mesut Meterelliyoz, Arthur Araujo Mitrano, Shane Story

[1] Data from Evans Data Software Developer surveys, 2011-2018

Outline

- Intel MKL Solutions for Small, Medium, and Skewed Sizes
- Just-in-time (JIT) run-time compilation for matrix multiplication
- Packed GEMM APIs
- Integer GEMMs

Classification of Matrix Sizes and Performance Challenges

Small Sizes

- M, N, K < ~20
- Challenges: High function call overheads, low vectorization, low parallelization

Medium Sizes

- ~20 < M, N, K < ~500
- Challenges: Low parallelization, high copy overheads

Skewed Sizes

- M < ~500 and large N
- N < ~500 and large M
- Challenge: High copy overheads

Large Sizes

- M, N, K > ~5000
- Performance close to machine’s theoretical peak

Intel MKL Solutions for Small, Medium, and Skewed Sizes

Direct call (since Intel MKL 2017)

- Reduces function call overheads, skips error checking, some compile-time optimizations
- Use preprocessor macros
- Enabled for: GEMM, GEMM3M, SYRK, TRSM, AXPY, DOT, POTRF, GETRF, GETRS, GETRI, GEQRF

Just-in-time (JIT) run-time compilation (since Intel MKL 2019)

- Decreases library, loop, and corner case handling overheads
- Use new preprocessor macros or new JIT APIs
- Enabled for: DGEMM, SGEMM; in an upcoming release: ZGEMM, CGEMM

Compact APIs (since Intel MKL 2018)

- Enables vectorization over very small matrix dimensions by reformatting the data in a compact layout
- Enabled for: GEMM, TRSM, GETRINP, GETRFNP, POTRF, GEQRF

Batch APIs (since Intel MKL 11.3)

- Groups several independent function calls together to improve core usage
- Enabled for: GEMM, GEMM3M, TRSM

Packed APIs (since Intel MKL 2017)

- Allows amortizing copy overheads over several GEMM calls with same input matrix
- Enabled for: DGEMM, SGEMM, GEMM_S8U8S32, GEMM_S16S16S32

JIT Capability in Intel MKL

With preprocessor macro MKL_DIRECT_CALL_JIT or MKL_DIRECT_CALL_SEQ_JIT

- No changes to user code
- Intel MKL may JIT a specific kernel
- Kernels are stored in an internal hash table to amortize cost of generation

With new JIT APIs

- User responsible for managing kernels
- Further eliminates overheads for even better performance

Utilizes Xbyak JIT compiler underneath (https://github.com/herumi/xbyak)

Packed APIs Overview

- GEMM may copy (pack) the input matrices into internal buffers for efficient computation
- Copy operation is costly for medium or skewed sizes (M or N < ~500)
- Amortize the copy operation over multiple GEMM calls with the same input matrix
- Copy the data once and reuse it in many GEMM calls
- Improves the performance when there is input matrix reuse

$$\left.\begin{aligned} \mathbf{C}^1 &= \mathit{alpha} \cdot \mathrm{op}(\mathbf{A}^1) \cdot \mathrm{op}(\mathbf{B}^1) + \mathit{beta} \cdot \mathbf{C}^1 \\ \mathbf{C}^2 &= \mathit{alpha} \cdot \mathrm{op}(\mathbf{A}^1) \cdot \mathrm{op}(\mathbf{B}^2) + \mathit{beta} \cdot \mathbf{C}^2 \\ \mathbf{C}^3 &= \mathit{alpha} \cdot \mathrm{op}(\mathbf{A}^1) \cdot \mathrm{op}(\mathbf{B}^3) + \mathit{beta} \cdot \mathbf{C}^3 \end{aligned}\right\}$$

Input matrix $\mathbf{A}^1$ is shared between three GEMM calls

Intel MKL Reduced Precision Support

Integer matrix-matrix multiplication routines that work with quantized matrices (since Intel MKL 2018)

- GEMM_S8U8S32, GEMM_S16S16S32
  - S - signed, U - unsigned; # bits
- User quantizes each matrix
- C := alpha*(op(A) + A_offset)*(op(B) + B_offset) + beta*C + C_offset
- {A,B}_offset are matrices with every element equal to a given value
- C_offset is a matrix with every element, row, or column equal to given value(s)

Packed APIs available since Intel MKL 2019 Update 1

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

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

Backup

JIT APIs Workflow

Create a handle and generate GEMM kernel:

```
mkl_jit_status_t status = mkl_jit_create_sgemm(&jit_handle,
    layout, transa, transb, m, n, k, alpha, lda, ldb, beta, ldc);
```

Get kernel associated with handle:

```
sgemm_jit_kernel_t kernel = mkl_jit_get_sgemm_ptr(jit_handle);
```

Repeatedly execute the GEMM kernel:

```
kernel(jit_handle, a, b, c);
```

Destroy the created handle/GEMM kernel:

```
mkl_jit_destroy(jit_handle);
```

Packed APIs Workflow

Get size needed for buffer

```
size = cblas_sgemm_pack_get_size(identifier, m, n, k);
```

Allocate buffer

```
Ap = mkl_malloc(size, 64);
```

Perform packing

```
cblas_sgemm_pack(layout, identifier, trans, m, n, k, alpha, A, lda, Ap);
```

Repeatedly compute GEMM with the packed matrix

```
cblas_sgemm_compute(layout, transa, transb, m, n, k, Ap, lda,
                    B1, ldb1, beta, C1, ldc1);
```

Free allocated buffer

```
mkl_free(Ap);
```

Example Usage of New JIT APIs (slide 1 of 2)

```
MKL_LAYOUT layout;
MKL_TRANSPOSE transA, transB;
MKL_INT m, n, k, lda, ldb, ldc;
float alpha, beta, *a, *b, *c;
void* jitter;
// Initialize user data (not shown)

// Create jitter handle and generate GEMM kernel
mkl_jit_status_t status = mkl_jit_create_sgemm(
    &jitter, layout, transA, transB, m, n, k, alpha, lda, ldb, beta, ldc);


// Check that creation was successful
if (MKL_JIT_ERROR == status) {
   printf("Error: cannot create jitter\n");
   return 1;
}
```

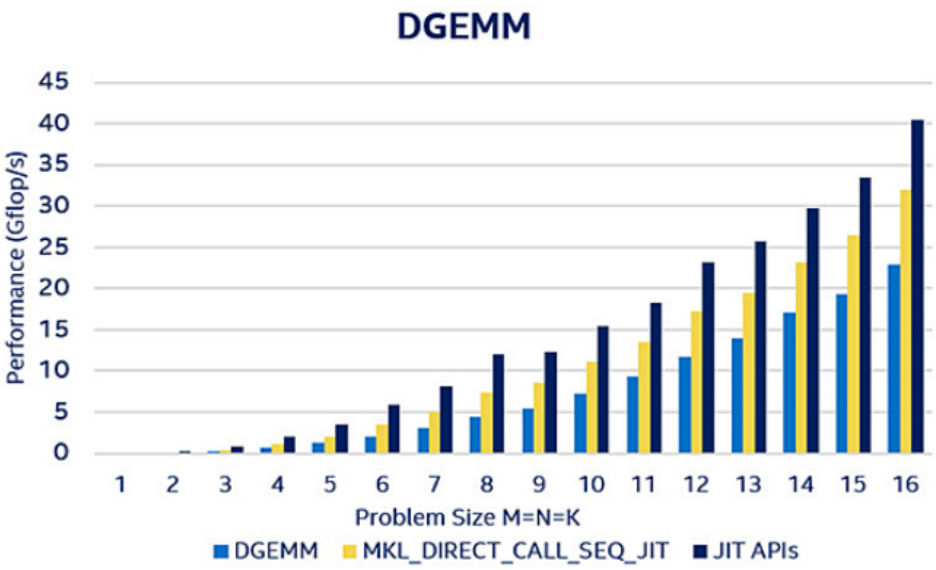
Example Usage of New JIT APIs (slide 2 of 2)

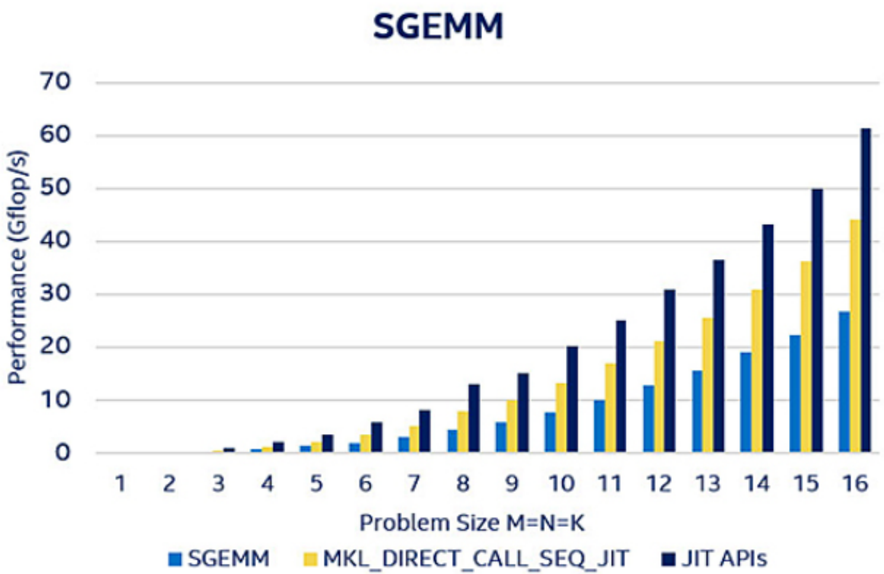
```
// Get kernel associated with jitter handle
sgemm_jit_kernel_t kernel = mkl_jit_get_sgemm_ptr(jitter);

// Repeatedly execute GEMM kernel
kernel(jitter, a, b, c);

// Destroy the created kernel
mkl_jit_destroy(jitter);
```

JIT DGEMM, SGEMM on Intel® Xeon® Platinum Processor





Performance results are based on testing as of **July 9, 2018** and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.
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. For more complete information, see Performance Benchmark Test Disclosure.
Testing by Intel as of July 9, 2018. Configuration: Intel® Xeon® Platinum 8180 H0 205W 2x28@2.5GHz 192GB DDR4-2666

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.
For more complete information about compiler optimizations, see our Optimization Notice.

Faster, Scalable Code with Intel® Math Kernel Library

- Speeds computations for scientific, engineering, financial and machine learning applications by providing highly optimized, threaded, and vectorized math functions
- Provides key functionality for dense and sparse linear algebra (BLAS, LAPACK, PARDISO), FFTs, vector math, summary statistics, deep learning, splines and more
- Dispatches optimized code for each processor automatically without the need to branch code
- Optimized for single core vectorization and cache utilization
- Automatic parallelism for multi-core and many-core
- Scales from core to clusters
- Available at no cost and royalty free
- Great performance with minimal effort!

Intel® MKL Offers…

- **Dense and SPARSE Linear Algebra**
- **Fast Fourier Transforms**
- **Vector Math**
- **Vector RNGs**
- **Fast Poisson Solver**
- **And More!**

Available as standalone or as a part of [Intel® Parallel Studio XE] and [Intel® System Studio]

Intel® Architecture Platforms

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

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

What’s Inside Intel® MKL

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

FFTs
- Multidimensional
- FFTW interfaces
- Cluster FFT

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

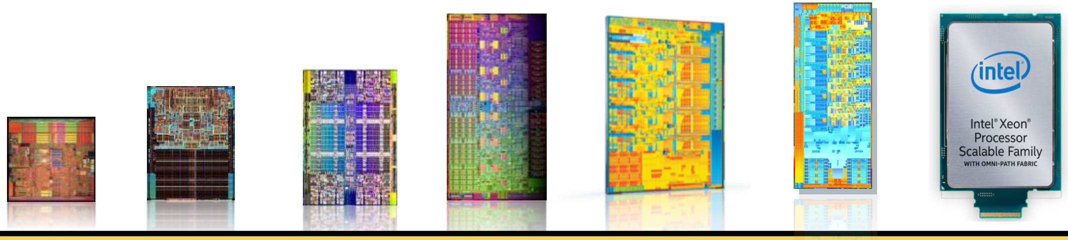
Summary Statistics
- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

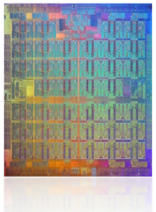
Vector Math
- Trigonometric
- Hyperbolic
- Exponential
- Log
- Power
- Root

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

Automatic Dispatching to Tuned ISA-specific Code Paths

More cores → More Threads → Wider vectors





	Intel® Xeon® Processor 64-bit	Intel® Xeon® Processor 5100 series	Intel® Xeon® Processor 5500 series	Intel® Xeon® Processor 5600 series	Intel® Xeon® Processor E5-2600 v2 series	Intel® Xeon® Processor E5-2600 v3 series v4 series	Intel® Xeon® Scalable Processor[1]	Intel® Xeon Phi™ x200 Processor (KNL)
Up to Core(s)	1	2	4	6	12	18-22	28	72
Up to Threads	2	2	8	12	24	36-44	56	288
SIMD Width	128	128	128	128	256	256	512	512
Vector ISA	Intel® SSE3	Intel® SSE3	Intel® SSE4-4.1	Intel® SSE 4.2	Intel® AVX	Intel® AVX2	Intel® AVX-512	Intel® AVX-512

1. Product specification for launched and shipped products available on ark.intel.com.

What's New for Intel® MKL 2019?

Just-In-Time Fast Small Matrix Multiplication

- Improved speed of S/DGEMM for Intel® AVX2 and Intel® AVX-512 with JIT capabilities

Sparse QR Solvers

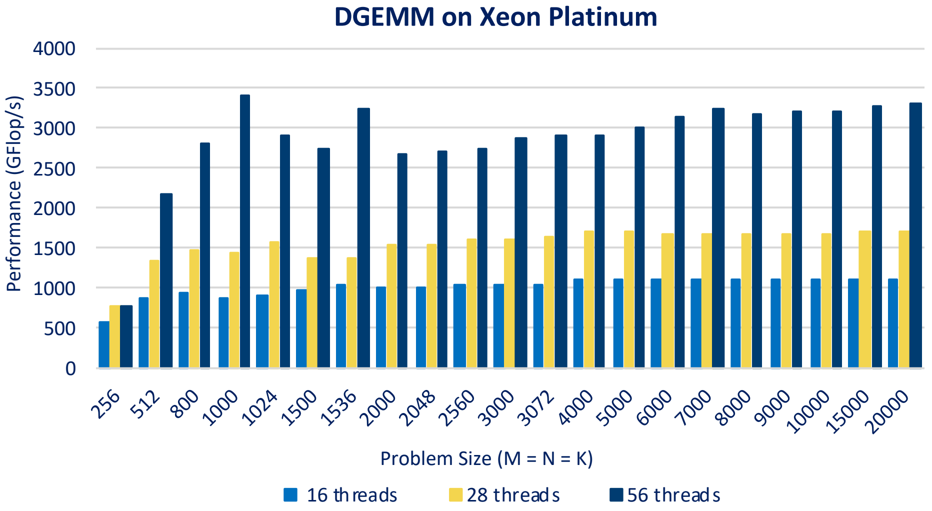
- Solve sparse linear systems, sparse linear least squares problems, eigenvalue problems, rank and null-space determination, and others

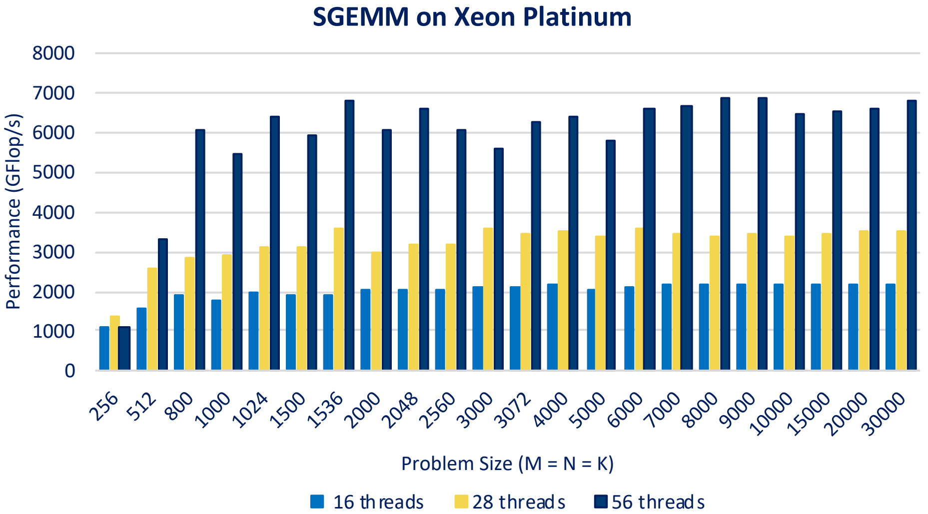
Generate Random Numbers for Multinomial Experiments

- Highly optimized multinomial random number generator for finance, geological and biological applications

Performance Benefits for the latest Intel Architectures

DGEMM, SGEMM Optimized by Intel® Math Kernel Library 2019 Gold for Intel® Xeon® Platinum Processor





Configuration: Intel® Xeon® Platinum 8180 H0 205W 2x28@2.5GHz 192GB DDR4-2666

Benchmark Source: Intel® Corporation.

Intel® MKL 11.0 - 2018 Noteworthy Enhancements

- Conditional Numerical Reproducibility (CNR)
- Intel® Threading Building Blocks (TBB) Composability
- Intel® Optimized High Performance Conjugate Gradient (HPCD) Benchmark
- Small GEMM Enhancements (Direct Call) and Batch
- Compact GEMM and LAPACK Support
- Sparse BLAS Inspector-Executor API
- Extended Cluster Support (MPI wrappers and macOS*)
- Parallel Direct Sparse Solver for Clusters
- Extended Eigensolvers

intel®

Software