Queues and Device Discovery in SYCL

André Cerqueira

15 November

Funded by Project 10110190 – EUROCC2, with support from FCT/MCTES via national funds (PIDDAC).

Objectives for Today

- **Learn about queues**.
- **Querying Device Information:** Querying specific features like memory, atomic support, etc.
- **Standard and Custom Device Selectors:** Built-in selectors and writing custom ones.

What is a SYCL Queue?

- A **queue** connects the host program to a specific device.
- All device code is submitted to a queue for execution.
- Each queue maps to one device:
  - Cannot manage multiple devices.
  - Cannot distribute work across devices.
- Multiple queues can target the same device.

Creating and Using a SYCL Queue

```
auto Q = queue{my_selector{}};
```

- Create a queue using a device selector.
- Then we can submit work using `parallel_for`, `submit`, or other queue methods.

Using Device Selectors

- Device selectors allow us to target specific types of devices:
    - `default_selector`: Implementation-defined default.
    - `cpu_selector`: Targets a CPU device.
    - `gpu_selector`: Targets a GPU device.
    - `accelerator_selector`: For accelerators like FPGAs.

Key Features of SYCL Queues

- Queues are central to work submission in SYCL.
- **Many-to-One Mapping:**
  - Many queues can target the same device.
- **Flexibility:**
  - Declare as many queues as needed.
  - It makes it easier for the programmer to send work to as many devices as they want.

Why Device Discovery in SYCL?

- SYCL supports heterogeneous computing devices (CPUs, GPUs, FPGAs).
- Device discovery allows us to make informed choices about device usage.
- Querying devices ensures our code is adaptable and performs optimally.

Querying Device Information: `get_info`

- SYCL's `get_info` template lets you retrieve key device information.
- Examples of device parameters:
  - `info::device::global_mem_size` - Global memory size.
  - `info::device::max_compute_units` - Number of compute units.
  - `info::device::name` - Device name.

```
auto name = Q.get_device().get_info<sycl::info::device::name>();
std::cout << "Device Name: " << name << std::endl;
```

Note on info namespace

- The ‘info::‘ namespace is vast.
- We can use it to querie many aspects of SYCL code at runtime using get_info, not just for devices. We can use it also to querie platform, context, queue, event and kernels also offer a get_info method.

Creating Custom Device Selectors

- Custom device selectors provide more control for selecting devices.
- Use custom selectors to filter devices based on specific criteria.
- Implemented by inheriting `device_selector` and overriding its function-call operator.
- The method takes a device object and returns a score for it:
    - The score is an integer value; the highest score gets selected.
    - The runtime calls this method once for each accessible device to rank them by score.
    - Devices are excluded from the ranking if their score is negative.

Example: Custom Device Selector

```
class my_selector : public device_selector {
public:
    int operator()(const device &dev) const override {
        if (dev.is_gpu()) {
            auto vendorName = dev.get_info<info::device::vendor>();
            if (vendorName.find("Intel") != std::string::npos) {
                return 1;  // Prioritize Intel GPUs
            }
        }
        return -1;  // Lower priority for other devices
    }
};

auto Q = queue { my_selector{} };
```

- This selector prioritizes Intel GPUs when available.

Using Aspects for Device Capabilities

- The standard defines the aspect_selector function, which return a selectors based on desired device aspects.
  - `aspect::usm_device_allocations` - Unified Shared Memory support.
  - `aspect::fp16` - Half-precision floating-point support.
  - `aspect::atomic64` - 64-bit atomic operations.
- Aspects help ensure devices meet application requirements.

Example: Using Aspect Selector

- The example selects devices that support both USM and FP16.
- Useful for applications relying on specific hardware features.

```
auto my_selector = aspect_selector(
                                  aspect::usm_device_allocations,
                                  aspect::fp16);
queue Q(my_selector);
```

Best Practices for Portability

- Use device selectors and custom selectors for flexible device management.
- Avoid hardcoding device-specific features; instead, use `get_info` and aspects.

Summary

- **SYCL Queues:** Central to work submission, connecting the host to a specific device.
- **Selectors:**
  - Built-in selectors provide convenience for targeting common devices.
  - Custom selectors allow fine-grained control, enabling prioritization and flexibility.
- **Runtime Queries:** The versatile `get_info` function retrieves key information for devices, platforms, queues, and more.
- **Aspects:** Enable filtering devices based on specific capabilities, ensuring hardware compatibility with application requirements.