MPI-RMA: notification proposals

Alexander Sannikov, Dmitry Durnov(SAT/DSE/PRE)

Problem statement

In passive target mode, notifying the target side that data has been transmitted is currently inefficient.
It requires sending additional message/sync primitive invocation after operation for the notification.

Target scenarios:

- producer-consumer flow
- halo-exchange
- DAG-like models

Requirements:

- Cover all scenarios above with multiple counters/notifications support
- Provide logic for efficient computation and communication overlap
- Allow device-initiated communication (Including ability to get notification within GPU kernel)
- Provide scalable notification infrastructure management and polling

Existing proposals

1) **Synchronize and notify.** Proposed in 2016 by James Dinnan (https://github.com/mpi-forum/mpi-issues/issues/59)
   Cons:
   - Does not solve overlap issue on target rank completely.
   - Single counter significantly limits number of use cases.

2) **Op-and-notify.** Proposed in 2016 by James Dinnan (https://github.com/mpi-forum/mpi-issues/issues/59)
   Cons:
   - Single counter significantly limits number of use cases.

3) **Matched notification.** Proposed in 2015 by Torsten Hoefler (https://spcl.inf.ethz.ch/Publications/.pdf/notified-access-extending-rma-slides.pdf)
   Cons:
   - Brings tag matching logic on target side to get notifications.
     Which makes it extremely inefficient for device-initiated communications, and offloaded code in general.

4) **OpenSHMEM approach.** Standardized in 2018 (https://github.com/openshmem-org/specification/issues/206)
   Cons:
   - Mapping of notification counters is completely on application side, which makes it difficult to manage for applications with offload.

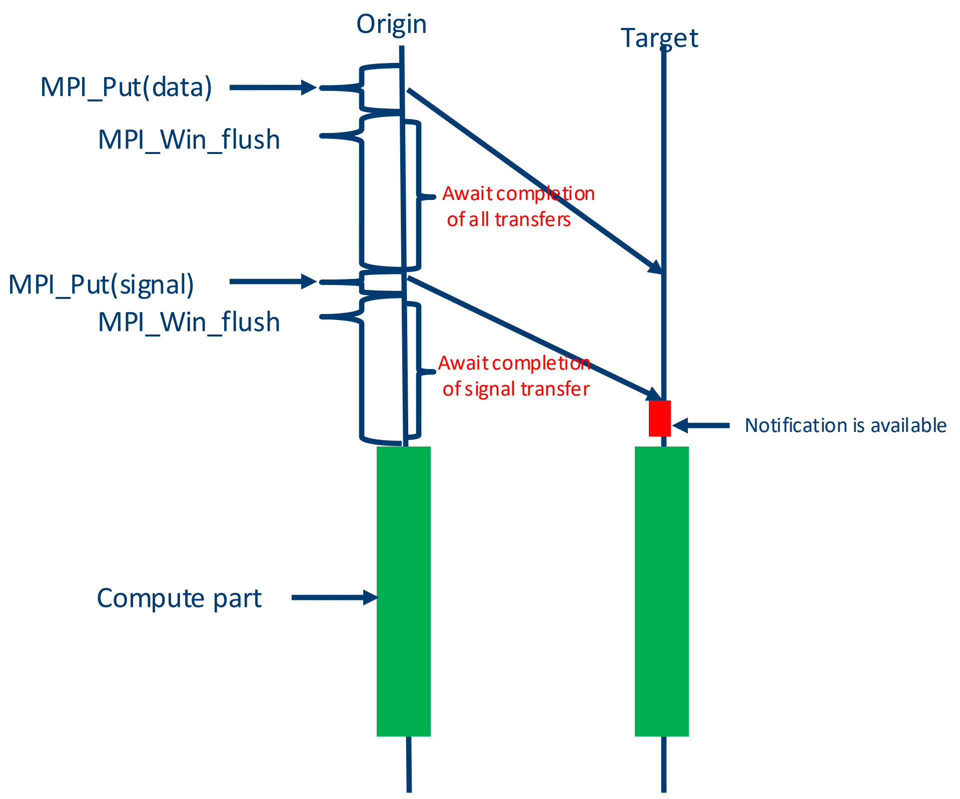
1-D Halo example (Existing implementation)

```
void halo_1D(MPI_Win win, MPI_Win signal_win,
              int buffer[][], int size_x,
              int size_y, int my_rank, int comm_size)
{
    int left_nbh = my_rank - 1;
    int right_nbh = my_rank + 1;
    if (right_nbh == comm_size) right_nbh = -1;
    //Aux setup phase:
    //      No extra actions required

    while(…) {
        //Perform data transfer
        if (left_nbh >=0 ) {
            MPI_Put( … , left_nbh, …, win);

        if (right_nbh >=0 )
            MPI_Put(…, right_nbh, …, win);
        MPI_Win_flush_all(win);
        // Notify neighbours regarding data
        if (left_nbh >=0 ) {
            MPI_Put( … , left_nbh, …, signal_win);

        if (right_nbh >=0 )
            MPI_Put(…, right_nbh, …, signal_win);
        MPI_Win_flush_all(signal_win);
        // Compute part
    }
}
```


Origin
Target
MPI_Put(data)
MPI_Win_flush
Await completion of all transfers
MPI_Put(signal)
MPI_Win_flush
Await completion of signal transfer
Notification is available
Compute part

Currently available approaches have **no way to notify target without explicit target/origin synchronization.**
Usage of MPI_Win_flush, or locks force target to wait till completion of data transfer during corresponding calls.

Proposal: dynamic counters

This approach gives ability to bind arbitrary amount of a notification counters to a window:

int MPI_Win_notify_set_num(MPI_Win win, int notification_num);
int MPI_Win_notify_get_num(MPI_Win win , int *notification_num);

Where notification_num is number of notification slots and could be inequal for different ranks.

Proposed communication primitives allows to update a target notification counter specified by **notification** parameter.

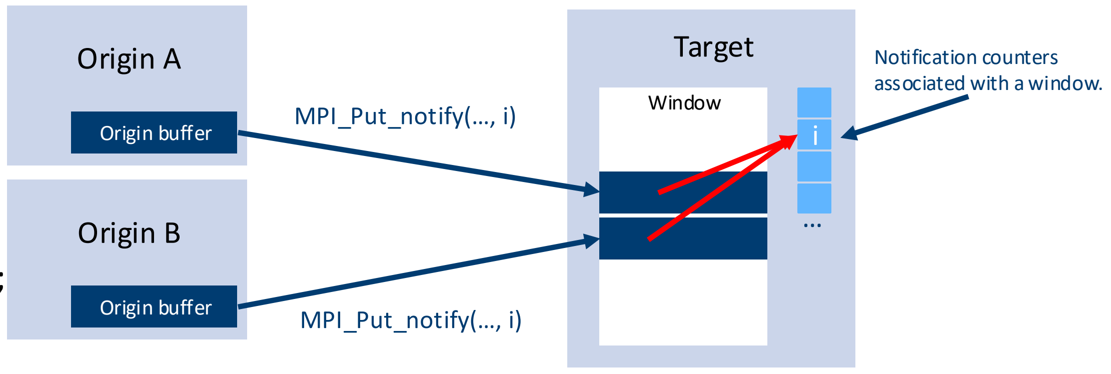
int MPI_Put_notify(…, **int notification_idx,**   MPI_Win win);
int MPI_Get_notify(…, **int notification_idx,** MPI_Win win);

Target could query/modify each individual notification counter associated with a window using accessor functions:

int MPI_Win_notify_get_value(MPI_Win win, **int notification_idx,** MPI_Count *value);
int MPI_Win_notify_set_value(MPI_Win win, **int notification_idx,** MPI_Count value);

In addition, request object may be associated with notification counter. More than single request is allowed:

int MPI_Win_notify_ithreshold(MPI_Win win, int notification_idx,
                                        MPI_Count expected_val, MPI_Request *request);



Cons of similar approach (according to Torsten Hoefler)

- Dataflow applications may require many counters	**<- Not a problem, if counters are managed by MPI.**
- High polling overhead to identify accesses	**<– Request-based polling minimize this problem**
- Does not preserve order (may not be linearizable)	**<- Could be addressed by application by dedicated counters**

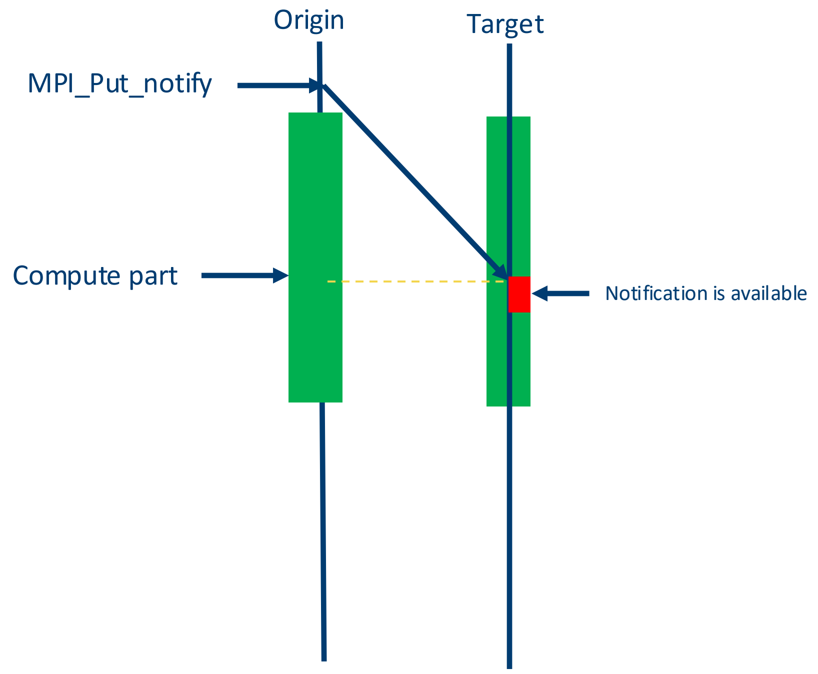
1-D Halo example (dynamic counters)

```
void halo_1D_variant4_IMPI(MPI_Win win, int buffer[][],
                           int size_x, int size_y,
                           int my_rank, int comm_size)
{
    int left_nbh = my_rank - 1;
    int right_nbh = my_rank + 1;
    if (right_nbh == comm_size) right_nbh = -1;
    // Aux setup phase
    // Counters address exchange
    int iteration = 1;
    MPI_Win_notify_set_num(win, 2);

    while(...) {
       MPI_Count c0,c1;
        if (left_nbh >=0 )
            MPI_Put_notify( ..., left_nbh, ..., 1, win);
        if (right_nbh >=0 )
            MPI_Put_notify(..., right_nbh, ..., 0, win);

        // Compute using internal points.

        do {
               MPI_Win_notify_get_value(win, 0, &c0);
               MPI_Win_notify_get_value(win, 1, &c1);
        } while (c0  != iteration  || c1 != iteration);
        iteration++;
        // Compute using border values
    }
    MPI_Win_notify_set_num(win, 0);
}
```



Dynamic counters: counters poll

Poll using counter value accessor.

```
MPI_Count c0;
do {
        MPI_Win_notify_get_value(win, 0, &c0);
} while (c  < expected);
```

This method guarantee uniform and atomic access to notification counters, for both CPU/GPU cases. Also, does not require any requests tracking on application side, and could be optimally mapped into memory polling for device-initiated communications.

Poll using notification request:

```
MPI_Request r;

MPI_Win_get_notify_ithreshold(win, 0, expected, &r);

MPI_Wait(r, MPI_STATUS_IGNORE);
```

Allows to mix notification handling with point-to-point and non-blocking collectives communications. Allows to create multiple threshold for a single counter via multiple requests.

Dynamic counters: notifications management

1) **Single primitive** to transform **any window into one with notifications**:
   Instead of duplication of multiple calls (MPI_Win_create/MPI_Win_create_c/MPI_Win_Create_dynamic/etc.)
   We add a single call which enables notifications support for a window.

2) This approach allows us to rearrange amount and purpose of counters **without window destruction/creation.**

```
MPI_Win_notify_set_num(win, X);
// Application run through one phase of execution.
…
// Application reset and reconfigure counters for another phase
MPI_Win_notify_set_num(win, Y);
```

3) MPI_Win_notify_set_num is collective (has semantic of "close and reopen epoch", like fence or flush).
   Leads to completion of all previously issued communications to keep value of notification counters consistent.
   That means that any communication operation with notification will complete and update corresponding counter during this function.

4) Application is responsible for attaching counters to a window before use, as well as using proper counter indexes during communications.
   If notification index is out of allowed range or notifications are not enabled for a window -
   communication primitive would return MPIX_ERR_INVALID_NOTIFICATION.

intel®

Software