oneMKL Technical Advisory Board

Session 20

September 21, 2022

Agenda

- Welcoming remarks – 5 minutes
- Updates from last meeting – 10 minutes
- Device APIs for BLAS – Peter Caday (40 minutes)
- Wrap-up and next steps – 5 minutes

Updates from last meeting

- Open source oneMKL interfaces updates:
  - RNG domain now supported on Intel GPUs on Windows
  - cuBLAS supported with hipSYCL compiler
- oneAPI open governance model

Expanding oneAPI Initiative

oneAPI Value – one Programming Model for Multiple Architectures and Vendors

Freedom to Make Your Best Choice

Realize all the Hardware Value

Develop & Deploy Software with Peace of Mind

oneAPI Initiative

Specification: Community can give feedback on oneAPI spec elements

Intel Open Source: actively taking contributions

Intel-led open community; website to curate the information (oneAPI.io). **News/Blogs**

No membership – under **Intel's governance**

Technical Advisory Board (invitation-only)

oneAPI community forum

Open ecosystem-led Specification: community for driving the spec; free to become a member

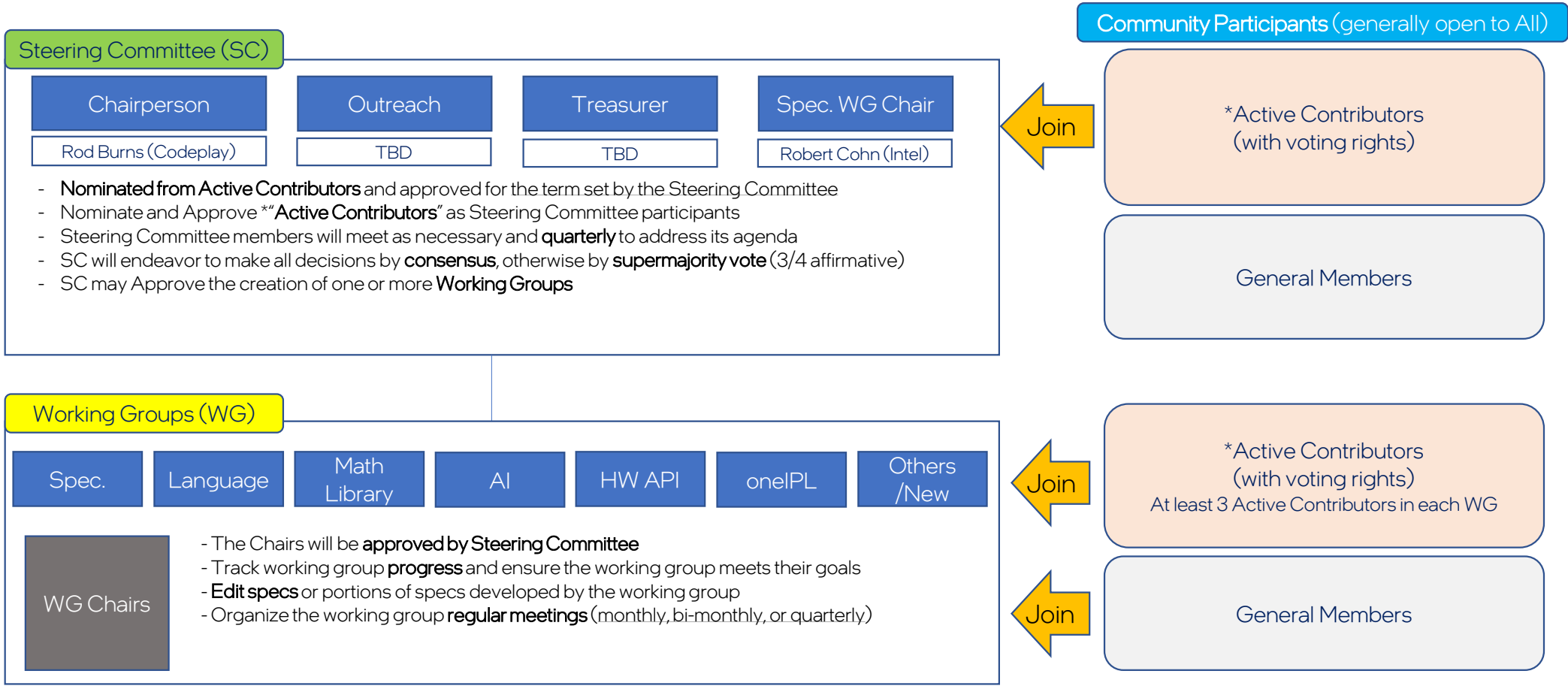
Intel Open Source: No Change – Actively taking contributions

Open governance; Community chairperson, Officers, and "Active Contributors" form a Steering Committee

Working Group for defining the spec changes and new directions; open to anyone

Open Discussion Channels

oneAPI community forum



*Attending 2 of the last 3 meetings of a Working Group or designated by Steering Committee

Device APIs for BLAS

Overview

- **Goal**: Allow users to call common BLAS routines inside their DPC++ kernels
  - "Common" might comprise gemm, syrk, trsm, gemv, axpy, dot, nrm2 *(others?)*
- **Major Issues**
  - **Execution scope**: which work-items are involved?
  - **Data storage**: where are matrices/vectors stored?
  - **Performance portability**: how to get reasonable performance across different devices?
  - **C++ standardization**: aligning with P1673 C++ proposal for BLAS-like operations

Execution and Data

- **Execution scope**: which work-items are involved?
  - Depends on the amount of work to do:
    - Single work-item *(SIMD channel)* – best only for tiny matrices (e.g. 2x2, 3x3)
    - Subgroup *(SIMD operation)*
    - Workgroup
    - Global (all work items)
- **Data storage**: where are the matrices/vectors?
  - Linked to the execution scope
  - Subgroup and smaller: want to keep data in **private memory** (registers)
    - Need an abstraction for in-register matrices/vectors
  - Workgroup and smaller can use **local memory** pointers
  - All options can use **global memory** pointers

API Examples: Matrix Multiplication

- Assume we have some encapsulations of:
  - matrices in local/global memory (like mdspan)
  - matrices in registers (like mdarray)
- BLAS-like per-work-item API

```
template <typename T, class TypeA, class TypeB, class TypeC>
void gemm(const TypeA &A, const TypeB &B, TypeC &C, T alpha, T beta);
```

- P1673 API

```
template <class TypeA, class TypeB, class TypeC>
void matrix_product(sequenced_policy P,
                    const TypeA &A, const TypeB &B, TypeC &C);
```

- All device APIs synchronous

Subgroup/Workgroup Cooperative APIs

- BLAS-like cooperative API

```
template <typename Group, /*…*/>
void gemm(Group G, const TypeA &A, /*…*/);
```

  - `Group` may be a `sycl::sub_group` or `sycl::group`
  - Follows pattern of other SYCL cooperative APIs

- P1673 API

```
template <typename Policy, /*…*/>
void matrix_product(Policy P, const TypeA &A, /*…*/);
```

  - `Policy` encodes the scope of the operation
  - Define execution policies for per-subgroup/workgroup operations that wrap the corresponding SYCL objects.
  - This is a bit of an overload on the meaning of an ExecutionPolicy

Global Cooperative APIs

- BLAS-like global cooperative API

```
template </*…*/>
void gemm(sycl::nd_item<…> i, const TypeA &A, /*…*/);
```

  - `nd_item` informs oneMKL of this work-item's position in the workgroup and the total size of the `nd_range`

- P1673 API

```
template </*…*/>
void matrix_product(parallel_nd_range_policy P, const TypeA &A, /*…*/);
```

  - (Proposed) `parallel_nd_range_policy` indicates parallelization over the whole nd_range, and wraps an `nd_item`

- Host-side queries for optimal `nd_range`

```
template </*…*/> sycl::nd_range<3>
gemm_optimal_range(sycl::queue &Q, const TypeA &A, const TypeB &B, const TypeC &C);
```

Data Storage – Non-Owning (in memory)

- **In-memory matrices/vectors**: mdspan
  - Define allowed subset of mdspan inputs
  - **Element type**: standard BLAS types + half/bfloat16/int8
  - **Extents**: 1D/2D
  - **Layouts**: layout_left, layout_right, P1673 layout_blas_general
  - **Accessors**: default accessor; might need sycl::multi_ptr based accessors to distinguish local/global memory spaces

Data Storage – In Register

- Many tricky aspects here:
  - Optimal implementation is heavily dependent on the architecture
  - Want to allow direct mapping onto vector registers
  - Need sub-group joint storage to allow efficient SIMD vectorization
- Ideally could use P1684 mdarray…
  - **Extents**: 1D/2D, fixed size
  - **Containers**: not flexible enough for the points considered above
- Alternative: dedicated in-register matrix/vector types
  - Do not allow pointer/iterator access to data
  - Definition of matrix type may depend on the architecture
  - joint_matrix type encapsulate a matrix owned by a subgroup

LABB (Linear Algebra Building Blocks)

LABB Matrix Objects

- Basic type is a fixed-size owning matrix (intended to be resident in registers)

```
labb::matrix<float, 8, 8, column_major> M, N;      // 8 x 8 float matrices
```

- Matrix objects support full subscripting and slicing:

```
auto _ = labb::all;
M(2,2) *= 2;
M(_,0) = 1;                                  // _ (a.k.a. all) like Matlab :
M(slice<0,3>, 0) = M(slice<4,7>, 1)          // slice<a,b> like Matlab a:b
M(slice<2>(j), 0) = 1;                       // slice<n>(i) like i:i+n-1
N += M(_,0) * M(0,_);
if (M(0,0) > 0) { … }        // 1x1 slices implicitly convert to scalar type
```

- Overloaded operators
  - +, -, +=, -=: elementwise addition/subtraction/negation; scalars are broadcast.
  - *, *=: scalar and matrix multiplication
  - /: scalar division
- Other common operations (transpose, broadcast, reduction, complex arithmetic)

Subgroups and `joint_matrix`

- Unless matrix very small (e.g. 2x2, 4x4), best to vectorize along one dim of matrix
- For most devices, this requires matrix to be shared across work-items in a subgroup
- Introduce `joint_matrix` variant

```
cgh.submit(…, [=](item &i) {
    auto sg = i.get_sub_group();
    joint_matrix<float, 8, 8, column_major> M(sg), N(sg);
    // Use M/N…
}
```

- `joint_matrix` supports all the regular `matrix` operations
- All assignments/loads/stores to a `joint_matrix` are subgroup operations – must be executed by all work items in the subgroup.

```
M(2, 3) = 1;                                  // OK
if (i[0] == 0) M(2, 3) = 1;                   // Not OK, not all WIs execute
if (i[0] == 0) float f = M(2, 3);             // OK, WIs can read individually
```

Wrap-up

Next Steps

- Focuses for next meeting(s):
  - Any topics from oneMKL TAB members?

- If anyone has content that they would like posted on oneAPI.io, please let us know

Resources

- oneAPI Main Page: https://www.oneapi.io/
- Latest release of oneMKL Spec (currently v. 1.1): https://spec.oneapi.com/versions/latest/elements/oneMKL/source/index.html
- GitHub for oneAPI Spec: https://github.com/oneapi-src/oneAPI-spec
- GitHub for oneAPI TAB: https://github.com/oneapi-src/oneAPI-tab

- GitHub for open source oneMKL interfaces (currently BLAS, RNG, and LAPACK domains): https://github.com/oneapi-src/oneMKL