



oneAPI Distributed Computing

oneAPI DPC++/DPL TAB

David Ozog, Robert Cohn

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

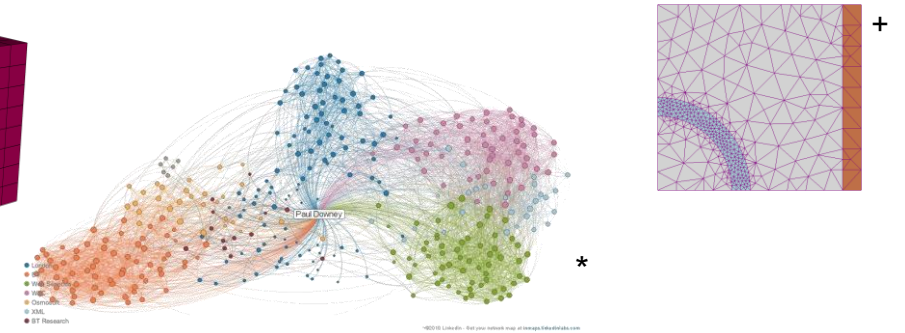
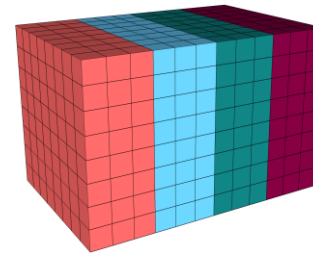
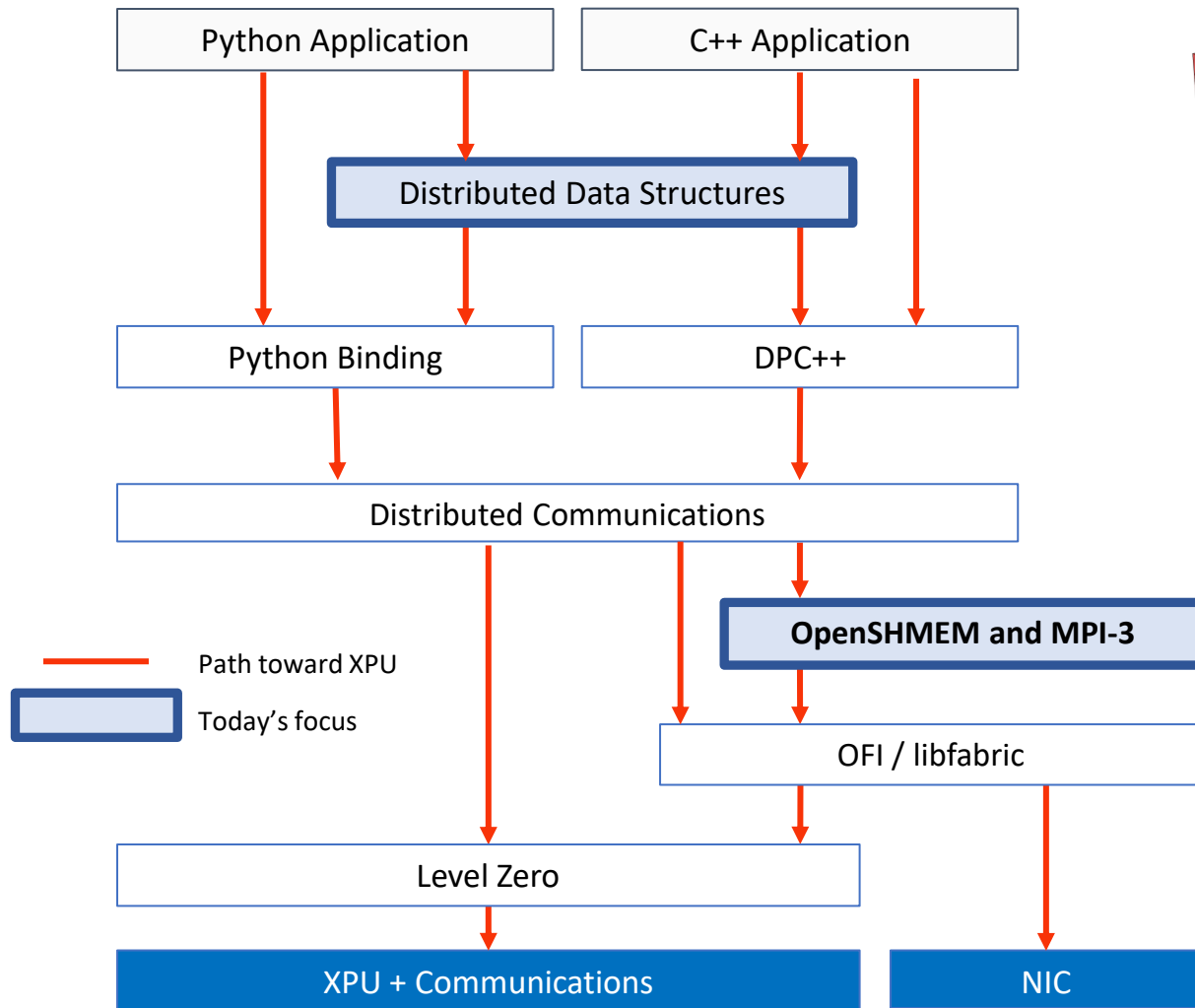
Outline

1. Vision for a better distributed oneAPI architecture
2. PGAS/OpenSHMEM for heterogeneous architectures
3. Distributed Data Structures for heterogeneous architectures

Outline

- 1. Vision for a better distributed oneAPI architecture**
2. PGAS/OpenSHMEM for heterogeneous architectures
3. Distributed Data Structures for heterogeneous architectures

Vision for a Distributed oneAPI Architecture



Why distributed data structures?

- Modern problems require scaling-out
- Writing distributed programs is hard:
Races, domain decomposition, synchronization, etc.
- Limited support for distributed containers, libraries, algorithms
- XPU heterogeneity exacerbates NUMA with disjoint memory models

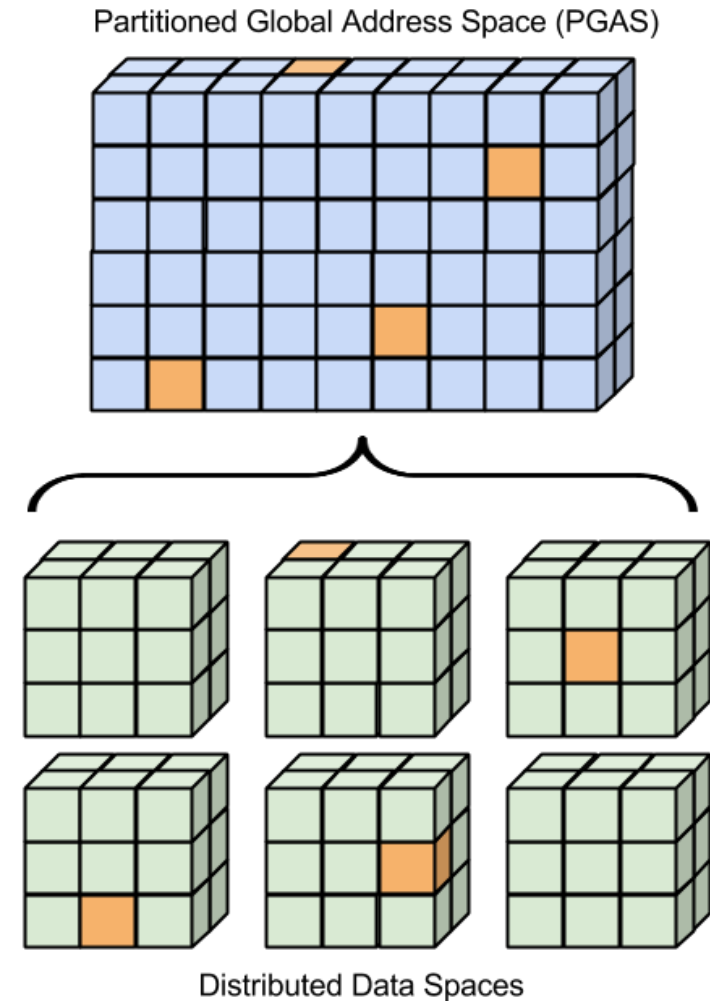
* Attr: Paul Downey; <https://creativecommons.org/licenses/by/2.0/>

+ Attr: Zureks; <https://creativecommons.org/licenses/by-sa/3.0/deed.en>

OpenSHMEM / PGAS Overview

What is OpenSHMEM?

- Open standard for the SHMEM programming model:
 - `shmem_put`, `shmem_get`, etc.
- Partitioned Global Address Space (PGAS) memory model:
 - distributed memory w/ process ID.
- Single Process Multiple Data (SPMD) execution model:
 - All processing elements (PEs) execute the same program



Basic OpenSHMEM (PGAS) Operations

Initialization

- `shmem_init()` sets up *symmetric* data regions

Memory Management

- “Symmetric” memory is remotely accessible
- `shmem_{malloc|calloc|realloc}`

Remote Memory Access (RMA):

- Puts / Gets / Put-with-signal 

Atomic Memory Operations (AMO):

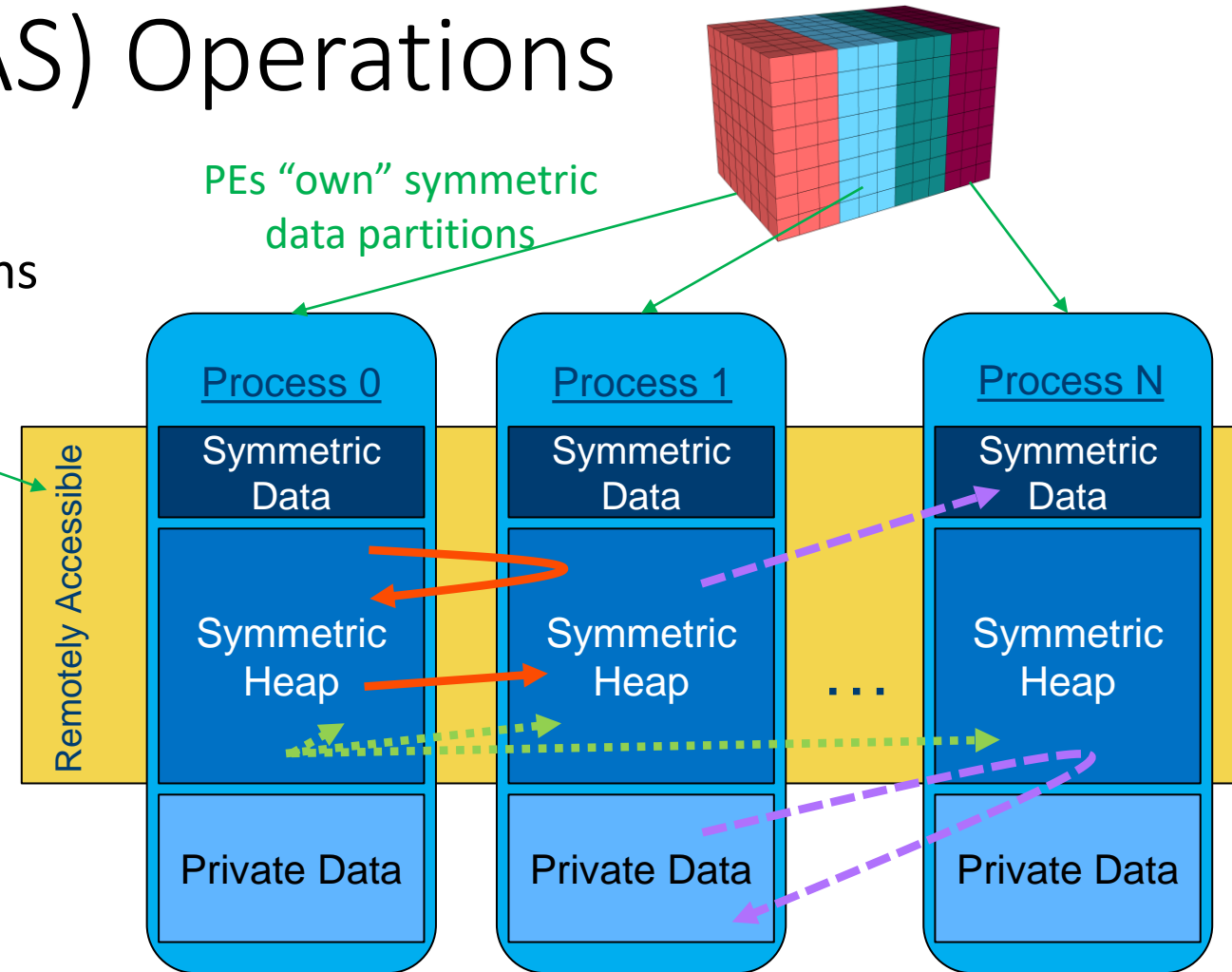
- increment, compare-swap, fetch, bitwise, etc.

Collective Operations:

- Barrier, Broadcast, reduction, all-to-all, etc.

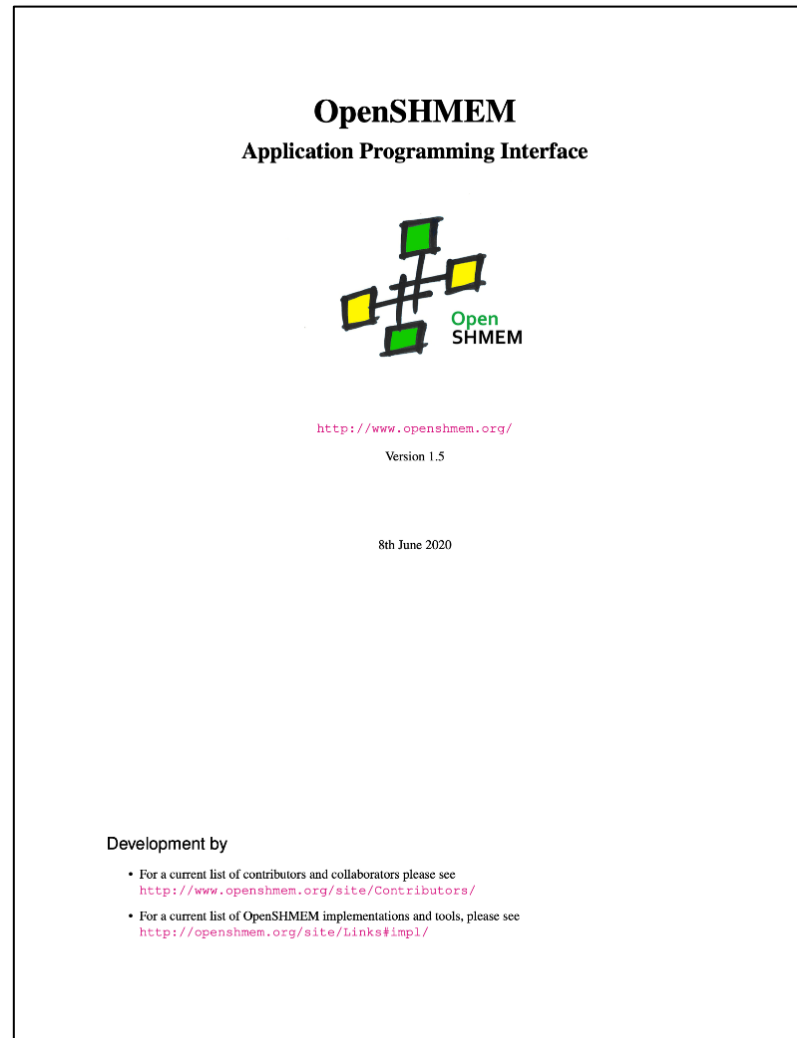
Memory Ordering:

- OpenSHMEM ops are *unordered* – user needs Fence to order, Quiet to complete.



The OpenSHMEM Specification (now v1.5)

- Very active vendor, government, and academic community
- Driven to support new HPC fabric technologies
- Notable new features:
 - Thread safety
 - Communication contexts
 - Teams
- Upcoming
 - Memory Spaces
 - Formal memory model
 - Bundling / queues
 - GPU integration?



*names and images may be claimed as the property of others

Outline

1. Vision for a better distributed oneAPI architecture
- 2. PGAS/OpenSHMEM for heterogeneous architectures**
3. Distributed Data Structures for heterogeneous architectures

OpenSHMEM Integration with oneAPI?

Why it makes sense:

- A mature OpenSHMEM specification (host-side) is already in place.
- Provides user-friendly support for 1-sided distributed memory programming.
- OpenSHMEM interfaces are congruous with SYCL/Level-Zero Shared Memory (USM).
- GPU-initiated SHMEM is possible (e.g., see ROC_SHMEM* and NVSHMEM*).

What are the challenges?

- Resource sharing with MPI (over OFI/libfabric) is non-trivial
 - Possible solution: OpenSHMEM over MPI exists and performs well (e.g., “OSHMPI” at Argonne)
- Key memory and execution model differences for GPU-initiated SHMEM
 - Possible solution: Support a subset of OpenSHMEM with extensions inside SYCL device kernels

OpenSHMEM Integration with oneAPI?

Why it makes sense:

- A mature OpenSHMEM specification (host-side) is already in place.
- Provides user-friendly support for 1-sided distributed memory programming.
- OpenSHMEM interfaces are congruous with SYCL/Level-Zero Shared Memory (USM).
- GPU-initiated SHMEM is possible (e.g., see ROC_SHMEM* and NVSHMEM*).

What are the challenges?

- Resource sharing with MPI (over OFI/libfabric) is non-trivial
 - Possible solution: OpenSHMEM over MPI exists and performs well (e.g., “OSHMPI” at Argonne)
- Key memory and execution model differences for GPU-initiated SHMEM
 - Possible solution: Support a subset of OpenSHMEM with extensions inside SYCL device kernels

Examples: Init on device?
Create Team?
(no...)



Examples: Put per *workgroup*?
Collectives on *queues*?



OpenSHMEM on Device: Specification Challenges

Hypothetical SYCL kernel w/ OpenSHMEM:

```
int *src = (int *) shmem_malloc(array_size * sizeof(int));
int *dst = (int *) shmem_malloc(array_size * sizeof(int));

q.parallel_for( nd_range<1>{N, N}, [=]( nd_item<1>idx ) ) {
    /* Do some work on src buffer */
    do_work(src, idx, chunk_size);

    /* Pass some data to a neighboring device (in a ring fashion) */
    ① shmem_putmem_nbi(&dst_put[idx*chunk], &src[idx*chunk_sz], sizeof(int), (shmem_my_pe() + 1) % shmem_n_pes());
    shmem_quiet();                                ③
});                                              ②
```

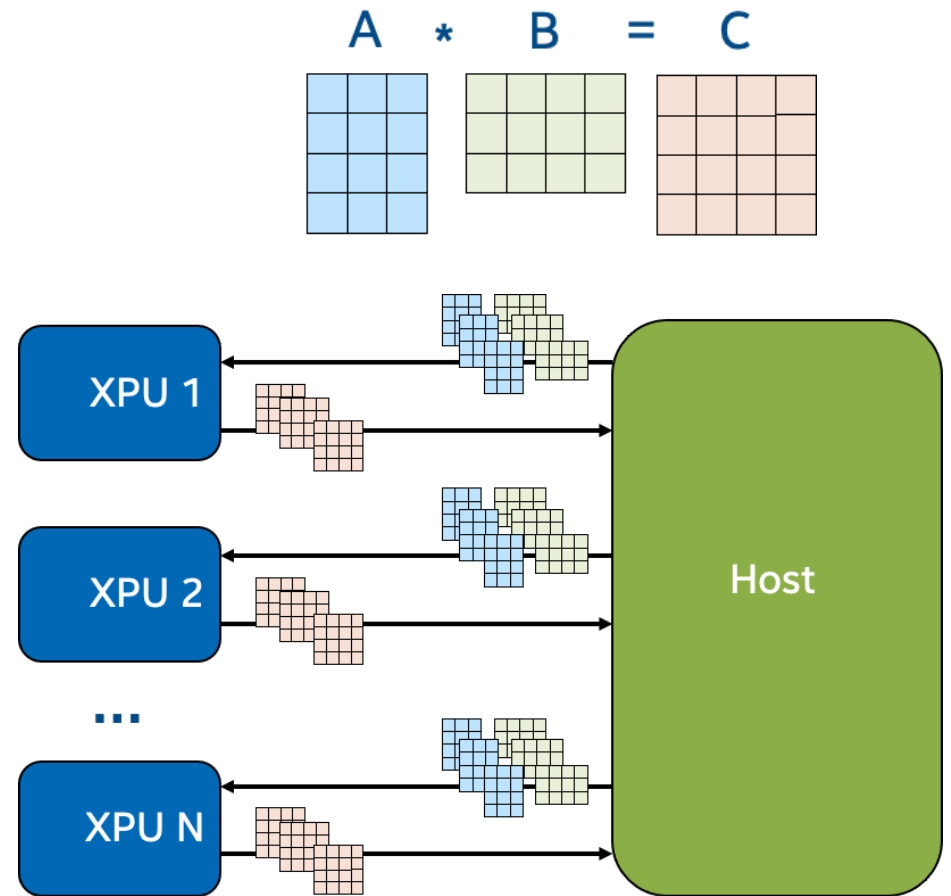
- ① Does `shmem_put_nbi()` (and other RMA) behave the same on the device as on the host?
- ② Does this `shmem_quiet()` complete pending operations within kernel, or also on the host? Across all workitems? PEs?!
- ③ All workitems (threads) doing RDMA simultaneously will likely perform poorly.
Current solution varies across vendors : `vendorA_rma_block()`, `vendorB_rma_ndrange()`, `vendorC_rma_workgroup()`
`vendorA_rma_warp()`, `vendorB_rma_wave()`, `vendorC_rma_subgroup()`

Heterogeneous PGAS

Consider something like matrix multiplication:

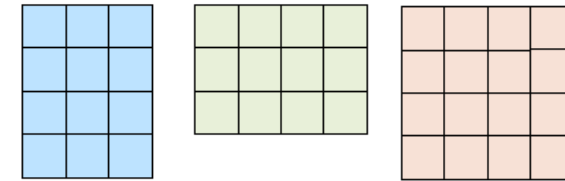
Old Model:

- Host manages everything: buffers, copies, overlap, etc.
- No communication between devices unless routed through host.
- Limits problem size, productivity, and simplicity.



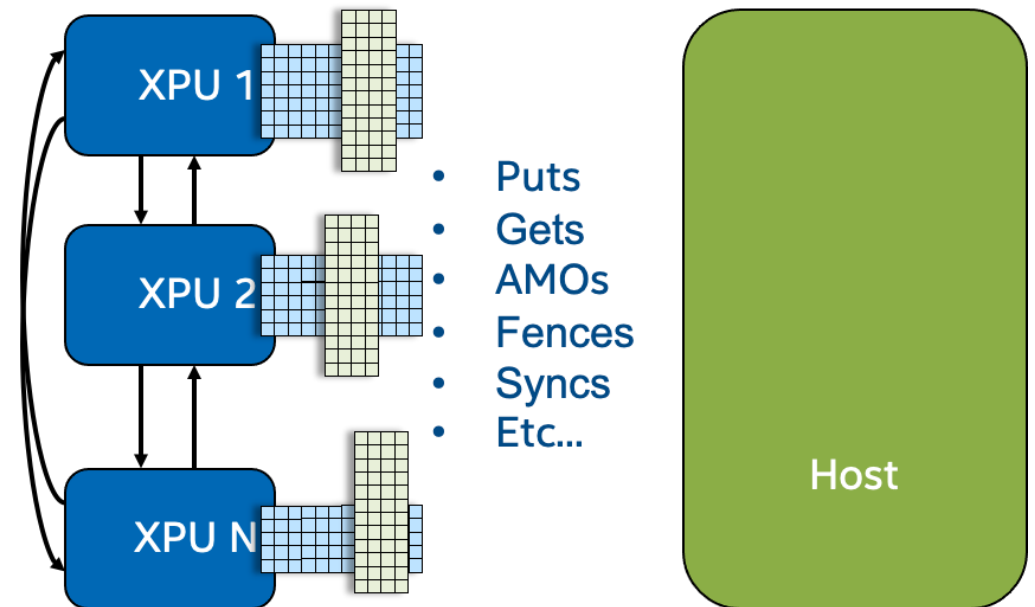
Heterogeneous PGAS

Consider something like matrix multiplication:

$$A * B = C$$


New Model:

- XPU-initiated communication, less host involvement
- Simpler programming, more flexibility
- Performance advantage: fewer kernel boundaries



Outline

1. Vision for a better distributed oneAPI architecture
2. PGAS/OpenSHMEM for heterogeneous architectures
- 3. Distributed Data Structures for heterogeneous architectures**

oneAPI support for Partitioned Memory

- Examples
 - Multiple GPU's in a node with single NUMA memory
 - Multiple GPU's in a node, each with their own memory
 - Inter-GPU communication via host
 - Inter-GPU communication via direct link
 - Multiple hosts connected by network card
- Challenges
 - Routing communication over the most efficient transport
 - Managing remote pointers
 - GPU run to completion not well suited for 2 sided communication
 - 2 sided communication combines communication, buffering, and synchronization
 - 1 sided requires explicit buffering and global barriers

Approach

- Defining library of data structures for partitioned memory
 - matrix, queue, hash, stencil
- Operations support local & global addressing
 - Global: Operate on remote data as if it were local
 - Local: Native pointer access
- Operations callable from SYCL kernel, openMP offload

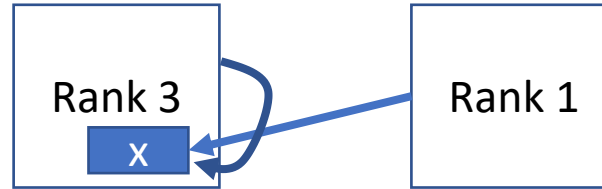
Division of Labor

- Programmer: **Distribution & Parallelization**
 - Defines parallel decomposition (explicit SYCL, openMP, MPI, SHMEM)
 - Defines data distribution (block/cyclic, hosted)
 - Aware of which operations trigger communication
 - Responsible for placement of barriers/flush
 - Uses MPI/SHMEM for global data that is not managed by DDS
- DDS: **Bookeeping**
 - Memory allocation
 - Mechanics of communication (no direct MPI/SHMEM calls)
 - Mechanics of atomic operations (hashmap update)
 - Choosing best transport (PCIe, NIC, ...)
 - Index calculation for partitioned data

API

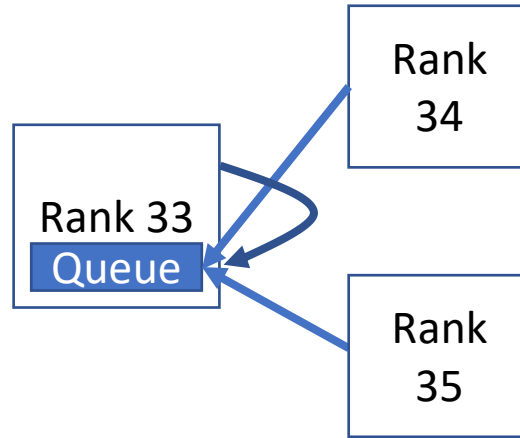
- Header-only library of C++ classes
- Leveraging Berkeley Container Library (BCL) & other research projects
- Support for SYCL
 - USM memory allocation
 - Operations that can be called from kernels
- Support for algorithms well suited to GPU programming models

Global Pointer



```
BCL::GlobalPtr<float> ptr;  
if (BCL::rank() == 0)  
    ptr = BCL::alloc<float>(10);  
ptr = BCL::broadcast(ptr, 3);  
ptr[BCL::rank()] = BCL::rank();
```

Hosted Queue



Global Addressing

```
BCL::Queue<int> bq(33, 100);  
BCL::barrier();  
bq.push(BCL::rank());
```

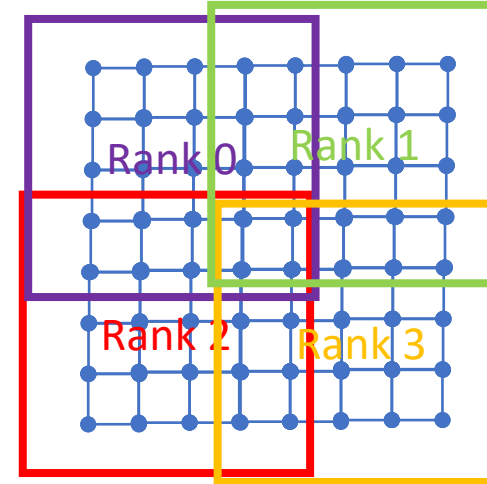
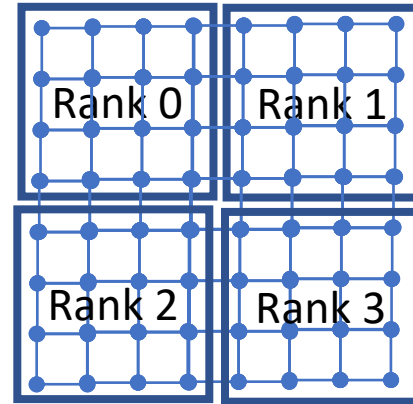
Consistency

```
bq.flush();  
BCL::barrier();
```

Local Addressing

```
if (BCL::rank() == 33) {  
    int * data = bq.begin().local();  
    for (int i = 0; i < bq.size(); i++)  
        sum += data[i]  
}
```

Distributed Matrix



Global Addressing

```
BCL::DMatrix<float> A(queue, {num_rows, num_cols},
                      BCL::BlockSquare());

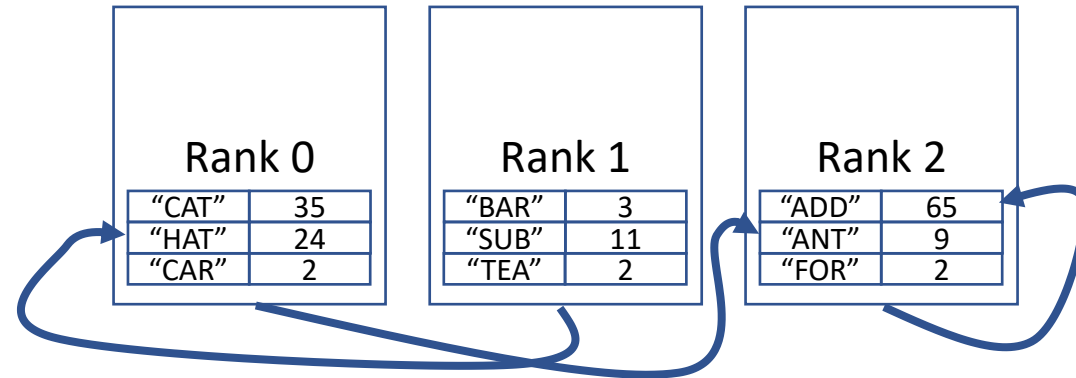
BCL::barrier();

if (BCL::rank() == 0) {
    for (int i = 0; i < num_rows; i++) {
        for (int j = 0; j < num_cols; j++)
            A(i,j) = 0;
    }
}
```

Local Addressing

```
for (int gi = 0; gi < A.grid_shape()[0]; gi++) {
    for (int gj = 0; gj < A.grid_shape()[1]; gj++) {
        if (A.tile_ptr(gi, gj).is_local()) {
            auto a = A.tile_ptr(gi, gj).local();
            for (int i = 0; i < Ap->tile_size(); i++)
                a[i] = 1.0;
        }
    }
}
```

Distributed Hash



```
BCL::HashMap<std::string, int> map(1000);  
BCL::barrier();  
map[std::to_string(BCL::rank())] = BCL::rank();
```


SYCL-related Issues

- Avoiding virtual functions and exceptions in methods
- Sharing data structure metadata (Queue/Hashmap) between host and multiple devices
 - Host USM has desired functionality, but concerns about performance
 - Kernel arguments/device copyable
- GPU-initiated communication and guaranteed forward progress
- Work-item granularity for memory transfers, work-group/kernel granularity for NIC transfers

Feedback

- Appropriate data structures/applications
- What are the difficult problems that a library can help
- What is best left for the user