oneAPI Technical Advisory Board Meeting:

Proposals for dynamic selection of execution contexts and devices

9/22/2021

Mike Voss

Rules of the Road

- DO NOT share any confidential information or trade secrets with the group
- DO keep the discussion at a High Level
  - Focus on the specific Agenda topics
  - We are asking for feedback on features for the oneAPI specification (e.g. requirements for functionality and performance)
  - We are NOT asking for feedback on any implementation details
- Please submit any implementation feedback in writing on Github in accordance with the Contribution Guidelines at spec.oneapi.com. This will allow Intel to further upstream your feedback to other standards bodies, including The Khronos Group SYCL* specification.

Notices and Disclaimers

The content of this oneAPI Specification is licensed under the Creative Commons Attribution 4.0 International License . Unless stated otherwise, the sample code examples in this document are released to you under the MIT license.

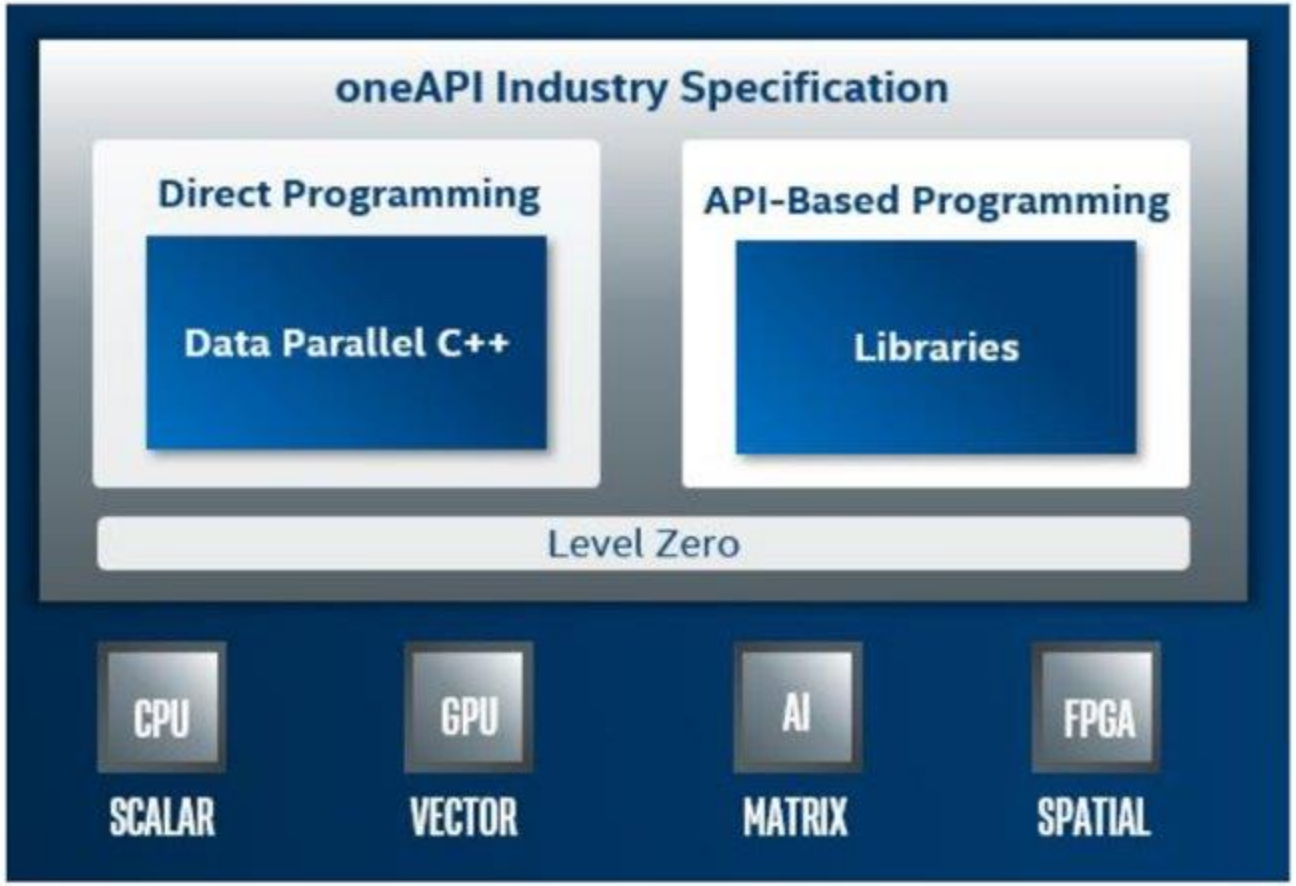
This specification is a continuation of Intel's decades-long history of working with standards groups and industry/academia initiatives such as The Khronos Group*, to create and define specifications in an open and fair process to achieve interoperability and interchangeability. oneAPI is intended to be an open specification and we encourage you to help us make it better. Your feedback is optional, but to enable Intel to incorporate any feedback you may provide to this specification, and to further upstream your feedback to other standards bodies, including The Khronos Group SYCL* specification, please submit your feedback under the terms and conditions below. Any contribution of your feedback to the oneAPI Specification does not prohibit you from also contributing your feedback directly to The Khronos Group or other standard bodies under their respective submission policies.

By opening an issue, providing feedback, or otherwise contributing to the specification, *you agree that Intel will be free to use, disclose, reproduce, modify, license, or otherwise distribute your feedback in its sole discretion without any obligations or restrictions of any kind, including without limitation, intellectual property rights or licensing obligations.* For complete contribution policies and guidelines, see Contribution Guidelines on www.spec.oneapi.com.

Introduction

Assist in Dynamic Selection of XPUs

- Goal: To automatically select an appropriate set of execution resources for applications dynamically.
- Must be pragmatic, not rely on magic, and not assume performance portability.



Two directions considered for the API

- Both provide a dynamic dispatcher that selects an appropriate set of resources for a given piece of work
- They differ in what the dispatcher looks like:

1. A queue-based API
   + Might be approachable for those familiar with SYCL queues
   - Might lead to semantic confusion
2. An execution-policy-based API
   + Might be approachable for those familiar with C++ execution policies
   - Might be less friendly for direct use with SYCL-based apps

Why not a level zero or SYCL API?

- The proposed APIs are more abstract than level zero or SYCL
  - Defer memory allocation, data transfer and glue code to the user-provided functions.
- Provides users with flexibility in choosing scheduling granularity
  - Does not just pick a device for each submitted SYCL kernel
  - A user-supplied function or algorithm receives a handle to the selected resources and can use them to schedule a single kernel, a graph of work, etc.
- Even so, lower-level foundational support may be desirable for efficient implementations.

Queue-based High-Level API using defaults (SYCL)

```
// Create a dynamic selection queue that is NOT tied to a single device
ds::queue dsq;

auto w = dsq.submit([&](sycl::queue q) {
  /* uses the sycl::queue and returns a sycl_event */
});

w.wait();  // wait on specific submission

dsq.wait(); // wait on all tasks submitted to queue
```

Execution Policy-based High-Level API using defaults (SYCL)

```
// Create a dynamic selection execution policy that is NOT tied to a single device
ds::default_policy dsp;

auto w = ds::invoke_async(dsp,
                          [&](sycl::queue q) {
                            /* uses the sycl::queue and returns a sycl_event */
                          }
);

w.wait();  // wait on specific submission

dsp.wait(); // wait on all tasks submitted via this policy
```

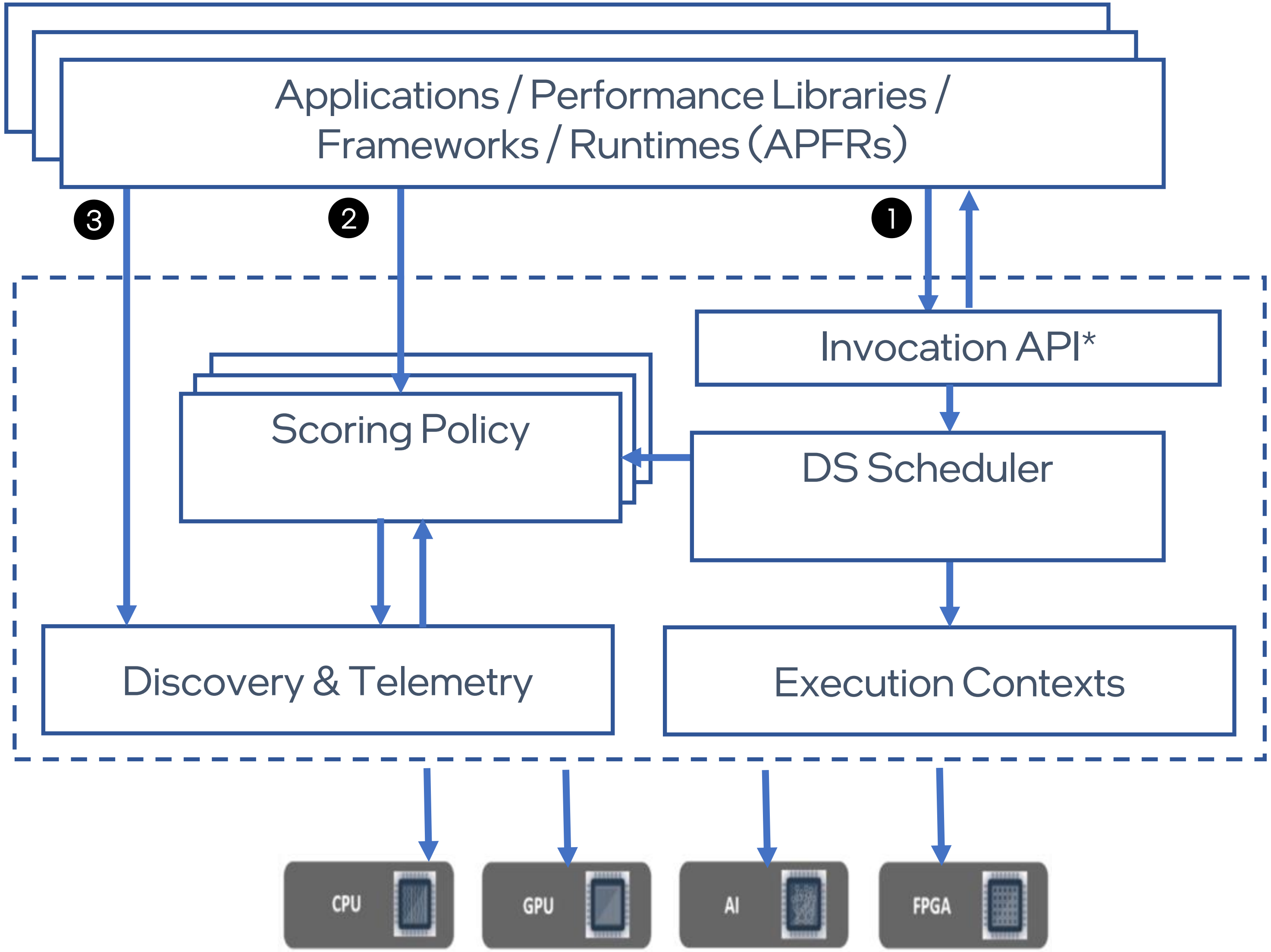
What is the target audience for this?

- Applications that do not get best performance from a static device selection
- Applications that benefit from distributing work to more than 1 device
  - The devices might be of the same type (e.g., multiple GPUs)
  - Or the devices might be of different types (CPU and GPU)

- Some possible use cases deserve caution
  - New developers
  - Developers that do not want to make a choice
  - We can only follow a selection policy not provide performance portability!

Deeper Dive

Conceptual Flows of Dynamic Selection



* Invocation API may be queue submission or execution-policy + algorithm

High-Level API when using defaults (SYCL)

```
void sycl_example() {
  const int num_items = 1000000;
  DoubleVector a(num_items),b(num_items),
c(num_items);
  initialize(a, b, c);

  ds::queue dsq;
  for (int i = 0; i < 100; ++i) {
    dsq.submit([&](sycl::queue q) {
      return f(q, num_items, a, b, c);
    });
  }
  dsq.wait();
}
```

Possible Defaults:

- Scheduler is a SYCL-aware Scheduler
- Universe of options is the single device returned by default_selector unless SYCL_DEVICE_FILTER is set
- A single queue is selected at each submit
- A default scoring policy

High-Level API when using defaults (SYCL)

```
void sycl_example() {
  const int num_items = 1000000;
  DoubleVector a(num_items),b(num_items),
c(num_items);
  initialize(a, b, c);

  ds::queue dsq;
  for (int i = 0; i < 100; ++i) {
    dsq.submit([&](sycl::queue q) {
      return f(q, num_items, a, b, c);
    });
  }
  dsq.wait();
}
```

```
// User function for submission
// Addresses lack of SYCL graph
sycl::event f(sycl::queue q, int num_items,
       DoubleVector& a, DoubleVector& b, DoubleVector& c) {

  sycl::buffer<double> a_buf(a), b_buf(b), c_buf(c);
  q.submit([&](sycl::handler &h) {
    sycl::accessor a_(a_buf, h, sycl::read_only);
    sycl::accessor b_(b_buf, h, sycl::read_only);
    sycl::accessor c_(c_buf, h, sycl::write_only);
    h.parallel_for(num_items, [=](auto j) {
      c_[j] = a_[j] + b_[j];
    });
  });

  return q.submit([&](sycl::handler &h) {
    sycl::accessor a_(a_buf, h, sycl::read_only);
    sycl::accessor b_(b_buf, h, sycl::read_only);
    sycl::accessor c_(c_buf, h, sycl::write_only);
    h.parallel_for(num_items, [=](auto j) {
      c_[j] = a_[j] + b_[j];
    });
  });
}
```

The selected queue can be used to submit 1 or more kernels

Waiting on specific submissions using defaults (SYCL)

```
void sycl_example() {
  const int num_items = 1000000;
  DoubleVector a(num_items),b(num_items), c(num_items);
  initialize(a, b, c);

  ds::queue dsq;
  for (int i = 0; i < 100; ++i) {
    auto w = dsq.submit([&](sycl::queue q) {
      return f(q, num_items, a, b, c);
    });
    w.wait();
  }
}
```

```
// User function for submission
// Addresses lack of SYCL graph
sycl::event f(sycl::queue q, int num_items,
       DoubleVector& a, DoubleVector& b, DoubleVector& c) {

  sycl::buffer<double> a_buf(a), b_buf(b), c_buf(c);
  q.submit([&](sycl::handler &h) {
    sycl::accessor a_(a_buf, h, sycl::read_only);
    sycl::accessor b_(b_buf, h, sycl::read_only);
    sycl::accessor c_(c_buf, h, sycl::write_only);
    h.parallel_for(num_items, [=](auto j) {
      c_[j] = a_[j] + b_[j];
    });
  });

  return q.submit([&](sycl::handler &h) {
    sycl::accessor a_(a_buf, h, sycl::read_only);
    sycl::accessor b_(b_buf, h, sycl::read_only);
    sycl::accessor c_(c_buf, h, sycl::write_only);
    h.parallel_for(num_items, [=](auto j) {
      c_[j] = a_[j] + b_[j];
    });
  });
}
```

By contract the user function object must return a SYCL event.

High-Level API when using defaults (SYCL)

Queue-based

```
void sycl_example() {
  const int num_items = 1000000;
  DoubleVector a(num_items),b(num_items), c(num_items);
  initialize(a, b, c);

  ds::queue dsq;
  for (int i = 0; i < 100; ++i) {
    auto w = dsq.submit([&](sycl::queue q) {
      return f(q, num_items, a, b, c);
    });
  }
  dsq.wait();
}
```

Execution-policy-based

```
void sycl_example_execution_policy() {
  const int num_items = 1000000;
  DoubleVector a(num_items),b(num_items), c(num_items);
  initialize(a, b, c);

  ds::default_policy dsp;
  for (int i = 0; i < 100; ++i) {
    auto w = ds::invoke_async(dsp, [&](sycl::queue q) {
      return f(q, num_items, a, b, c);
    });
  }
  dsp.wait();
}
```

Using a Non-Default Scoring Policy

Queue-based

```
void sycl_example() {
  const int num_items = 1000000;
  DoubleVector a(num_items),b(num_items), c(num_items);
  initialize(a, b, c);

  sycl::queue gpu_queue(sycl::gpu_selector{});
  sycl::queue cpu_queue(sycl::cpu_selector{});

  ds::queue dsq{ds::round_robin_policy{gpu_queue, cpu_queue}};
  for (int i = 0; i < 100; ++i) {
    auto w = dsq.submit([&](sycl::queue q) {
      return f(q, num_items, a, b, c);
    });
  }
  dsq.wait();
}
```

Execution-policy-based

```
void sycl_example_execution_policy() {
  const int num_items = 1000000;
  DoubleVector a(num_items),b(num_items), c(num_items);
  initialize(a, b, c);

  sycl::queue gpu_queue(sycl::gpu_selector{});
  sycl::queue cpu_queue(sycl::cpu_selector{});

  ds::round_robin_policy dsp{gpu_queue, cpu_queue};
  for (int i = 0; i < 100; ++i) {
    auto w = ds::invoke_async(dsp, [&](sycl::queue q) {
      return f(q, num_items, a, b, c);
    });
  }
  dsp.wait();
}
```

Using a non-default backend

```
using namespace InferenceEngine;

void openvino_example() {
  Core ie;
  CNNNetwork network =
    ie.ReadNetwork("semantic-segmentation-adas-0001.xml");
  ExecutableNetwork cpu_nw = ie.LoadNetwork(network, "CPU");
  ExecutableNetwork gpu_nw = ie.LoadNetwork(network, “GPU");
  // other init code …

  // select a non-default Scoring Policy
  // and non-default Backend engine
  ds::queue dsq{ds:round_robin_policy_t<ov_scheduler>{gpu_nw, cpu_nw}};
  for (int i = 0; i < 100; ++i) {
    dsq.submit([&](ExecutableNetwork nw) {
      return f(nw, outWidth, outHeight);
    }).wait();
  }
}
```

```
// user-function for submission
InferRequest f(ExecutableNetwork executableNetwork,
        int outWidth, int outHeight) {
  InferRequest inferRequest =
    executableNetwork.CreateInferRequest();

  /* … */

  inferRequest.StartAsync();
  return inferRequest;
}
```

ov_scheduler is an example of a custom Scheduler:

- defines execution_context type
- defines the return type
- provides a submit function
- provides a wait function
- supports discovery queries and telemetry

Using a non-default backend

Queue-based

```
using namespace InferenceEngine;

void openvino_example() {
  Core ie;
  CNNNetwork network =
    ie.ReadNetwork("semantic-segmentation-adas-0001.xml");
  ExecutableNetwork cpu_nw = ie.LoadNetwork(network, "CPU");
  ExecutableNetwork gpu_nw = ie.LoadNetwork(network, "GPU");
  // other init code …

  // select a non-default Scoring Policy
  // and non-default Backend engine
  ds::queue dsq{ds:round_robin_policy_t<ov_scheduler>{gpu_nw, cpu_nw}};
  for (int i = 0; i < 100; ++i) {
    dsq.submit([&](ExecutableNetwork nw) {
      return f(nw, outWidth, outHeight);
    }).wait();
  }
}
```

Execution-policy-based

```
using namespace InferenceEngine;

void openvino_example() {
  Core ie;
  CNNNetwork network =
    ie.ReadNetwork("semantic-segmentation-adas-0001.xml");
  ExecutableNetwork cpu_nw = ie.LoadNetwork(network, "CPU");
  ExecutableNetwork gpu_nw = ie.LoadNetwork(network, "GPU");
  // other init code …

  // select a non-default Scoring Policy
  // and non-default Backend engine
  ds:round_robin_policy<ov_scheduler> dsp{gpu_nw, cpu_nw};
  for (int i = 0; i < 100; ++i) {
    ds::invoke_async(dsp, [&](ExecutableNetwork nw) {
      return f(nw, outWidth, outHeight);
    }).wait();
  }
}
```

Using a non-default backend

Queue-based

```
using namespace InferenceEngine;

void openvino_example() {
  Core ie;
  CNNNetwork network =
    ie.ReadNetwork("semantic-segmentation-adas-0001.xml");
  ExecutableNetwork cpu_nw = ie.LoadNetwork(network, "CPU");
  ExecutableNetwork gpu_nw = ie.LoadNetwork(network, “GPU");
  // other init code …

  // select a non-default Scoring Policy
  // and non-default Backend engine
  ds::queue dsq{ds:round_robin_policy_t<ov_scheduler>{gpu_nw, cpu_nw}};
  for (int i = 0; i < 100; ++i) {
    dsq.submit([&](ExecutableNetwork nw) {
      return f(nw, outWidth, outHeight);
    }).wait();
  }
}
```

Execution-policy-based

```
using namespace InferenceEngine;

void openvino_example() {
  Core ie;
  CNNNetwork network =
    ie.ReadNetwork("semantic-segmentation-adas-0001.xml");
  ExecutableNetwork cpu_nw = ie.LoadNetwork(network, "CPU");
  ExecutableNetwork gpu_nw = ie.LoadNetwork(network, “GPU");
  // other init code …

  // select a non-default Scoring Policy
  // and non-default Backend engine
  ds:round_robin_policy<ov_scheduler> dsp{gpu_nw, cpu_nw};
  for (int i = 0; i < 100; ++i) {
    ds::invoke(dsp, [&](ExecutableNetwork nw) {
      return f(nw, outWidth, outHeight);
    });
  }
}
```

Some Possible Scoring Policies

- Static Global Rank
  - Execution contexts are statically ranked
  - No platform state or task characteristics are considered
- Round Robin
  - Cycle through Universe of Options
  - No platform state or task characteristics are considered
- Static Score Per Task
  - Preloaded scores (from offline profiling) for each task
  - No platform state is considered
- Dynamic Load
  - Greedy scheduling to devices based on current load
  - Load is a scheduler-specific metric (could be queue length, measured load, etc.)
  - No task characteristic is considered
- Auto Tuning
  - Try and measure performance on different contexts
  - Settle on best performing context
- Custom Policies

Use Case	How to Specify Use Case
Testing against known config	Static Score Per Task
Always use default device	Static Global Rank with default device first
Bias with Fallback	Static Global Rank with bias device. (Scheduler provides fallback)
Greedy by device load	Dynamic Load
Simple load balancing	Round Robin
Auto-tuning	Auto Tuning

What specification to intercept for High-level DS API?

API spec	Library Description	Pros	Cons
oneDPL	C++ algorithms with execution policies, Tested standard C++ APIs, RNGs	Most general C++ library in oneAPI. Contains features for hetero programming.	Prefers C++ standard-like APIs.
oneTBB	C++ library for task-based, shared memory parallel programming on the host.	More API flexibility as it does not have close connection to an external specification.	Currently, exclusively for host-only threading; no hetero/offload features.
SYCL	API for direct programming of accelerators	High-level DS Queue-based API mirrors SYCL API (queues, submit, wait, etc.)	Cannot/should not support non-SYCL kernels. No support for graphs of kernels (yet).
Level Zero	Low-level, direct to the metal interface for offloading to accelerators	Can be a common target for high-level APIs. High-level DS Queue-based API maps well to L0 API (queues, submit, wait, kernels, …)	Too low-level. Foundational support might be put here though.
New oneAPI component	Create a new component for dynamic device selection	Is not forced to fit into any existing component, muddying the scope of that component.	A huge cost.

Questions for TAB

- How would you use a Dynamic Selection API?
- Are we proposing at the right level of abstraction, or would a lower-level API be better?
- Is oneDPL the right place for this API, or would SYCL or Level Zero be better?
- Which approach to take?
  - A queue-based API
  - An execution-policy-based API
- What kind of information is needed to make dynamic selection effective?
  - Do you find the proposed set of scoring policies useful/applicable for your use cases?
  - Do you think useful policies can be implemented without knowing the content of the user-functor? If not, what information is needed?
- Should we support a user-function per execution context type, so that the matching tuned function is invoked?

Call to Action

- The oneAPI specification enables portable access to XPU devices
- We are proposing to evolve the oneAPI specification
- This will include functionality to support dynamic selection

- We are looking for feedback on the direction