Intel® Developer Workshop
Intel oneMKL

Dr Aleksandra Krawiec

April 2026

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

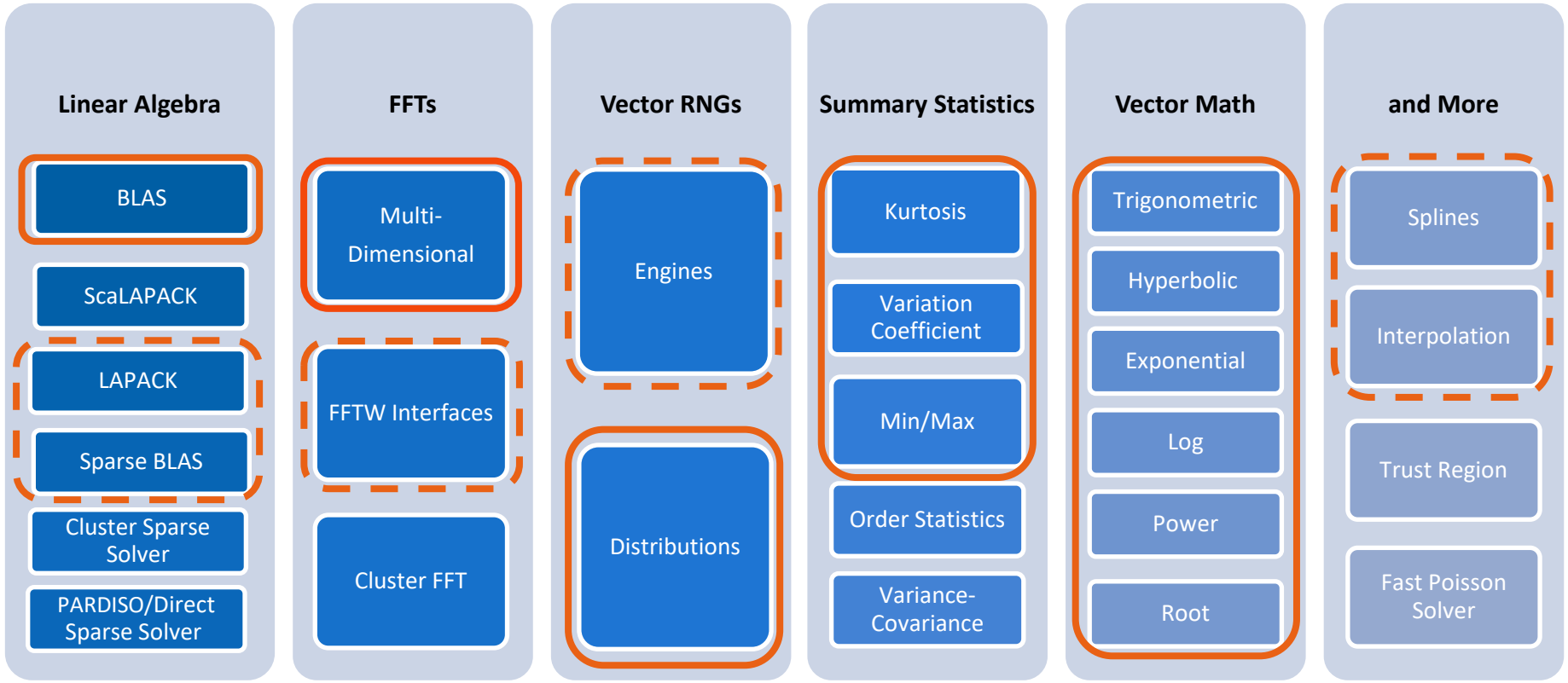
Agenda

1. Introduction to oneMKL
2. Dense BLAS and LAPACK
3. DFT
4. Sparse BLAS
5. Sparse solvers

Introduction to oneMKL

- What’s oneMKL and what’s inside?
- How to use oneMKL
- Performance numebrs
- What’s new in recent releases
- Other performance libraries

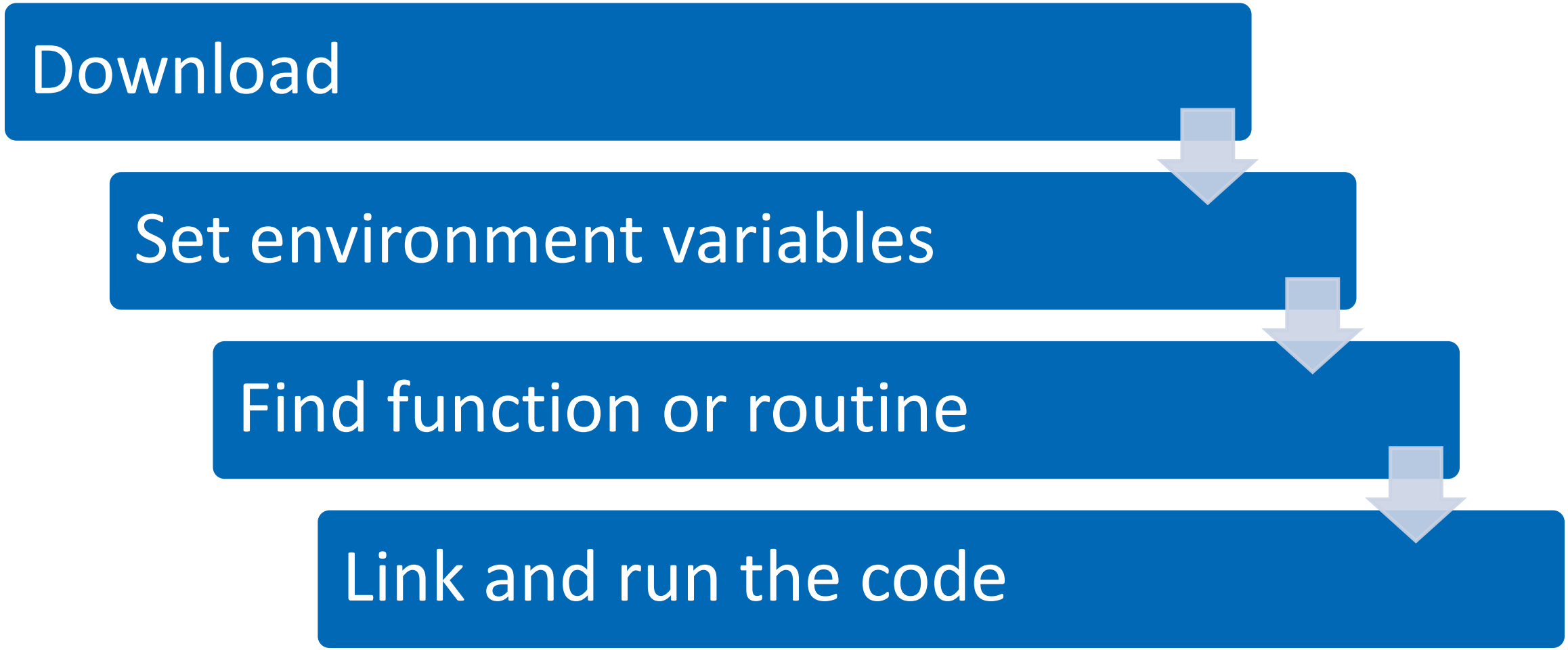
What's inside oneMKL?



Introduction to oneMKL

- What’s oneMKL and what’s inside?
- How to use oneMKL
- Performance numebrs
- What’s new in recent releases
- Other performance libraries

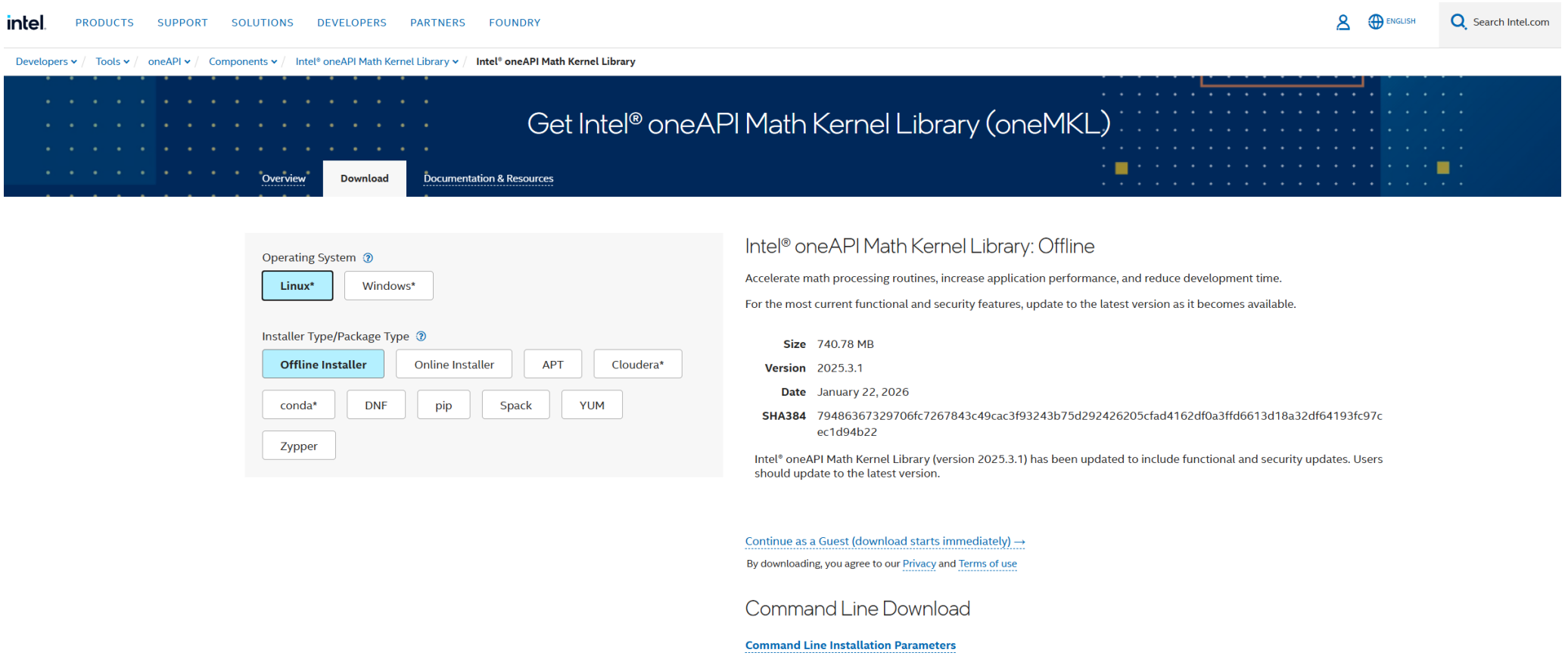
How to run oneMKL code?



Download oneMKL
(standalone or as part of the toolkit)

[Get Intel® oneAPI Math Kernel Library]

[Get the Intel® oneAPI Base Toolkit]



Set environment variables

Linux:

source /opt/intel/oneapi/setvars.sh

Setting Environment Variables

Windows:

"C:\Program Files (x86)\Intel\oneAPI\setvars.bat“

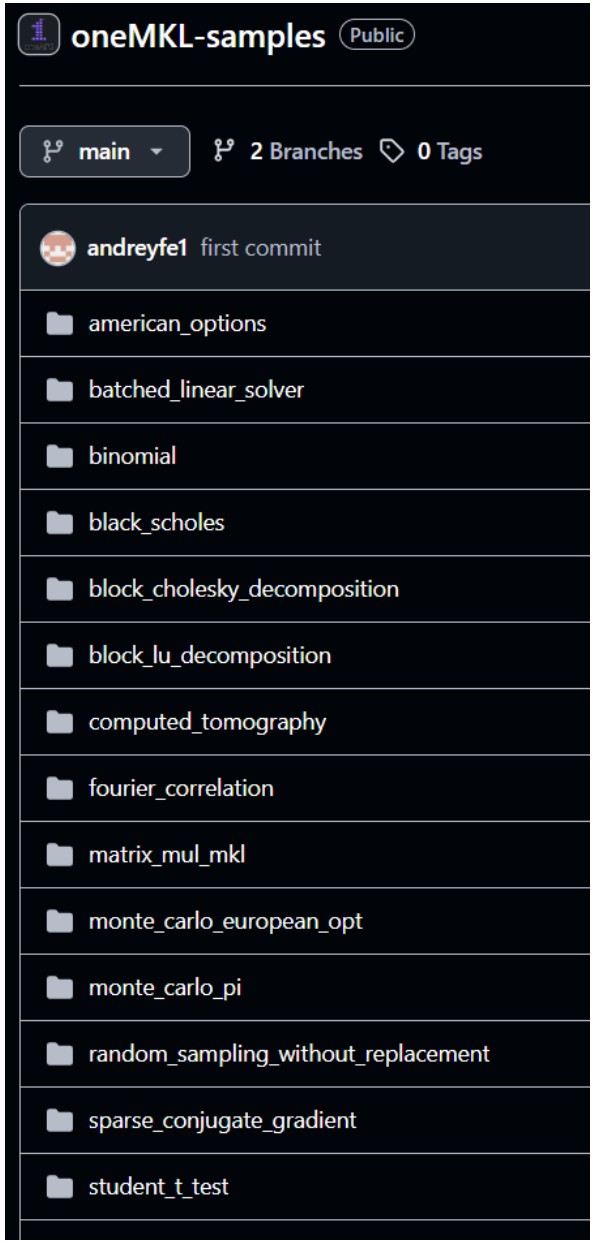
Setting Environment Variables

Find function or routine

- Developer Reference for Intel® oneAPI Math Kernel Library – C
- Developer Reference for Intel® oneAPI Math Kernel Library – Fortran
- Intel® oneAPI Math Kernel Library - Data Parallel C++ Developer...

Where to find examples?

- Github repository with examples
  https://github.com/oneapi-src/oneMKL-samples
- Inside the downloaded package:
  - Linux:
    /opt/intel/oneapi/mkl/latest/share/doc/mkl/examples
  - Windows:
    C:\Program Files (x86)\Intel\oneAPI\mkl\latest\share\doc\mkl\examples
- Developer reference: Appendix: Code Examples



Linking to oneMKL

Intel® oneAPI Math Kernel Library (oneMKL) Link Line Advisor v6.27

Reset

Select Intel® product:	oneMKL 2025
Select OS:	Linux*
Select programming language:	C/C++
Select compiler:	Intel(R) oneAPI DPC++/C++
Select architecture:	Intel(R) 64
Select dynamic or static linking:	Dynamic
Select interface layer:	C API with 64-bit integer
Select threading layer:	OpenMP threading
Select OpenMP library:	Intel(R) (libiomp5)
Enable OpenMP offload feature to GPU:	☐
Select cluster library:	☑ Parallel Direct Sparse Solver for Clusters (BLACS required) ☐ Cluster Discrete Fast Fourier Transform (BLACS required) ☐ ScaLAPACK (BLACS required) ☑ BLACS
Select MPI library:	Intel(R) MPI
Select the Fortran 95 interfaces:	☐ BLAS95 ☐ LAPACK95
Select SYCL domain library:	<Select Domain>
Select SYCL distributed library:	☐ DFT
Link with Intel® oneMKL libraries explicitly:	☑
Link with SYCL debug runtime compatible libraries:	☐

[Link Line Advisor for Intel® oneAPI Math Kernel Library]

Compiler options:

```
-DMKL_ILP64  -I"${MKLROOT}/include"
```

Use this link line:

```
-L${MKLROOT}/lib -lmkl_intel_ilp64 -lmkl_intel_thread -lmkl_core -
lmkl_blacs_intelmpi_ilp64 -liomp5 -lpthread -lm -ldl
```

LP64 vs ILP64 interfaces

Using the ILP64 Interface vs. LP64 Interface

LP64

- index arrays with the 32-bit integer type
- MKL_INT is 32 bits
- Compilation and linking
  - icx / icpx: -I"${MKLROOT}/include“
  - ifx: -I"${MKLROOT}/include“
  - -lmkl_intel_**lp64**, -qmkl-**lp64**

ILP64

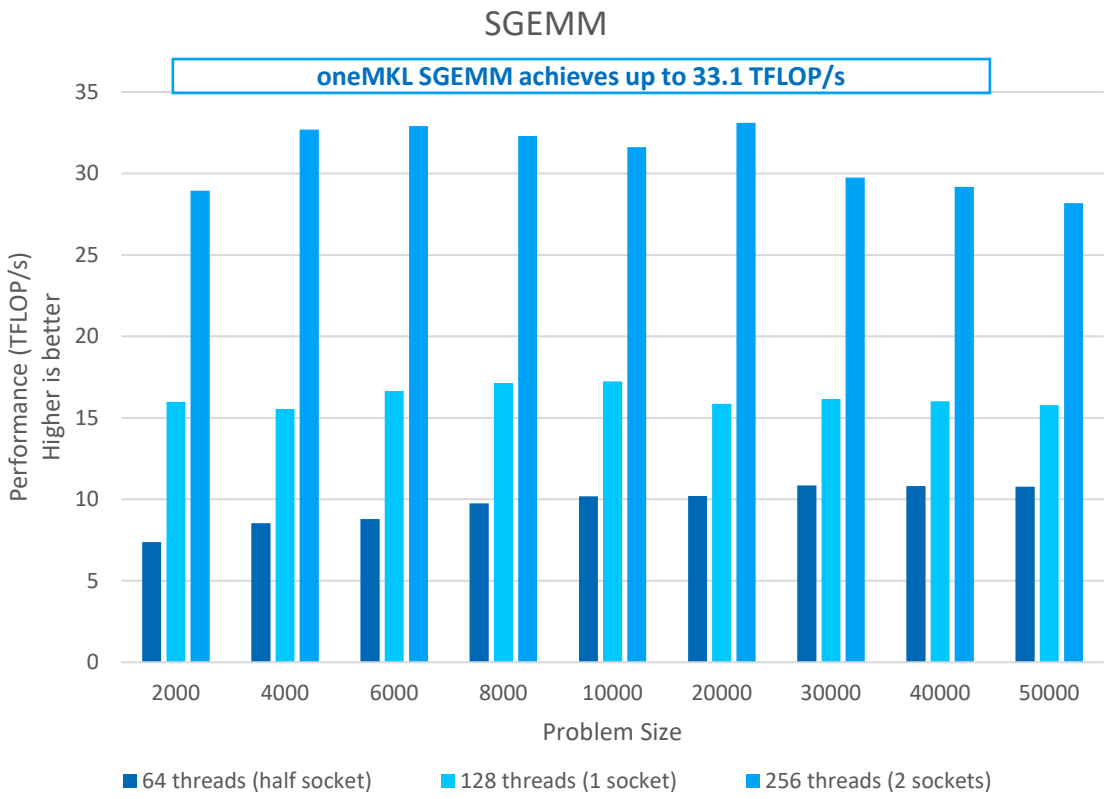
- use the 64-bit integer type (necessary for indexing large arrays, with more than 2^{31}-1 elements)
- MKL_INT is 64 bits
- Compilation and linking
  - icx / icpx : -DMKL_**ILP64** -I"${MKLROOT}/include“
  - ifx: -i8 -I"${MKLROOT}/include“
  - -lmkl_intel_**ilp64**, -qmkl-**ilp64**

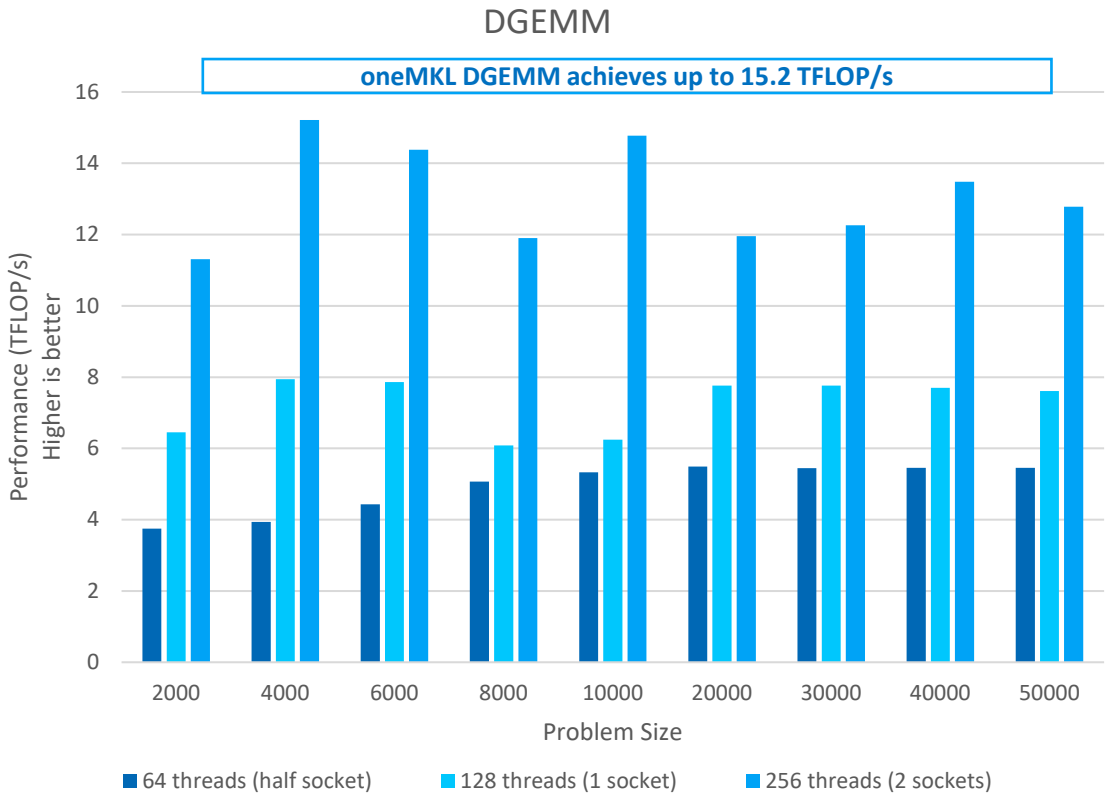
Introduction to oneMKL

- What’s oneMKL and what’s inside?
- How to use oneMKL
- Performance numebrs
- What’s new in recent releases
- Other performance libraries

oneMKL General Matrix Multiplication (GEMM) Performance

Intel® oneAPI Math Kernel Library (oneMKL) 2025.3.0 GEMM on Intel® Xeon® 6 Processor with P-Cores





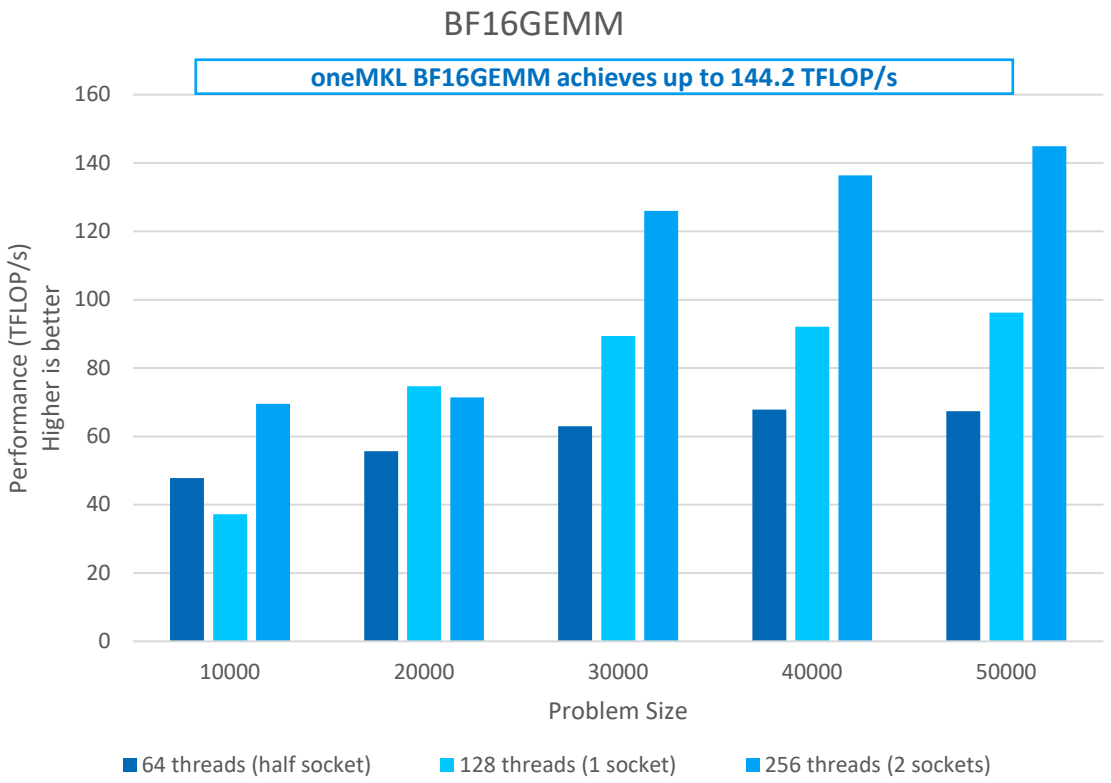
Testing Date: Performance results are **based on testing by Intel as of October 16, 2025** and may not reflect all publicly available security updates.

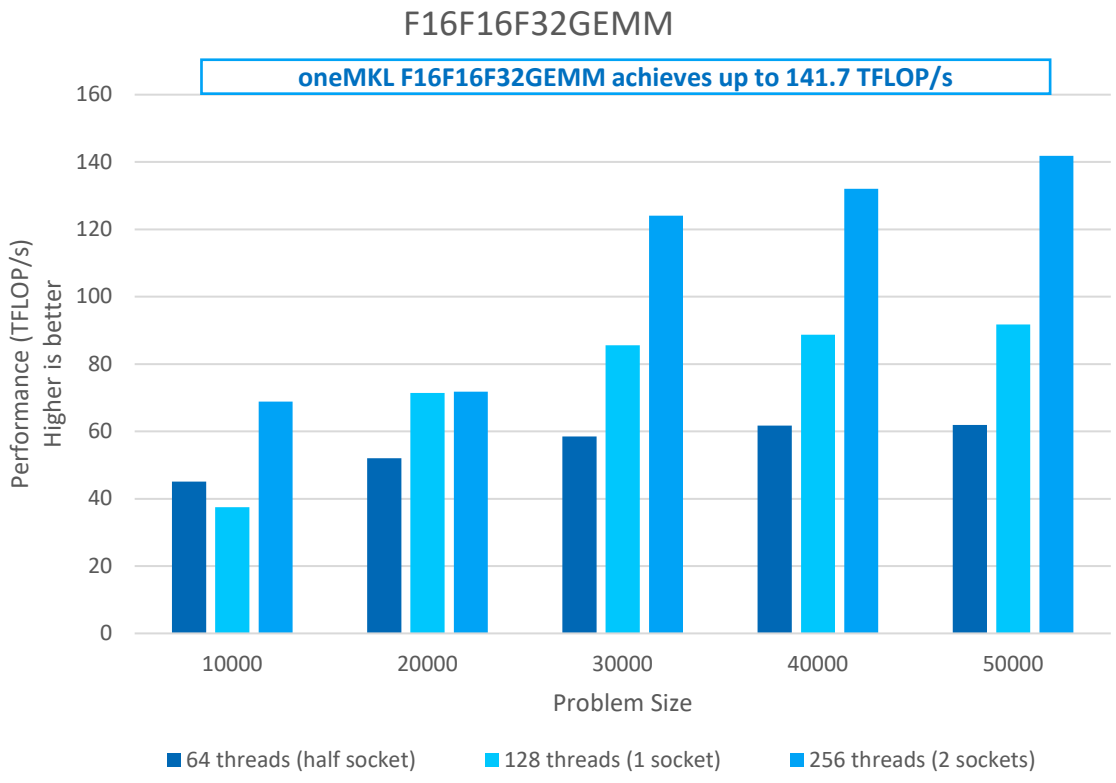
Configuration: : 1-node, 2x Intel® Xeon® 6980P processor on AvenueCity platform with 1536 GB (24 slots/ 64GB/ 8800) total DDR5 memory, ucode 0x10003d0, HT on, Turbo on, Ubuntu 24.04 LTS, 6.8.0-79-generic, 1x Micron_7450_MTFDKBG960TFR 960 GB; Intel® oneAPI Math Kernel Library 2025.3.0 (oneMKL). SGEMM & DGEMM performance for square matrix dimensions.

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
Your costs and results may vary.
Intel technologies may require enabled hardware, software or service activation.
© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

oneMKL General Matrix Multiplication (GEMM) Performance

Intel® oneAPI Math Kernel Library (oneMKL) 2025.3.0 GEMM on Intel® Xeon® 6 Processor with P-Cores





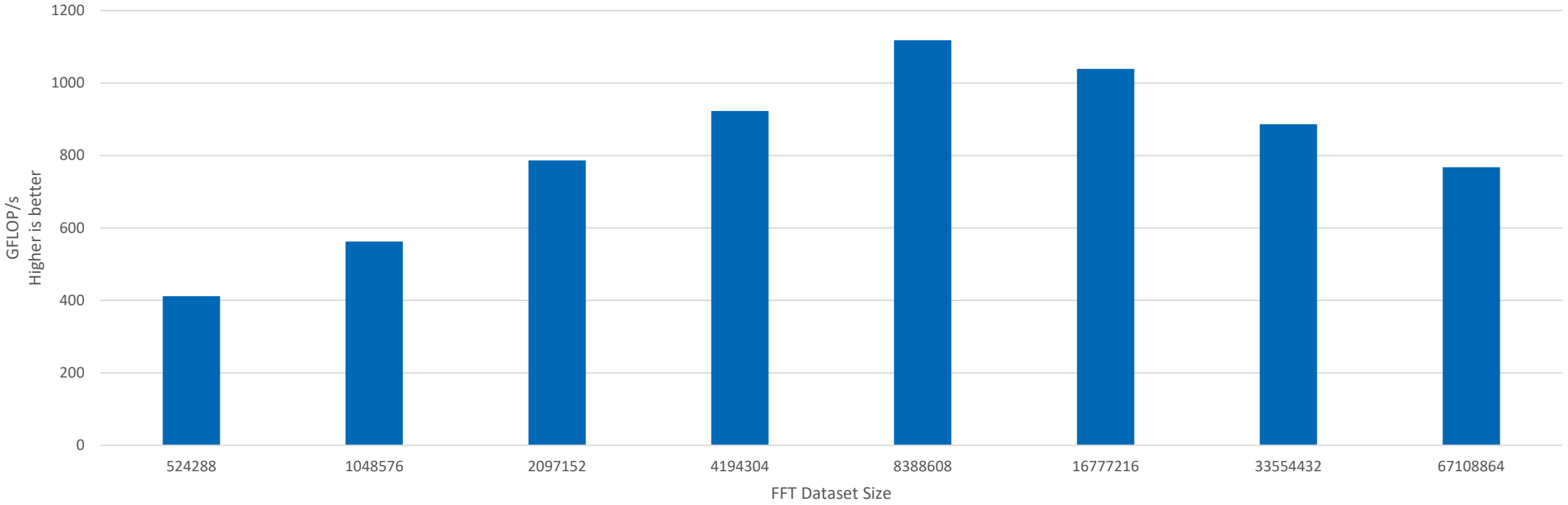
Testing Date: Performance results are **based on testing by Intel as of October 16, 2025** and may not reflect all publicly available security updates.

Configuration: : 1-node, 2x Intel® Xeon® 6980P processor on AvenueCity platform with 1536 GB (24 slots/ 64GB/ 8800) total DDR5 memory, ucode 0x10003d0, HT on, Turbo on, Ubuntu 24.04 LTS, 6.8.0-79-generic, 1x Micron_7450_MTFDKBG960TFR 960 GB; Intel® oneAPI Math Kernel Library 2025.3.0 (oneMKL). BF16GEMM & F16F16F32GEMM performance for square matrix dimensions.

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
Your costs and results may vary.
Intel technologies may require enabled hardware, software or service activation.
© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Fast Fourier Transform (FFT) Performance for Large Problem Sizes

Intel® oneAPI Math Kernel Library (oneMKL) 2025.3.0 FFT on Intel® Xeon® 6 Processor with P-Cores



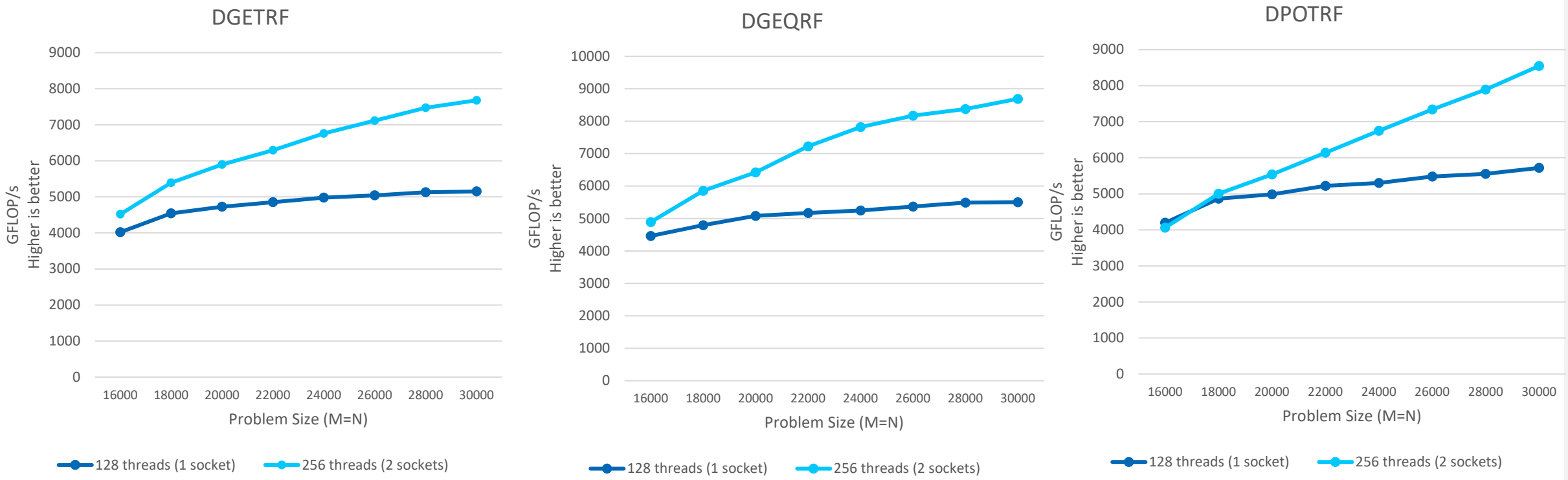
Testing Date: Performance results are **based on testing by Intel as of October 16, 2025** and may not reflect all publicly available security updates.

Configuration: : 1-node, 2x Intel® Xeon® 6980P processor on AvenueCity platform with 1536 GB (24 slots/ 64GB/ 8800) total DDR5 memory, ucode 0x10003d0, HT on, Turbo on, Ubuntu 24.04 LTS, 6.8.0-79-generic, 1x Micron_7450_MTFDKBG960TFR 960 GB; Intel® oneAPI Math Kernel Library 2025.3.0 (oneMKL); Performance measured for forward out-of-place 1-Dimensional FFT with double precision data sets. In general, size of input and size of L1, L2 and L3 caches affect performance of FFT algorithm.

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
Your costs and results may vary.
Intel technologies may require enabled hardware, software or service activation.
© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

oneMKL Linear Algebra Package (LAPACK) Performance

Intel® oneAPI Math Kernel Library (oneMKL) 2025.3.0 LAPACK on Intel® Xeon® 6 Processor with P-Cores



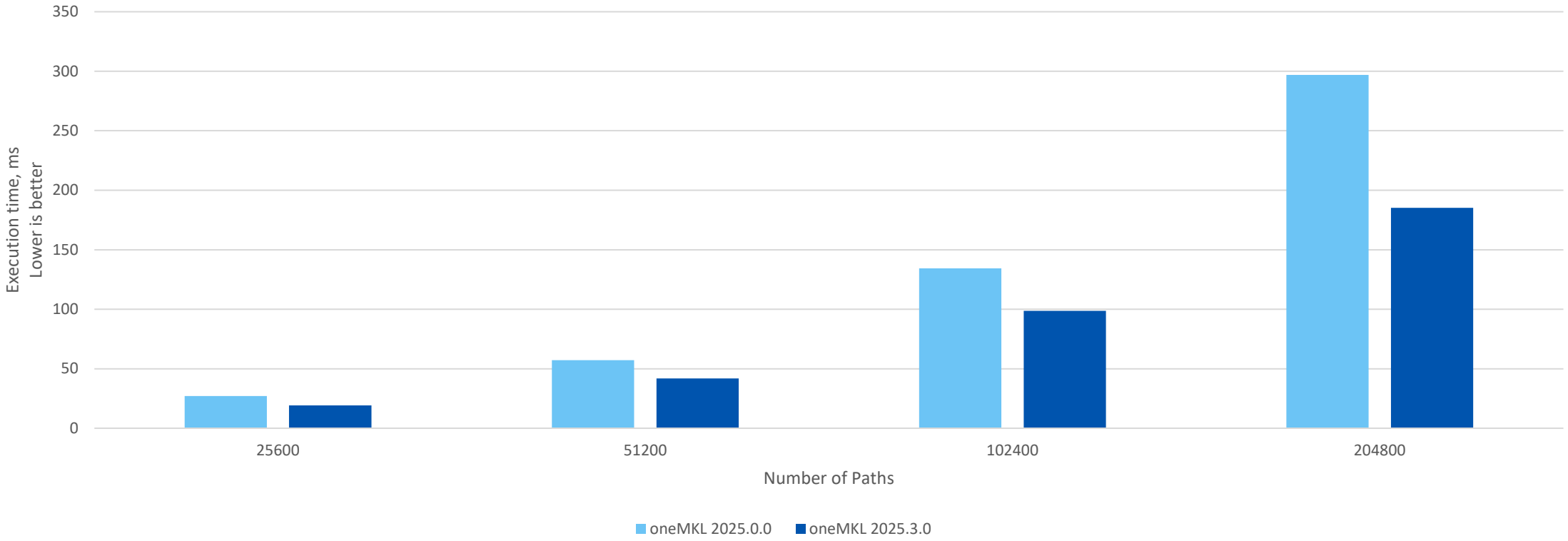
Testing Date: Performance results are **based on testing by Intel as of October 16, 2025** and may not reflect all publicly available security updates.

Configuration: 1-node, 2x Intel® Xeon® 6980P processor on AvenueCity platform with 1536 GB (24 slots/ 64GB/ 8800) total DDR5 memory, ucode 0x10003d0, HT on, Turbo on, Ubuntu 24.04 LTS, 6.8.0-79-generic, 1x Micron_7450_MTFDKBG960TFR 960 GB; Intel® oneAPI Math Kernel Library 2025.3.0 (oneMKL);

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
Your costs and results may vary.
Intel technologies may require enabled hardware, software or service activation.
© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Single Thread RNG American Monte Carlo Options Model Performance

Intel® oneAPI Math Kernel Library (oneMKL) 2025.3.0 vs. 2025.0.0 on Intel® Xeon® 6 Processor with P-Cores



Testing Date: Performance results are **based on testing by Intel as of October 16, 2025** and may not reflect all publicly available security updates.

Configuration: 1-node, 2x Intel® Xeon® 6980P processor on AvenueCity platform with 1536 GB (24 slots/ 64GB/ 8800) total DDR5 memory, ucode 0x10003d0, HT on, Turbo on, Ubuntu 24.04 LTS, 6.8.0-79-generic, 1x Micron_7450_MTFDKBG960TFR 960 GB; Intel® oneAPI Math Kernel Library 2025.3.0; Intel® oneAPI Math Kernel Library 2025.0.0. Performance measured with mrg32k3a random number generator and double precision gaussian distribution using American Options with Monte Carlo benchmark with single thread.

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
Your costs and results may vary.
Intel technologies may require enabled hardware, software or service activation.
© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Introduction to oneMKL

- What’s oneMKL and what’s inside?
- How to use oneMKL
- Performance numebrs
- What’s new in recent releases
- Other performance libraries

Improvements in recent oneMKL releases

- Basic Linear Algebra Subroutines (BLAS) and Sparse BLAS
  - New Inspector-Executor C/Fortran APIs for conversion between dense matrix representation and sparse matrix format (mkl_sparse_?_convert_dense, mkl_sparse_?_convert_dense2csr/coo/csc/bsr) (2025.3)
  - Improved performance of mkl_sparse_?_trsv and mkl_sparse_?_symgs for single, complex single and complex double data types for Intel® CPUs with AVX2 or AVX512 ISA available (2025.3)
  - New routines `mkl_sparse_convert_coo`, mkl_sparse_convert_csc, mkl_sparse_?_export_coo for C and Fortran. (2025.2)
  - Improved performance for IE Sparse BLAS APIs for CSR (mkl_sparse_?_mv, mkl_sparse_?_mm) and COO (mkl_sparse_?_mv) on Intel CPUs. (2025.2)
  - Improved performance of mkl_sparse_?_mv and mkl_sparse_?_mm for BSR and CSR matrix formats on Intel® AVX2 and AVX512 ISA architectures. (2025.0, 2025.1)
- Linear Algebra Package (LAPACK)
  - Improved performance of geqrf for tall-skinny complex matrices on Intel® CPU (2025.3)
  - Improved performance of SVD and least squares solvers for complex precisions, and TRTRI for all precisions on Intel® CPUs. (2025.3)
  - Improved performance in dsyevd eigensolver and geqrf QR factorization. (2025.2)
  - Improved performance of double precision LU and QR factorizations on Intel® Xeon® processors with Intel® AVX-512 architecture and high thread count. (2025.1)
  - Improved performance of eigensolver (?syev, ?heev) on CPU for very large matrices (n>100K). (2025. 0)

Improvements in recent oneMKL releases

- Vector Mathematical Functions
  - Improved performance of single and double precision real functions ln, pow, sin, erf, tanh and atan2, for all accuracy versions on CPU (2025.1)
  - Improved performance of single and double precision complex functions sin, asin, asinh, cos, acos, acosh, tanh, and atanh, for all accuracy versions on CPU (2025.1)
- Sparse Solvers
  - Improved load balancing when running on multiple threads in the factorization phase of oneMKL Pardiso (2025.3)
  - Improved performance in the factorization phase of PARDISO. (2025.2)
  - Improved iterative refinement feature of PARDISO and added optional printing of iterative refinement information (2025.0)

Introduction to oneMKL

- What’s oneMKL and what’s inside?
- How to use oneMKL
- Performance numebrs
- What’s new in recent releases
- Other performance libraries

Intel® Integrated Performance Primitives (Intel® IPP)

Your Building Blocks for Image, Signal & Data Processing Applications

Ready-to-use, domain-specific functions to accelerate image & signal processing, and data compression computation tasks

Image Processing

- Medical Imaging
- Computer Vision
- Digital Surveillance
- ADAS
- Automated Sorting
- Biometric Identification
- Visual Search

Signal Processing

- Games
- Echo Cancellation
- Telecommunications
- Energy

Data Compression

- Data Centers
- Data Storage
- Data Transmission
- Enterprise data management
- Smart Cards/wallets

What's New in IPP 2022.3.0:

- Added new APIs ippsAddProductC_32fc and ippsAddProductC_64fc for adding a product of a vector and a constant to an accumulator vector (for complex numbers and optimized on AVX2 and AVX512)
- Optimized IPP Warp Affine functionality (ippiWarpAffine API) for 16-bit and 32-bit floating point data with nearest neighbor interpolation on Intel® AVX512 hardware.
- Custom Library Tool is now able to generate hardware-specific libraries without function lists.
- Added LZ4 v1.10.0 dictionary-based data compression function.

Learn More & Download

Intel® Cryptography Primitives Library

Secure, fast, lightweight building blocks for cryptography, optimized for Intel CPUs

Common cross-platform and cross-operating system API for routines commonly used for cryptographic operations:

- Security (constant-time execution for secret processing functions)
- Designed for the small footprint size
- Optimized for Intel CPUs and instruction set architectures including hardware cryptography instructions
- Configurable CPU dispatching for the best performance
- Kernel mode compatibility
- Thread-safe design
- Data protection in the post-quantum era
- FIPS 140 Compliance

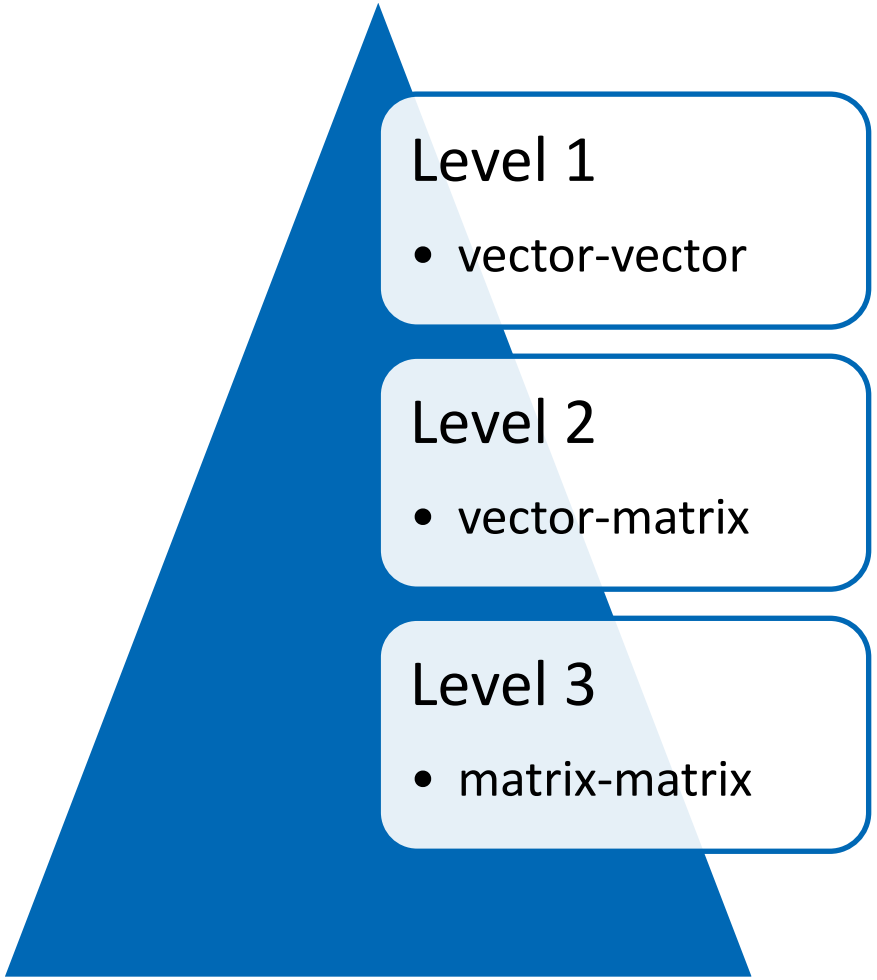
What's New in CPL 2025.3.0

- Added support for ML-KEM post-quantum algorithm with key generation, encapsulation and decapsulation functionalities implemented according to FIPS 203
- Added API for hash squeezing for extendable-output functions (XOF) from the Keccak family
- Optimized performance of hash algorithms (SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256)
- Optimized stack memory usage for Leighton-Micali Signatures (LMS) verification algorithm
- Added split HKDF API HKDF_Extract() and HKDF_Expand() for more granular usage of HKDF functionality

Learn More & Download

Dense BLAS & LAPACK

BLAS & naming convention



Indicates data type
s – real (single precision)
c – complex (single precision)
d – real (double precision)
z – complex (double precision)

Provides additional details, e.g.
c – conjugated vector
mv – matrix-vector product
rk – rank-k update of a matrix

<character> <name> <mod>

Level 1: indicates operation type (e.g. ?dot)
Levels 2, 3: reflects matrix argument type, e.g.
ge – general matrix
sy – symmetric matrix
sb – symmetric band matrix
he – Hermitian matrix
tr – triangular matrix

Examples of BLAS functions in cBLAS

Level 1

- cblas_?dot – dot product of vectors
- cblas_?nrm2 - Euclidean norm
- cblas_i?amax - Index of the maximum absolute value element of a vector

Level 2

- cblas_?gemv - Matrix-vector product using a general matrix
- cblas_?gbmv - Matrix-vector product using a general band matrix
- cblas_?trmv - Matrix-vector product using a triangular matrix

Level 3

- cblas_?gemm - Computes a matrix-matrix product with general matrices
- cblas_?hemm - Computes a matrix-matrix product where one input matrix is Hermitian.
- cblas_?symm - Computes a matrix-matrix product where one input matrix is triangular.

Row-Major vs. Column-Major Layout

Let N=5, M=4

$$A=\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{bmatrix}$$

- Row Major Layout (lda = N)

$\{a_{1,1}a_{1,2} \ldots a_{1,N}a_{2,1}a_{2,2} \ldots a_{2,N} \ldots \ldots a_{N,1}a_{N,2} \ldots a_{M,N}\}$

- Column Major Layout (lda = M)

$\{a_{1,1}a_{2,1} \ldots a_{M,1}a_{1,2}a_{2,2} \ldots a_{M,2} \ldots \ldots a_{1,N}a_{2,N} \ldots a_{M,N}\}$

*lda - leading dimension

Matrix storage schemes for BLAS

FULL STORAGE

a_{ij} is stored in the array element $a[i - 1 + (j - 1) * lda]$

uplo = ‘F’, ‘U’, ‘L’,

PACKED STORAGE

uplo = ‘U’
a_{ij} is stored in the array element
$ap[i - 1 + j(j - 1)/2]$ for $i \leq j$

uplo = ‘L’
a_{ij} is stored in the array element
$ap[i - 1 + (2n - j) * (j - 1)/2]$ for $i \geq j$

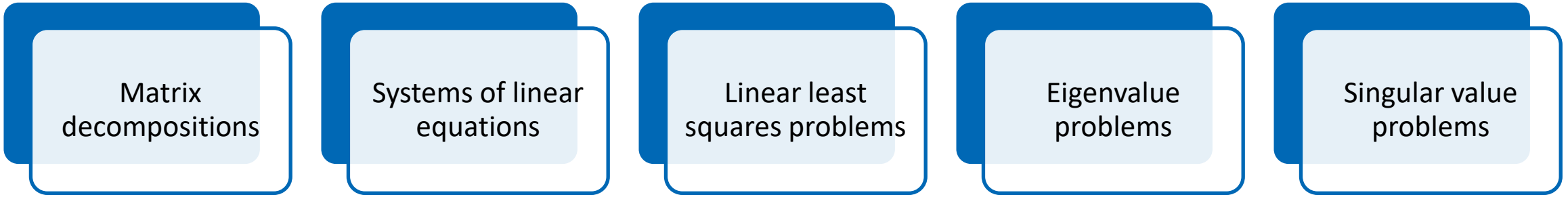
BAND STORAGE

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} \\ 0 & 0 & a_{4,3} & a_{4,4} \end{bmatrix} \Rightarrow \begin{bmatrix} * & a_{1,2} & a_{2,3} & a_{3,4} \\ a_{1,1} & a_{2,2} & a_{3,3} & a_{4,4} \\ a_{2,1} & a_{3,2} & a_{4,3} & * \end{bmatrix}$$

m = *n* = 4, *kl* = *ku* = 1

Linear Algebra Package

Main usage domains



Solving systems of linear equations

Computational routines

- ?getrs - Solves a system of linear equations with an LU-factored square coefficient matrix, with multiple right-hand sides. Before calling this routine, you must call ?getrf to compute the LU factorization of A
- ?sbtrs – as above, for general banded matrix
- ?potrs – as above, for symmetric positive-definite matrix
- ?trtrs – as above, for triangular matrix
- ?tptrs – as above, for triangular matrix in packed storage

Driver routines

- ?gesv - Computes the solution to the system of linear equations with a square coefficient matrix A and multiple right-hand sides.
- ?gesvx – as above, with error bounds on the solution
- ?gesvxx – as above, with extra precise iterative refinement

Symmetric eigenvalue problem

Unpacked

- ?syev - computes all the eigenvalues and, optionally, the eigenvectors of a real symmetric matrix.
- ?syevd – as above, but using divide and conquer algorithm
- ?syevx – computes selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix.
- ?syevr – as above, but using Relatively Robust Representation

Packed

- ?spev - Computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix in packed storage.
- ?spevd – as above, using divide and conquer algorithm
- ?spevx – computes selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix.

Fourier Transform

Compute an FFT in five steps:

Allocate a descriptor for the problem	call the DftiCreateDescriptor
Optionally adjust the descriptor configuration	call to the DftiSetValue
Commit the descriptor	call DftiCommitDescriptor to make the descriptor ready for the transform computation
Compute the transform	call to the DftiComputeForward/DftiComputeBackward functions as many times as needed
Deallocate the descriptor	call to the DftiFreeDescriptor

Example

```
#include "mkl_dfti.h"

DFTI_DESCRIPTOR_HANDLE my_desc1_handle = NULL;
DFTI_DESCRIPTOR_HANDLE my_desc2_handle = NULL;
MKL_LONG status;

status = DftiCreateDescriptor(&my_desc1_handle, DFTI_SINGLE, DFTI_COMPLEX, 1, 32);
status = DftiCommitDescriptor(my_desc1_handle);
status = DftiComputeForward(my_desc1_handle, c2c_data);
status = DftiFreeDescriptor(&my_desc1_handle);

status = DftiCreateDescriptor(&my_desc2_handle, DFTI_SINGLE,DFTI_REAL, 1, 32);
status = DftiCommitDescriptor(my_desc2_handle);
status = DftiComputeForward(my_desc2_handle, r2c_data);
status = DftiFreeDescriptor(&my_desc2_handle);
```

DFT Descriptor

status = DftiCreateDescriptor(

&desc_handle,

precision,

forward_domain,

dimension,

length);

Input

- precision: DFTI_SINGLE or DFTI_DOUBLE.
- forward_domain: DFTI_COMPLEX or DFTI_REAL.
- dimension
- length - length of the transform for a one-dimensional transform, lengths of each dimension for a multi-dimensional transform

Output

- desc_handle – FFT descriptor
- status - function completion status

Default settings

- Scale factor: none (that is, σ = 1)
- Number of data sets: one
- Data storage: contiguous
- Placement of results: in-place (the computed result overwrites the input data)

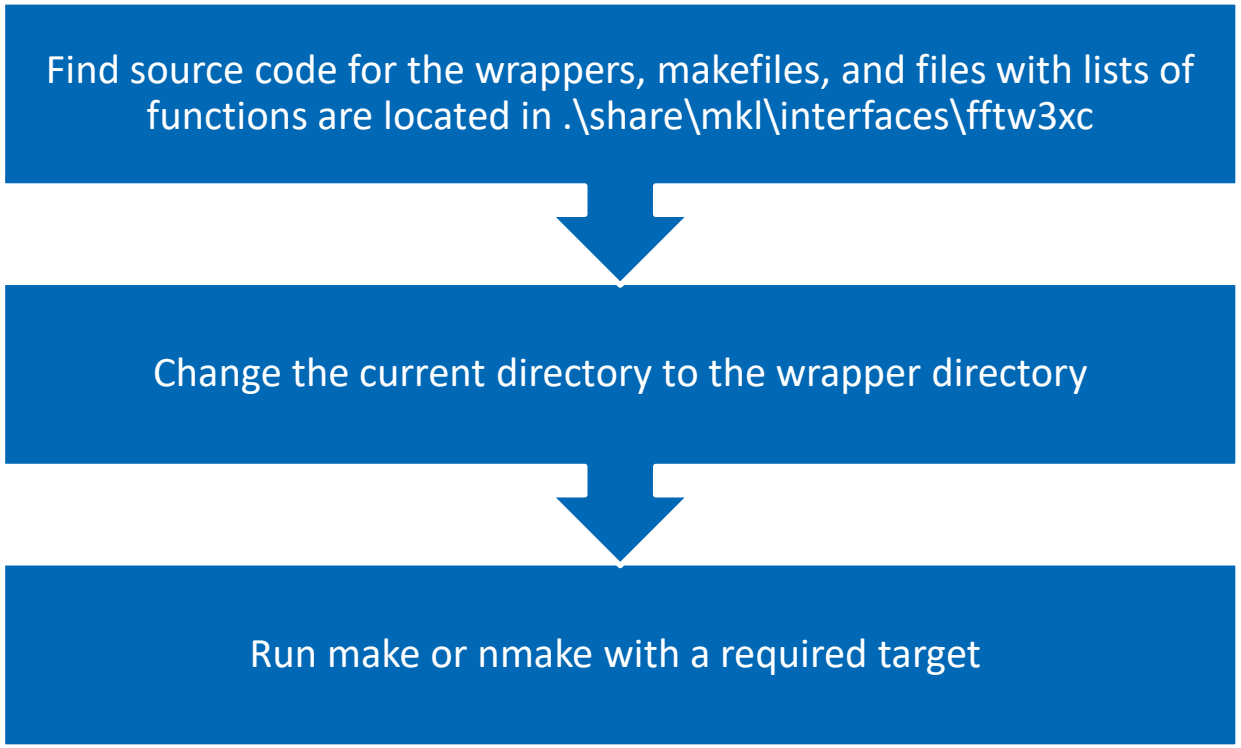
Building application with FFTW3 interface wrappers

Appendix C: FFTW Interface to Intel® Math Kernel Library

Using FFTW3 Wrappers

- Add oneMKL at the link stage
- If you recompile your application, add subdirectory include\fftw to the search path for header files to avoid FFTW3 version conflicts

Building Wrapper Library



Sparse BLAS

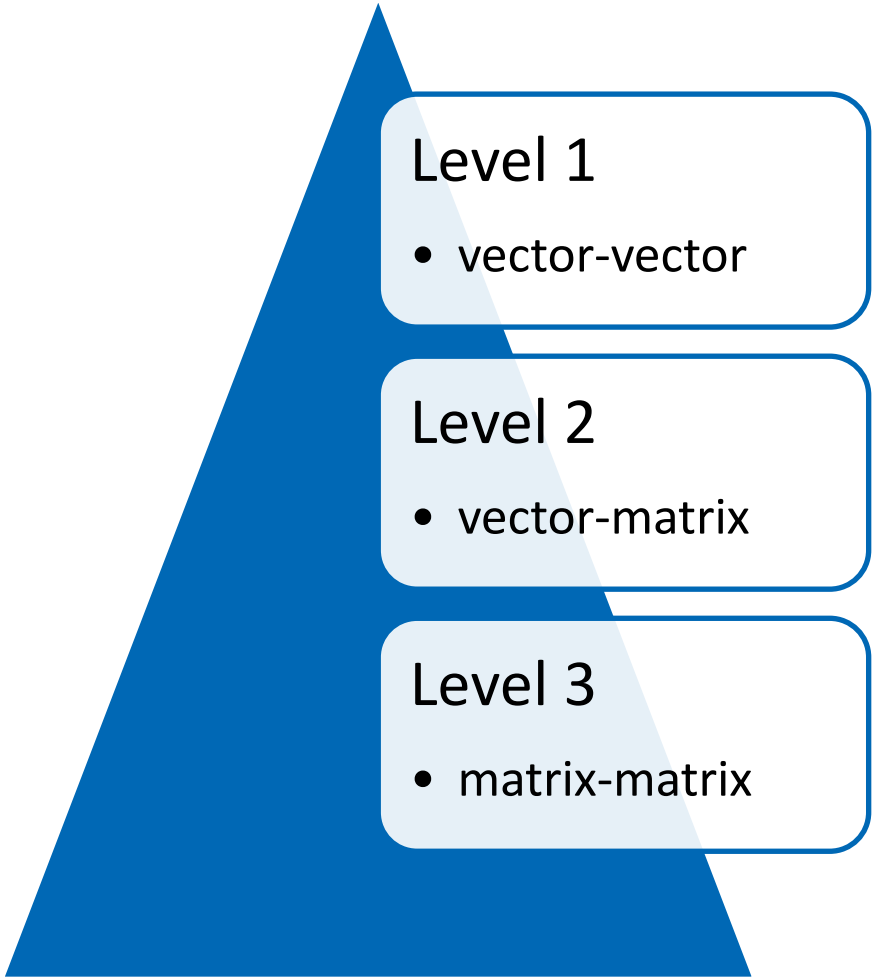
Why sparse BLAS?

Optimized to use sparsity for memory and performance

Follows naming convention from BLAS

Supports many sparse matrix formats

Sparse BLAS



- **Level 1: Vector-Vector Operations**
  - ?axpyi: y[i] = a*x[i] + y[i] (for compressed x)
  - ?dotui: complex dot product of compressed vector and full-storage vector
  - ?gthr/?sctr: gather/scatter operations
- **Level 2: Matrix-Vector Operations**
  - mkl_sparse_?_mv: sparse matrix - dense vector product
- **Level 3: Matrix-Matrix Operations**
  - mkl_sparse_?_add: Computes the sum of two sparse matrices. The result is stored in a newly allocated sparse matrix.
  - mkl_sparse_spmm: Computes the product of two sparse matrices. The result is stored in a newly allocated sparse matrix.

Inspector-Executor Sparse BLAS Routines

Analysis

- Inspect the matrix sparsity pattern
- Construct the structure of the output matrix

Execution

- Use the information from analysis phase to improve performance
- Perform computation

Supported APIs:

- Sparse matrix-vector multiplication
- Sparse matrix-matrix multiplication with a sparse or dense result
- Solution of triangular systems
- Sparse matrix addition
- …

Coordinate Matrix Storage Format

Original matrix

1	-1	*	-3	*
-2	5	*	*	*
*	*	4	6	4
-4	*	2	7	*
*	8	*	*	-5

COO representation

Values	1	-1	-3	-2	5	4	6	4	-4	2	7	8	-5
Rows	1	1	1	2	2	3	3	3	4	4	4	5	5
Columns	1	2	4	1	2	3	4	5	1	3	4	2	5

CSR / CSC format (one-based indexing, three array variation)

1	-1	*	-3	*
-2	5	*	*	*
*	*	4	6	4
-4	*	2	7	*
*	8	*	*	-5

CSR

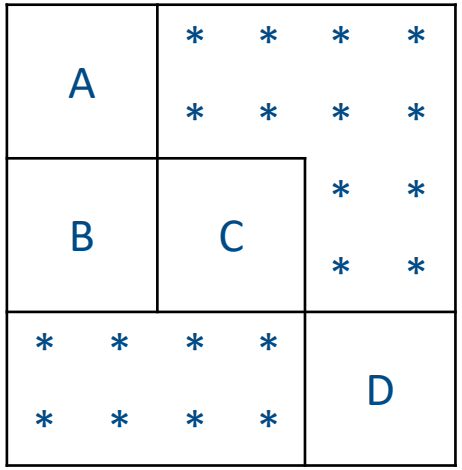
Values	1	-1	-3	-2	5	4	6	4	-4	2	7	8	-5
Columns	1	2	4	1	2	3	4	5	1	3	4	2	5
Rowindex	1	4	6	9	12	14							

CSC

Values	1	-2	-4	-1	5	8	4	2	-3	6	7	4	-5
Rows	1	2	4	1	2	3	4	5	1	3	4	2	5
Colindex	1	4	7	9	12	14							

BSR - Block compressed sparse row matrix format

2	3	*	*	*	*
5	7	*	*	*	*
4	0	1	4	*	*
3	6	9	6	*	*
*	*	*	*	0	6
*	*	*	*	2	3



0-based indexing

Values	2	3	5	7	4	0	3	6	1	4	9	6	0	6	2	3
Columns	0	0	1	2												
RowIndex	0	1	3	4												

1-based indexing

Values	2	5	3	7	4	3	0	6	1	9	4	6	0	2	6	3
Columns	1	1	2	3												
RowIndex	1	2	4	5												

Inspector-Executor sparse BLAS - example APIs

Matrix Manipulation Routines

- mkl_sparse_?_create_csr - creates a handle for a CSR-format matrix (similar for COO, CSC, BSR)
- mkl_sparse_convert_csr - converts internal matrix representation to CSR format
- mkl_sparse_?_export_csr - exports CSR matrix from internal representation
- mkl_sparse_?_set_value, mkl_sparse_?_update_values - changes a single, selected or all matrix values in internal representation
- mkl_sparse_copy - creates a copy of a matrix handle
- mkl_sparse_destroy - frees memory allocated for matrix handle

Analysis Routines

- mkl_sparse_set_mv_hint, mkl_sparse_set_mm_hint - provides estimate of number and type of upcoming matrix-vector (matrix-matrix) operations
- mkl_sparse_set_memory_hint - provides memory requirements for performance optimization purposes
- mkl_sparse_optimize - analyzes matrix structure and performs optimizations using the hints provided in the handle

Execution routines

- mkl_sparse_?_mv - computes a sparse matrix-vector product
- mkl_sparse_?_mm - computes the product of a sparse matrix and a dense matrix and stores the result as a dense matrix
- mkl_sparse_?_spmmd - computes the product of two sparse matrices and stores the result as a dense matrix
- mkl_sparse_?_trsm - solves a system of linear equations with multiple right-hand sides for a square sparse matrix
- mkl_sparse_?_dotmv - computes a sparse matrix-vector product followed by a dot product

Sparse solvers

Workshop

Sparse linear solvers

Find solution of the equation $Ax = b$

Iterative solvers

1. Start with initial guess (approximation) of the solution
2. Estimate the error of the approximation
3. Get a new approximation
4. Repeat 2. and 3. until the difference between the approximation and the true result is sufficiently small

Direct solvers

1. Factor the matrix *A* into the product of two triangular matrices (LU, LDL^T, Cholesky)
2. Perform a forward and backward triangular solve.
3. Get "exact" solution (bounded by machine precision and condition number)

Sparse linear solvers

Which one should I choose?

Iterative solvers

- Approximate solutions
- May converge slowly or not converge
- For multiple right-hand sides they need to solve each system separately
- Often need good preconditioner
- Memory efficient (store only original matrix + few vectors)

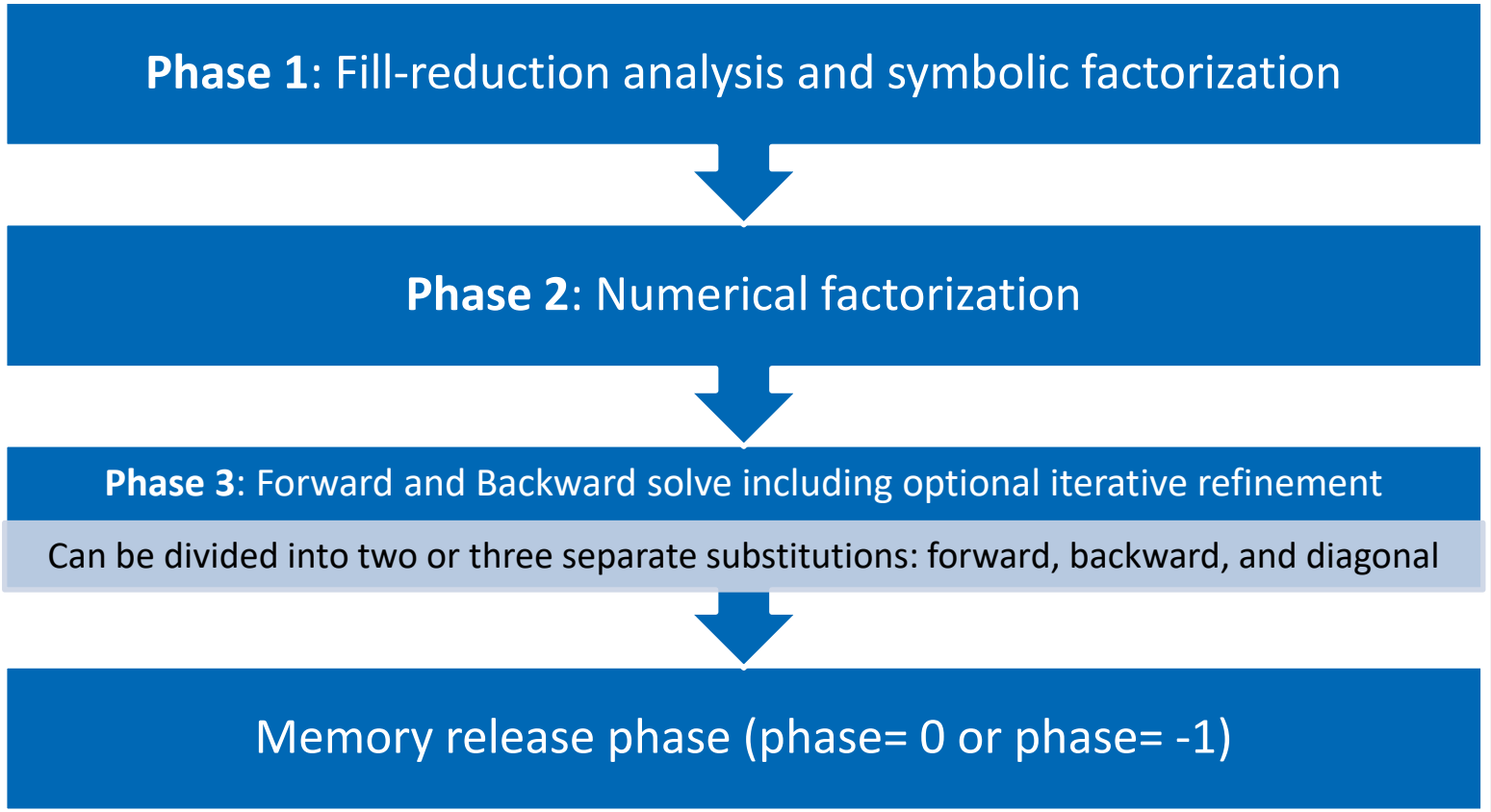
Direct solvers

- High accuracy solutions
- Always converge (if matrix is non-singular)
- Predictable, fixed number of operations
- Very efficient for many right-hand sides
- No preconditioner
- Creates new non-zeros during factorization (fill-in)
- Expensive factorization phase

oneMKL PARDISO - Parallel Direct Sparse Solver

Solve sparse linear system of equation on shared memory multiprocessors

- The solver uses a combination of left- and right-looking supernode techniques (Schenk et al.*)
- For sufficiently large problem sizes the scalability of the parallel algorithm is nearly independent of the shared-memory multiprocessing architecture.



*O. Schenk, K. Gartner, and W. Fichtner. *Efficient Sparse LU Factorization with Left-right Looking Strategy on Shared Memory Multiprocessors*. BIT, 40(1):158-176, 2000

How to use Pardiso?

PARDISO (pt, &maxfct, &mnum, &mtype, &phase, &n, a, ia, ja, &perm, &nrhs, iparm, &msglvl, b, x, &error);

mtype (matrix type)
1: real and structurally symmetric
2: real and symmetric positive definite
-2: real and symmetric indefinite
3: complex and structurally symmetric
etc.

phase (solver execution steps)
11: Analysis
12: Analysis, numerical factorization
13: Analysis, numerical factorization, solve iterative refinement
22: Numerical factorization
23: Numerical factorization, solve, iterative refinement
33: Solve, iterative refinement
331: Like 33, only forward substitution
332: Like 33, only diagonal substitution
333: Like 33, only backward substitution
0: release internal memory for L and U
-1: release all internal memory for all matrices

iparm (pardiso iparm Parameter)
Eg.
iparm[0] – use default values (set 0) or supply all values in components iparm[1] – iparm[63] (set 1)
iparm[6] - number of iterative refinement steps performed (output)
iparm[7] – Maximum number of iterative refinement steps (input, default = 2)
iparm[11] – solve with transposed or conjugated matrix A

Thank you for your attention!

Questions?


intel