

oneAPI

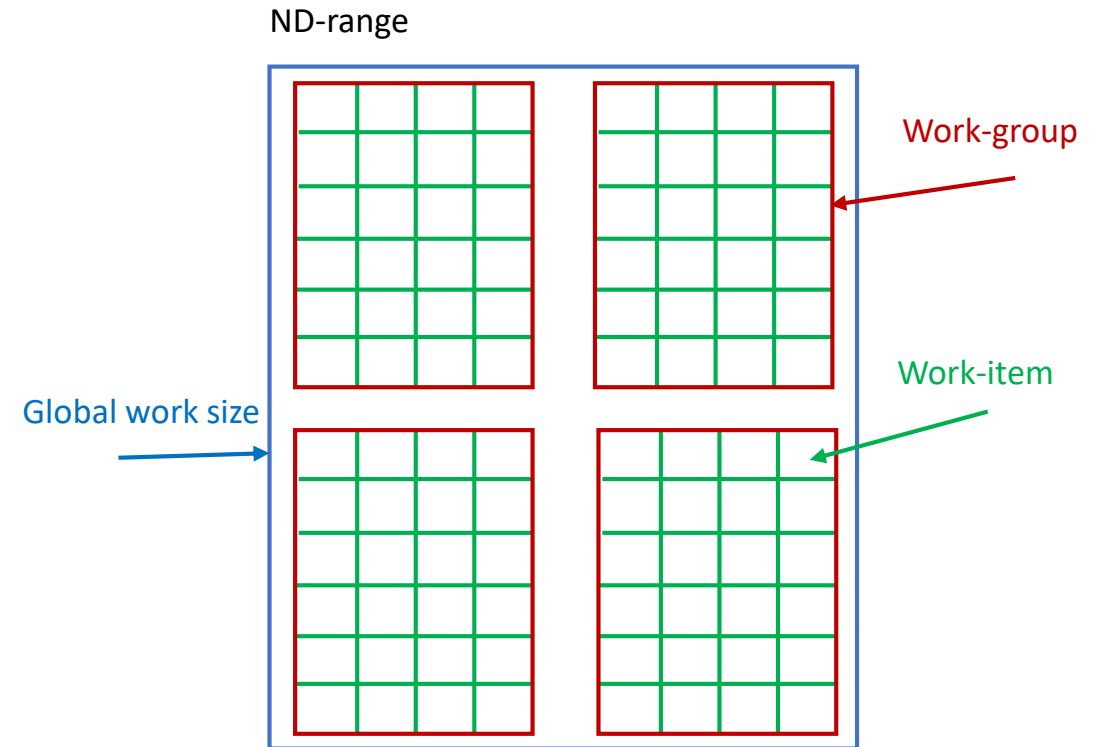
oneAPI Technical Advisory Board Meeting: Local Memory Allocation

08-26-2020

John Pennycook

Re-cap: Local Memory in SYCL Today

- **Work-items** in an ND-range are grouped into 1-, 2- or 3-dimensional **work-groups**.
- Work-items in the same work-group can communicate via work-group **local memory**.
- Local memory may be mapped to dedicated hardware memory where available.



Re-cap: Local Memory in SYCL Today

SYCL only permits local memory allocations via a `local_accessor`, declared outside of the kernel:

```
q.submit([&](handler& cgh) {  
    auto local = local_accessor<uint32_t, 1>(range<1>{L}, cgh);  
    cgh.parallel_for(nd_range<1>{N, L}, [=](nd_item<1> item) {  
        ...  
    })  
});
```

...we'd like to change that, to simplify programming, and to separate local memory allocations from the restrictions of accessors.

Proposal

```
template <typename T, typename Group, typename... Args>
T& group_local_memory(Group g, Args... args);
```

```
template <typename T, typename Group>
T& group_local_memory_for_overwrite(Group g);
```

```
q.parallel_for(nd_range<1>{N, L}, [=](nd_item<1> it) {
    uint32_t& initialized = *group_local_memory<uint32_t>(it.get_group(), 42);
    uint32_t& uninitialized = *group_local_memory_for_overwrite<uint32_t>(it.get_group());
});
```

- Allowed at kernel scope (like OpenCL local) and function scope
- Returns a multi_ptr to an object allocated once for the specified group in local address space
- Object is initialized upon or before first call to group_local_memory (like thread_local)
- Object's lifetime is tied to the group, so behaves like a static variable
- Uniform argument pack forwarded to constructor (like make_unique)

Realistic Example - Arrays

```
q.parallel_for(nd_range<1>{N, L}, [=](nd_item<1> it) {  
  
    // Create uninitialized scratchpad to be filled later  
    auto array = *group_local_memory_for_overwrite<uint32_t[2][2]>(  
        it.get_group());  
  
    // Create array with initial state  
    auto array = *group_local_memory<uint32_t[2][2]>(  
        it.get_group(), {x, y, z, w});  
  
});
```

Potential Issues & Discussion

- Who calls the destructor?
 - We propose to limit to trivially destructible types (for now) to avoid this issue
- What about library-only implementations?
 - Kernel scope is straightforward, if allocations appear before any other code
 - Arbitrary scopes is hard; would need a way to uniquely identify each function call
 - `std::source_location` is insufficient
 - Do we need to work towards something like a `std::unique_location`?
- Is this extension still useful with these limitations?
 - i.e. trivially destructible types, allocations only at kernel scope
- Does this proposal address concerns about `local_accessor`?
Are there alternative designs to consider?

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](https://spec.oneapi.com/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.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.*Other names and brands may be claimed as the property of others.

© Intel Corporation