Aurora Early Adopters Series

Overview of the Intel® oneAPI Math Kernel Library

Peter Caday

intel®

What's in Intel® oneMKL (Beta)?

LINEAR ALGEBRA
- BLAS
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Sparse solvers

FFT
- Multidimensional
- Cluster FFT

VECTOR RNGS
- Engines
- Distributions

SUMMARY STATISTICS
- Kurtosis
- Variation coefficient
- Order statistics
- Min/Max
- Variance-covariance

VECTOR MATH
- Trigonometric
- Hyperbolic
- Exponential
- Logarithmic
- Power
- Root

AND MORE
- Splines
- Interpolation
- Fast Poisson Solver

Legend:
- DPC++ API with GPU support
- DPC++/C/C++/Fortran API with GPU support
- C/C++/Fortran CPU support only

Zoom in: Dense Linear Algebra + FFT

BLAS
- Level 1 (vector ops)
- Level 2 (matrix-vector)
- Level 3 (matrix-matrix)
- Level 3 extensions
- Batched

LAPACK
- LU/QR
- Cholesky
- Eigensolvers
- Batch factorizations
- Utility routines
- ScaLAPACK

FFT
- Complex 1D/2D/3D
- Real-to-complex Complex-to-real 1D/2D/3D
- Cluster FFT

- DPC++/OpenMP offload with GPU support
- DPC++ API with GPU support
- CPU support only

Potential GPU Usage Models

Example: multiply double-precision matrices C ← AB

- On host (A/B/C in host memory)

```
dgemm(…, A, …, B, …, C, …);
```

- Automatic offload to GPU (A/B/C on host)

```
dgemm(…, A, …, B, …, C, …);
```

- OpenMP offload (A/B/C on host or device)

Lots of CPU ↔ GPU transfer overhead!

```
#pragma omp target variant dispatch …
dgemm(…, A, …, B, …, C, …);
```

- Manual offload (A/B/C on host or device)

```
dgemm(device_queue, …, A, …, B, …, C, …);
```

- Inside device kernel (A/B/C on device)

```
void my_kernel(…) {
    dgemm(…, A, …, B, …, C, …);
}
```

oneMKL GPU Usage Models

	Automatic offload	OpenMP offload	Manual offload	Device API
Invocation side	CPU			GPU
Data location	CPU	GPU / CPU	GPU / CPU / shared	GPU
Interface	C/C++/Fortran	C/C++/Fortran + OpenMP	DPC++	DPC++
EOY support	None	Most oneMKL GPU functionality	All oneMKL GPU functionality	Limited support (selected RNG)

Using oneMKL OpenMP Offload Interfaces

Offload: Key OpenMP Directives (C)

```
#pragma omp target data
```

Map host-side variables to device variables inside this block.

```
#pragma omp target enter data
#pragma omp target exit data
```

Map/unmap host-side variables to device variables: the two halves of #pragma omp target data.

```
#pragma omp target
```

Offload execution of block to the GPU.

```
#pragma omp target variant dispatch
```

Offload oneMKL calls inside this block to the GPU.

GEMM with oneMKL C API

```
int main() {
    long m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10;
    long sizea = lda * k, sizeb = ldb * n, sizec = ldc * n;
    double alpha = 1.0, beta = 0.0;

    // Allocate matrices
    double *A = (double *) mkl_malloc(sizeof(double) * sizea, 64);
    double *B = (double *) mkl_malloc(sizeof(double) * sizeb, 64);
    double *C = (double *) mkl_malloc(sizeof(double) * sizec, 64);

    // Initialize matrices […]
    …



            // Compute C = A * B on CPU
            cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n, k,
                        alpha, A, lda, B, ldb, beta, C, ldc);



    …
}
```

$C \leftarrow \alpha AB + \beta C$

GEMM with oneMKL C OpenMP Offload

$$C \leftarrow \alpha AB + \beta C$$

```
int main() {
    long m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10;
    long sizea = lda * k, sizeb = ldb * n, sizec = ldc * n;
    double alpha = 1.0, beta = 0.0;

    // Allocate matrices
    double *A = (double *) mkl_malloc(sizeof(double) * sizea, 64);
    double *B = (double *) mkl_malloc(sizeof(double) * sizeb, 64);
    double *C = (double *) mkl_malloc(sizeof(double) * sizec, 64);

    // Initialize matrices […]
    …
#pragma omp target data map(to:A[0:sizea],B[0:sizeb]) map(tofrom:C[0:sizec])
    {
#pragma omp target variant dispatch use_device_ptr(A, B, C) nowait
        {
            // Compute C = A * B on GPU
            cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n, k,
                        alpha, A, lda, B, ldb, beta, C, ldc);
        }
    }
    …
}
```

Use `target data map` to send matrices to the device

Use `target variant dispatch` to request GPU execution for cblas_dgemm

List mapped device pointers in the `use_device_ptr` clause

Optional `nowait` clause for asynchronous execution
Use `#pragma omp taskwait` for synchronization

GEMM with oneMKL Fortran OpenMP Offload

```
… // module files
include "mkl_omp_offload.f90"

program main
use mkl_blas_omp_offload

integer                :: m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10
integer                :: sizea = lda * k, sizeb = ldb * n, sizec = ldc * n
double                 :: alpha = 1.0, beta = 0.0
double, allocatable :: A(:), B(:), C(:)

// Allocate matrices…
allocate(A(sizea))
…

// Initialize matrices…
…
!$omp target data map(to:A(1:sizea), B(1:sizeb)) map(tofrom:C(1:sizec))
!$omp target variant dispatch use_device_ptr(A, B, C) nowait

! Compute C = A * B on GPU
call dgemm('N', 'N', m, n, k, alpha, A, lda, B, ldb, beta, C, ldc)

!$omp end target variant dispatch
!$omp end target data
…
end program
```

Module for Fortran OpenMP offload

Use `target data map` to send matrices to the device

Use `target variant dispatch` to request GPU execution for `dgemm`

List mapped device pointers in the `use_device_ptr` clause

Optional `nowait` clause for asynchronous execution
Use `!$omp taskwait` for synchronization

Using oneMKL DPC++ Interfaces

Data Parallel C++ (DPC++) Introduction

- **SYCL** is a C++-based, single-source programming language for heterogeneous computing.
- **DPC++** is SYCL + many new extensions.
  - *e.g.* pointer-based programming (Unified Shared Memory)
- Open, standards-based, multi-vendor.

https://software.intel.com/en-us/oneapi

DPC++: Key SYCL Constructs for oneMKL Usage

```
sycl::queue Q{sycl::cpu_selector{}};
sycl::queue Q{sycl::gpu_selector{}};
sycl::queue Q{device};
```

Create device queue attached to a given device or device type.

All device execution goes through a queue object.

```
void *mem = sycl::malloc_shared(bytes, Q);
void *mem = sycl::malloc_device(bytes, Q);
```

Allocate device-accessible memory. `malloc_shared` memory is also accessible from the host.

```
sycl::buffer<T,1> mem(elements);
sycl::buffer<T,1> mem(elements, hostptr);
```

Smart buffer object. Migrates memory automatically and tracks data dependencies.

Can be attached to host memory (synchronized at creation and destruction).

GEMM with oneMKL C API

$$C \leftarrow \alpha AB + \beta C$$

```
int main() {

    int64_t m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10;
    int64_t sizea = lda * k, sizeb = ldb * n, sizec = ldc * n;
    double alpha = 1.0, beta = 0.0;


    // Allocate matrices
    double *A = (double *) mkl_malloc(sizeof(double) * sizea);
    double *B = (double *) mkl_malloc(sizeof(double) * sizeb);
    double *C = (double *) mkl_malloc(sizeof(double) * sizec);

    // Initialize matrices […]
    …
    cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n, k,
                alpha, A, lda, B, ldb, beta, C, ldc);
    …
}
```

intel.

GEMM with oneMKL DPC++

```
int main() {
    using namespace oneapi::mkl;

    int64_t m = 10, n = 6, k = 8, lda = 12, ldb = 8, ldc = 10;
    int64_t sizea = lda * k, sizeb = ldb * n, sizec = ldc * n;
    double alpha = 1.0, beta = 0.0;

    sycl::queue Q{sycl::gpu_selector{}};

    // Allocate matrices
    double *A = malloc_shared<double>(sizea, Q);
    double *B = malloc_shared<double>(sizeb, Q);
    double *C = malloc_shared<double>(sizec, Q);

    // Initialize matrices […]
    …
    auto e = blas::gemm(Q, transpose::N, transpose::N, m, n, k,
                        alpha, A, lda, B, ldb, beta, C, ldc);
    …
}
```

$$C \leftarrow \alpha AB + \beta C$$

Set up GPU queue

Allocate CPU/GPU-accessible shared memory

New oneMKL DPC++ API

Computation is performed on given queue

Output **e** is a sycl::event object representing command completion

Call `e.wait()` to wait for completion

Batch Computations in oneMKL

Batching Overview

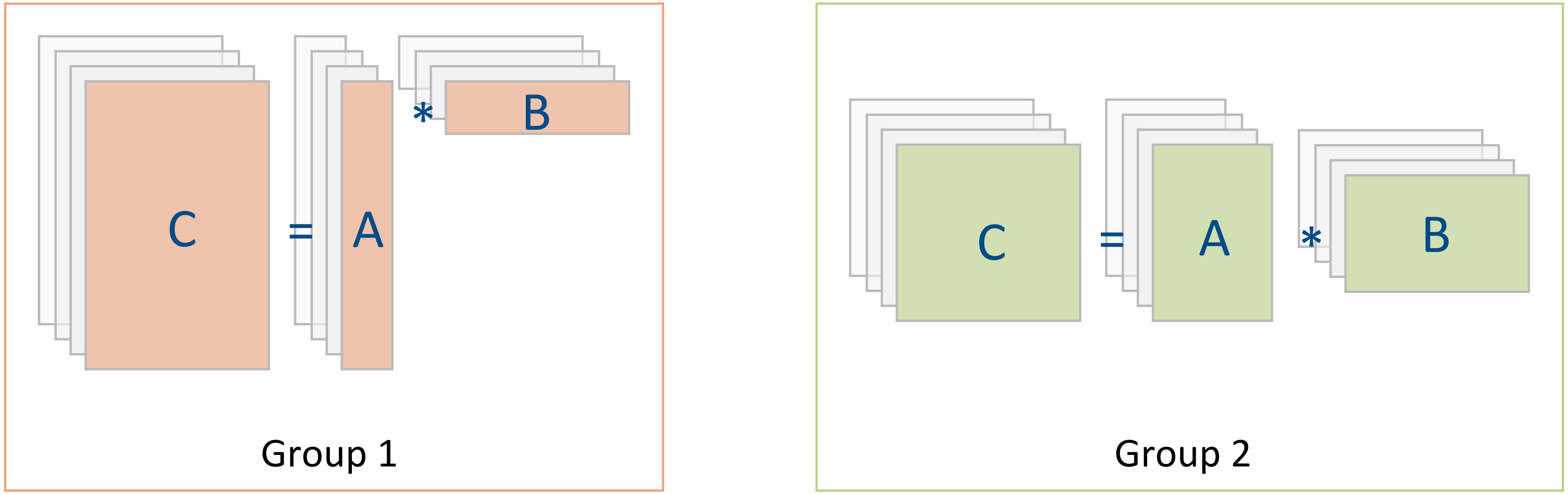
- Execute multiple **independent** operations of the same type in a single call
  - e.g. invert 100 different 8x8 matrices
- Benefits: increased parallelism, reduced overhead
- Available APIs:
  - BLAS: gemm, trsm, axpy
  - LAPACK*: LU (getrf, getri, getrs), Cholesky (potrf, potrs), QR (geqrf, orgqr, ungqr)
  - FFT: all DFTs

* only available in DPC++

BLAS/LAPACK Group APIs

Group = set of operations with identical parameters (size, transpose…)
but different matrix/vector data

Group batch APIs process one or more groups simultaneously.



BLAS/LAPACK Group APIs

Group = set of operations with identical parameters (size, transpose…)
but different matrix/vector data

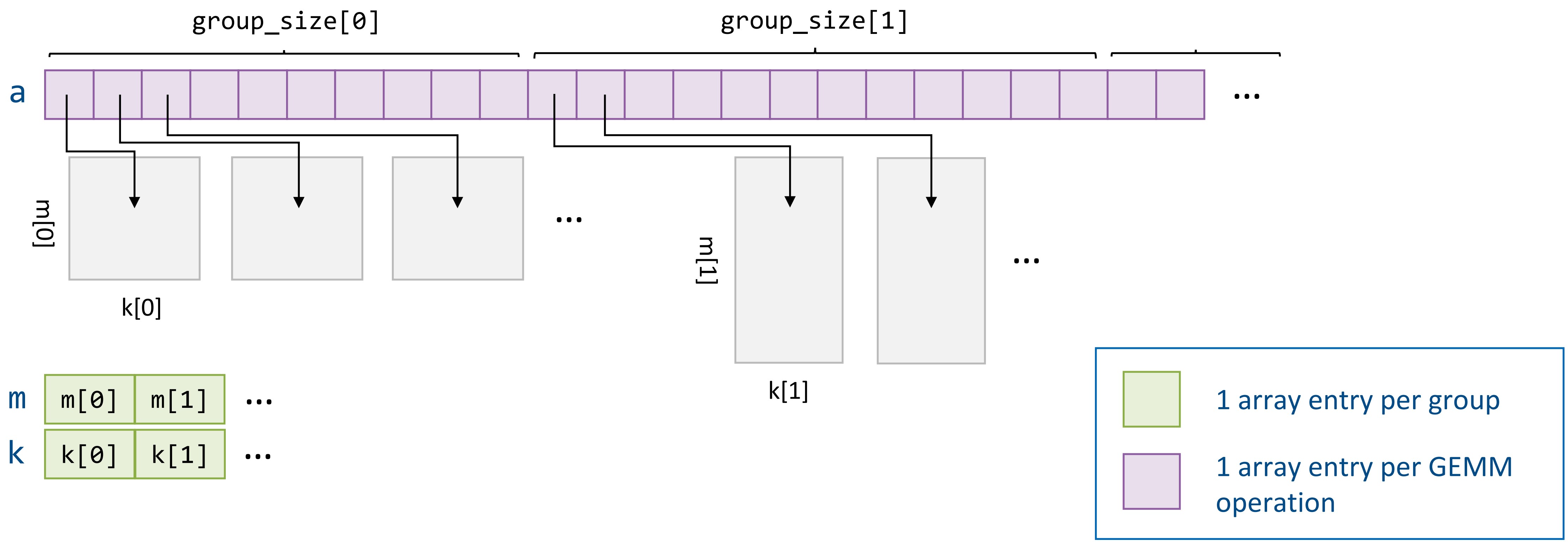
Group batch APIs process one or more groups simultaneously.

Examples:

- *n* operations, 1 group: all parameters identical
- *n* operations, *n* groups: each operation has different parameters

Example: Batch DGEMM

```
cblas_dgemm_batch(CblasColMajor, transA, transB, m, n, k, alpha, a, lda, b, ldb,
                  beta, c, ldc, num_groups, group_sizes);
```



BLAS/LAPACK Strided DPC++ APIs

- DPC++ oneMKL adds strided APIs for simple batch cases
  - **Single group**: all matrix/vector sizes, parameters are homogeneous
  - **Fixed stride** between successive matrices/vectors in batch
    - Base address + stride replaces array of pointers
    - Strides on inputs may be zero to reuse an input for all operations in the batch

Strided Batch Snippet - LU

```
#include “oneapi/mkl.hpp”
using namespace oneapi::mkl;

int64_t batch = 100;                    // 100 matrices
int64_t N = 10;                         // Matrix size – square in this example
int64_t stride = N * N;                 // 10x10 matrices are contiguous in memory
int64_t stride_piv = N;                 // Pivot entries also contiguous in memory

sycl::queue Q{sycl::gpu_selector{}};

// Allocate memory for matrices and pivot indices, as well as scratch space.
auto A_array      = sycl::malloc_shared<double>(stride * batch, Q);
auto pivot_array  = sycl::malloc_shared<double>(stride_piv * batch, Q);

auto scratch_size = lapack::getrf_batch_scratchpad_size(Q, n, n, n, stride, stride_piv, batch);
auto scratch      = sycl::malloc_shared<double>(scratch_size, Q);
…
// [Initialize A_array here]

// Batch computation
lapack::getrf_batch(Q, n, n, A_array, n, stride, pivot_array, stride_piv, scratch, scratch_size)
            .wait();
```

oneMKL Resources

Resources

Websites	https://software.intel.com/en-us/intel-mkl https://software.intel.com/en-us/oneapi/onemkl
Forum	https://software.intel.com/en-us/forums/intel-math-kernel-library
Developer Reference	https://software.intel.com/en-us/oneapi-mkl-dpcpp-developer-reference
Link Line Advisor	http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor
Benchmarks	https://software.intel.com/en-us/intel-mkl/benchmarks

Notices & Disclaimers

Questions & Answers