

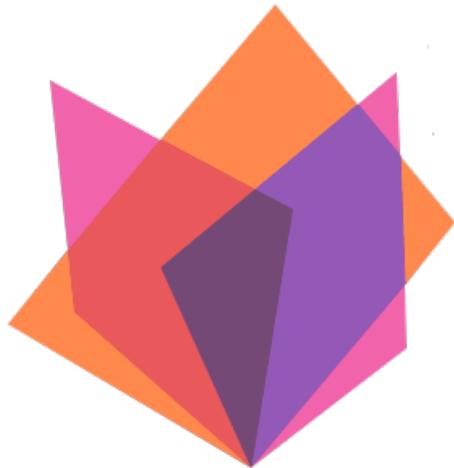
OpenMP[®]

SC'20 Booth talk

OpenMP 5.1: The Interop Construct

Tom Scogland, LLNL

Prepared by LLNL under Contract DE-AC52-07N 27344



OpenMP 5.1: Interoperability and a new construct

- OpenMP 5.1 introduces the **interop** construct
- The **interop** construct provides:
 - A mechanism to request low-level foreign runtime information from OpenMP runtimes
 - The ability to write portable fully async code that uses both OpenMP and native functionality together
 - A long-term extensible place to add interoperability features to OpenMP
- Concrete uses *in 5.1*
 - **targetsync**: Access a synchronization object from the foreign runtime, connect the OpenMP task graph and the foreign runtime task graph
 - **target**: Get native foreign runtime handles to resources associated with a device (context, platform, device information)

Interop: Example get stream

```
async_omp_work(arr);

omp_interop_t o = 0; intptr_t type;

#pragma omp interop init(targetsync: o) depend(inout: arr)

auto type = omp_get_interop_property_int(o, omp_ipr_fr_id);

if (type == omp_ifr_cuda) {

    cudaStream_t s = omp_get_interop_property_ptr(o, omp_ipr_targetsync);

    cublasSetStream(s);

    call_cublas_async_stuff(arr);

} else {

    // handle other cases

}

#pragma omp interop destroy(o) depend(inout: arr)
```

Available properties

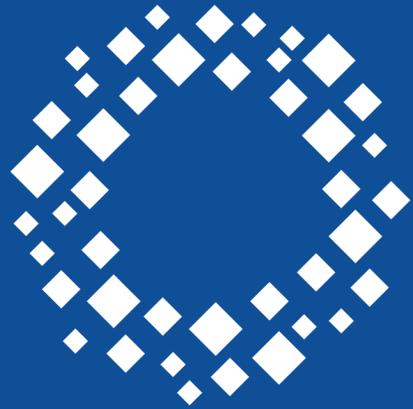
	Available in interop-type	property
Foreign runtime id	all	Numeric id of the foreign runtime system
Foreign runtime name	all	String id
Vendor id	all	Numeric vendor id
Vendor name	all	String vendor id
Device number	all	OpenMP device number
Platform	target	Foreign runtime object that may span devices, OpenCL Platform
Device	target	Foreign runtime object representing a device, cuDevice or OpenCL Device
Device context	target	Context object
Targetsync object	targetsync	Stream/Queue sync object

Fully asynchronous dependencies

```
void foo(double *a) {  
    #pragma omp target teams distribute parallel for nowait depend(inout:a[0])  
    process_a_1();  
    omp_interop_t o = 0;  
    #pragma omp interop init(targetsync: o) depend(inout: arr) nowait  
  
    process_a_with_sycl_without_blocking(a, my_get_queue(o));  
  
    #pragma omp interop destroy(o) depend(inout: arr) nowait  
  
    #pragma omp target teams distribute parallel for nowait depend(inout:a[0])  
    process_a_3();  
}
```

Interop going forward

- Additional interop types in consideration:
 - Thread: access the underlying thread information, say a `posix_thread_t` or `std::thread`
 - Lock: directly access underlying lock implementations
- Other possible properties
 - Memory handle types for systems that don't directly use pointers



CASC

Center for Applied
Scientific Computing



Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

