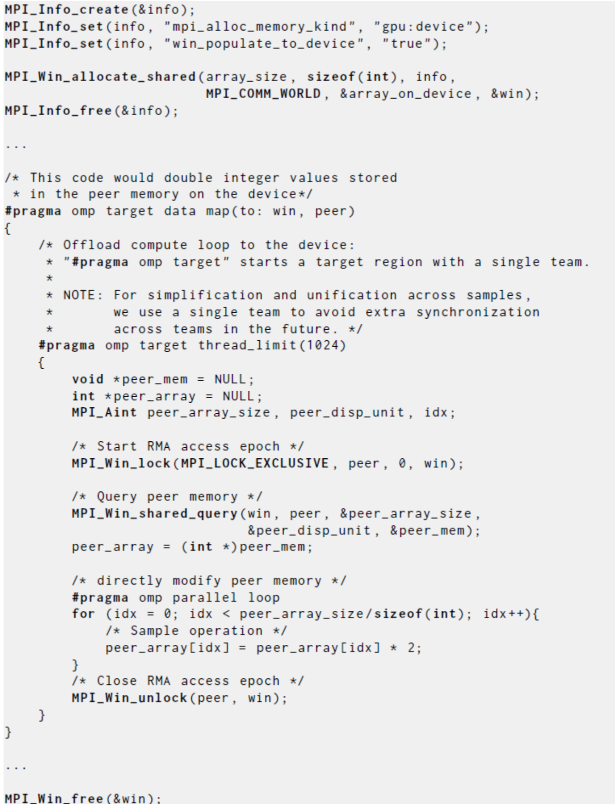
Device-initiated one-sided communications

RMA WG

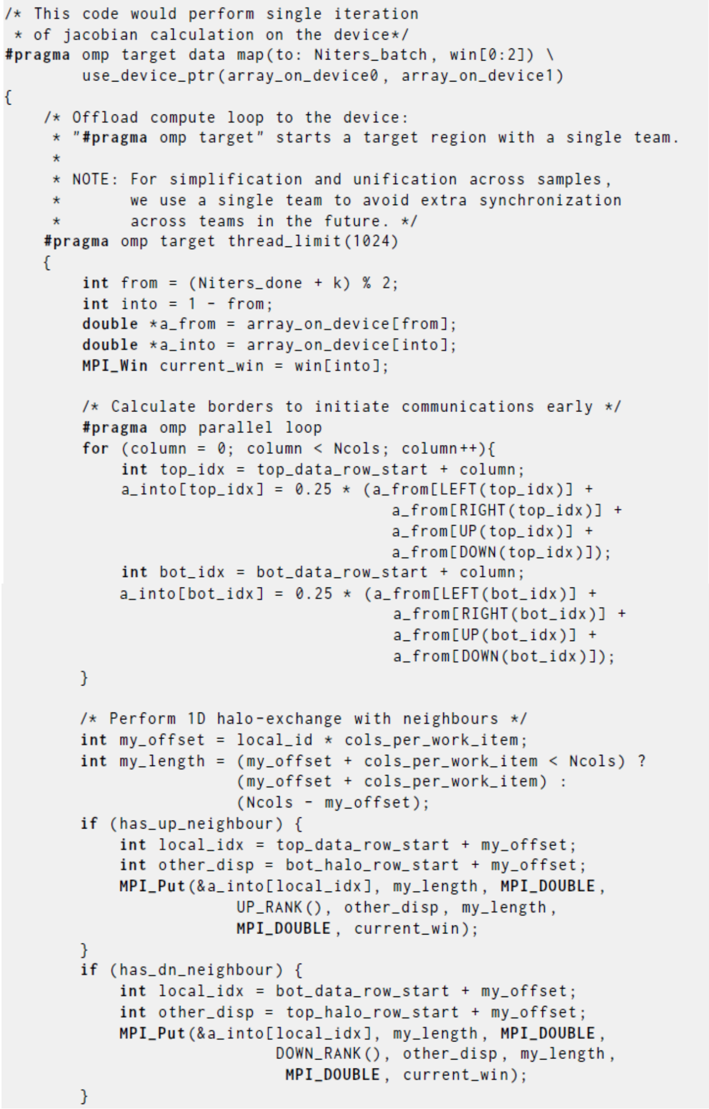
Uniform CPU/Accelerator interface

Goals:

- To provide uniform and consistent interface for accelerator-centric applications.
- Provide ability to drive communications from the offloaded code.



```
MPI_Info_create(&info);
MPI_Info_set(info, "mpi_alloc_memory_kind", "gpu:device");
MPI_Info_set(info, "win_populate_to_device", "true");

MPI_Win_allocate_shared(array_size, sizeof(int), info,
                        MPI_COMM_WORLD, &array_on_device, &win);
MPI_Info_free(&info);

...

/* This code would double integer values stored
 * in the peer memory on the device*/
#pragma omp target data map(to: win, peer)
{
    /* Offload compute loop to the device:
     * "#pragma omp target" starts a target region with a single team.
     *
     * NOTE: For simplification and unification across samples,
     *       we use a single team to avoid extra synchronization
     *       across teams in the future. */
    #pragma omp target thread_limit(1024)
    {
        void *peer_mem = NULL;
        int *peer_array = NULL;
        MPI_Aint peer_array_size, peer_disp_unit, idx;

        /* Start RMA access epoch */
        MPI_Win_lock(MPI_LOCK_EXCLUSIVE, peer, 0, win);

        /* Query peer memory */
        MPI_Win_shared_query(win, peer, &peer_array_size,
                             &peer_disp_unit, &peer_mem);
        peer_array = (int *)peer_mem;

        /* directly modify peer memory */
        #pragma omp parallel loop
        for (idx = 0; idx < peer_array_size/sizeof(int); idx++){
            /* Sample operation */
            peer_array[idx] = peer_array[idx] * 2;
        }
        /* Close RMA access epoch */
        MPI_Win_unlock(peer, win);
    }
}

...

MPI_Win_free(&win);
```



```
/* This code would perform single iteration
 * of jacobian calculation on the device*/
#pragma omp target data map(to: Niters_batch, win[0:2]) \
        use_device_ptr(array_on_device0, array_on_device1)
{
    /* Offload compute loop to the device:
     * "#pragma omp target" starts a target region with a single team.
     *
     * NOTE: For simplification and unification across samples,
     *       we use a single team to avoid extra synchronization
     *       across teams in the future. */
    #pragma omp target thread_limit(1024)
    {
        int from = (Niters_done + k) % 2;
        int into = 1 - from;
        double *a_from = array_on_device[from];
        double *a_into = array_on_device[into];
        MPI_Win current_win = win[into];

        /* Calculate borders to initiate communications early */
        #pragma omp parallel loop
        for (column = 0; column < Ncols; column++){
            int top_idx = top_data_row_start + column;
            a_into[top_idx] = 0.25 * (a_from[LEFT(top_idx)] +
                                      a_from[RIGHT(top_idx)] +
                                      a_from[UP(top_idx)] +
                                      a_from[DOWN(top_idx)]);
            int bot_idx = bot_data_row_start + column;
            a_into[bot_idx] = 0.25 * (a_from[LEFT(bot_idx)] +
                                      a_from[RIGHT(bot_idx)] +
                                      a_from[UP(bot_idx)] +
                                      a_from[DOWN(bot_idx)]);
        }

        /* Perform 1D halo-exchange with neighbours */
        int my_offset = local_id * cols_per_work_item;
        int my_length = (my_offset + cols_per_work_item < Ncols) ?
                        (my_offset + cols_per_work_item) :
                        (Ncols - my_offset);
        if (has_up_neighbour) {
            int local_idx = top_data_row_start + my_offset;
            int other_disp = bot_halo_row_start + my_offset;
            MPI_Put(&a_into[local_idx], my_length, MPI_DOUBLE,
                        UP_RANK(), other_disp, my_length,
                        MPI_DOUBLE, current_win);
        }
        if (has_dn_neighbour) {
            int local_idx = bot_data_row_start + my_offset;
            int other_disp = top_halo_row_start + my_offset;
            MPI_Put(&a_into[local_idx], my_length, MPI_DOUBLE,
                            DOWN_RANK(), other_disp, my_length,
                             MPI_DOUBLE, current_win);
        }
```

https://github.com/mpiwg-rma/mpi-standard/pull/13

Proposed changes

Keep original signatures intact and compatible with existing standard
Extend existing MPI primitive semantics only if necessary.
Introduce MPI_Win_populate_to_device for host/device interoperability.

Communication:

- MPI_Put
- MPI_Get
- MPI_Accumulate
- MPI_Get_accumulate
- MPI_Fetch_and_op
- MPI_Compare_and_swap

Aux:

- MPI_Win_get_attr
- MPI_Win_shared_query

Syncronization:

- MPI_Win_(un)lock **(stricter requirements)**
- MPI_Win_(un)lock_all **(stricter requirements)**
- MPI_Win_flush
- MPI_Win_flush_local
- MPI_Win_flush_all
- MPI_Win_flush_local_all
- MPI_Win_sync

Host-device interoperability:

- MPI_Win_populate_to_device **(new)**

Host/accelerator interoperability

int **MPI_Win_populate_to_device**(MPI_Win win)

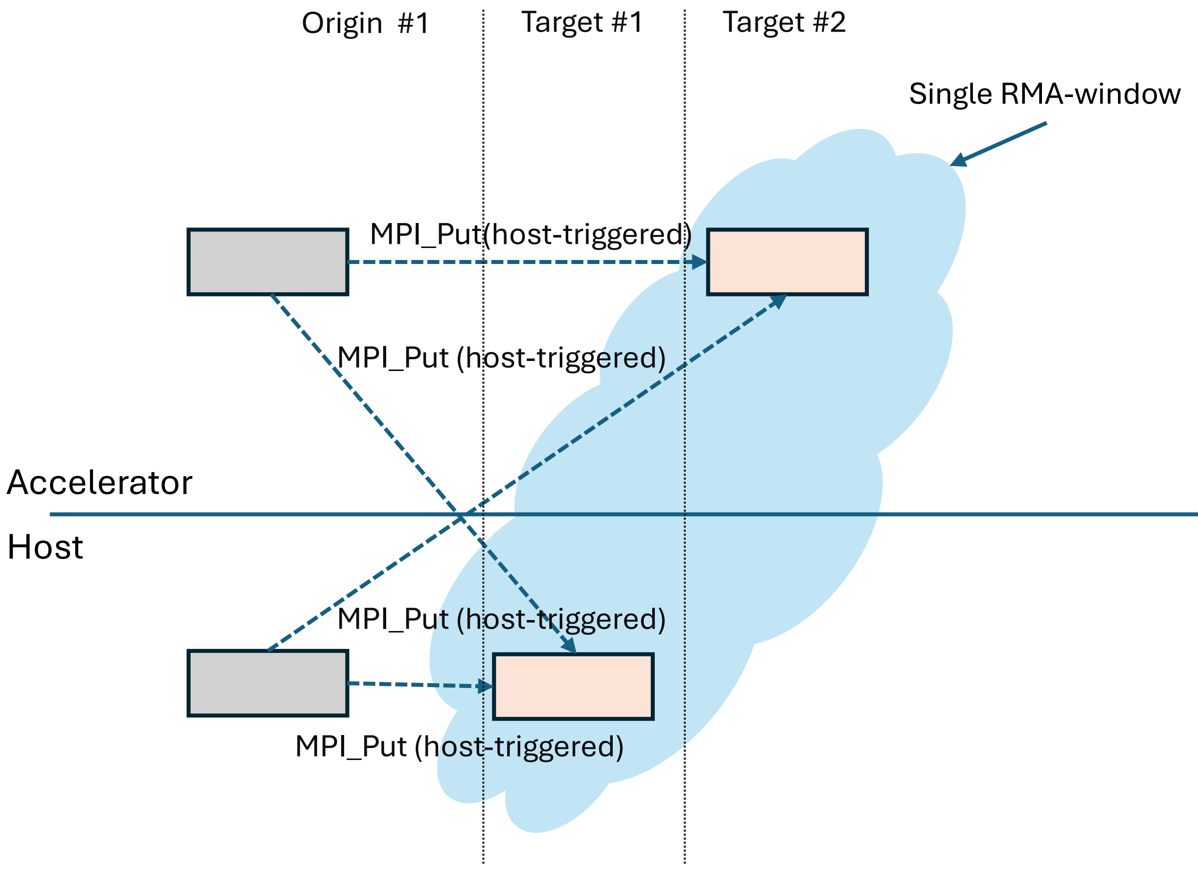
- Local, non-collective operation
- Makes window handle available for MPI calls on the device.
- May populate some internal structures to the GPU memory.
- Should be called before any GPU-initiated operation involving the window.
- No pair “un-populate” call – window reside on GPU till MPI_Win_free call.
- As an alternative, “win_populate_to_device”:”true” could be passed as a part of info object to window creation function.

New MPI_Win attribute is added to check if window already present on the device:

MPI_Win_get_attr(win, **MPI_WIN_RESIDING**, &device_residing, &flag);

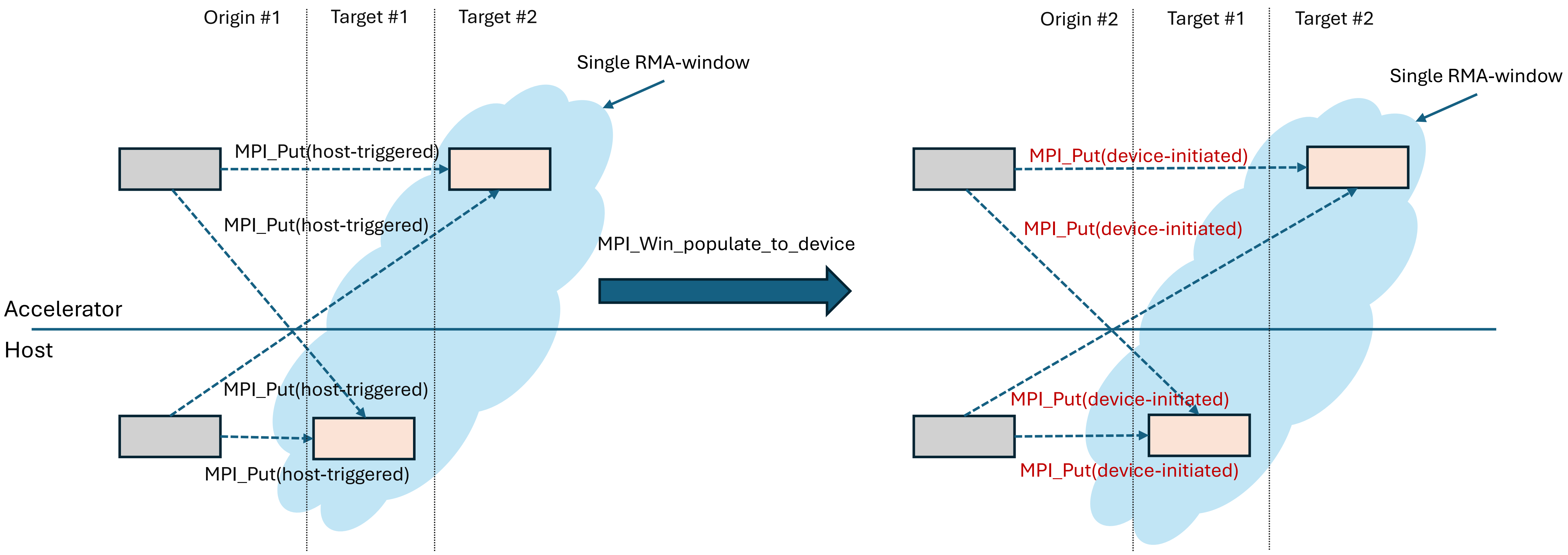
- Value of attribute could be **MPI_WIN_HOST_RESIDING** or **MPI_WIN_DEVICE_RESIDING**.

Target-origin placement



□ - Window memory (target) ▩ - Origin buffer

Target-origin placement



🟧 - Window memory (target) ⬜ - Origin buffer

Detecting accelerators support and DI-comms

Compile time detection:

- MPI_HAVE_[CUDA,ROCM,ZE]_SUPPORT – is GPU supported by MPI runtime.
- MPI_HAVE_[CUDA,ROCM,ZE]_DEVICE_INITIATED – is device-initiated RMA supported.

Runtime detection:

- "is GPU supported by MPI runtime" – could be deduced from supported memory kinds.
- "is device-initiated RMA supported for particular GPU" - could be detected from "device_initiated" field of info object.

MPI_Comm_get_info (MPI_COMM_WORLD , & info);
MPI_Info_get_string (info , ***"mpi_memory_alloc_kinds"*** *, & len , mem_kinds, & flag);*
MPI_Info_get_string (info , ***"device_initiated"*** *, & len , device_initiated, & flag);*

Scope, limitations and current problems

Proposed scope/explicit limitation:

- Passive-target synchronization model only .
- Built-in contig datatypes only.

Current problems:

- ABI compatibility for device code – requires runtime linkage of a device binary code level.
- Multi-vendor GPUs support may introduce standard compliance issues.