# GROMACS - Feature #2915

Task # 3370 (New): Further improvements to GPU Buffer Ops and Comms

## GPU direct communications

04/02/2019 05:29 PM - Alan Gray

| | |
|---|---|
| **Status:** | In Progress |
| **Priority:** | High |
| **Assignee:** | |
| **Category:** | |
| **Target version:** | |
| **Difficulty:** | uncategorized |

**Description**

Part of this family of developments involves enablement of direct communications between GPU memory spaces. This issue is to discuss common sub-issues related to the development of these new features.

TODO

- ~~Investigate if GPU halo exchange inter-GPU sync in thread MPI case can be made more lightweight with cuStreamWaitValue32 (or similar).~~ (done with MPI exchange of pointers to events, with remote event enqueued to stream).

**Subtasks:**

| | |
|---|---|
| Task # 3082: move launch/synchronization points to clarify task dependencies | **New** |
| Feature # 3087: enable GPU peer to peer access | **Closed** |

**Related issues:**

| | |
|---|---|
| Related to GROMACS - Feature #2891: PME/PP GPU communications | **In Progress** |

### Associated revisions

**Revision 8f42be1e - 08/16/2019 11:29 AM - Alan Gray**

GPU halo exchange

Activate with GMX_GPU_DD_COMMS environment variable.

Class to initialize and apply halo exchange functionality directly on
GPU memory space.

Fully operational for position buffer. Functionality also present for
force buffer, but not yet called (awaiting acceptance of force buffer
ops patches).

Data transfer for halo exchange is wrapped and has 2 implementations:
cuda-aware MPI (default with "real" MPI) and direct cuda memcpy
(default with thread MPI). With the latter, the P2P path will be taken
if the hardware support it, otherwise D2H,H2D.

Limitation: still only supports 1D data decomposition

TODO: implement support for higher numbers of dimensions.
TODO: integrate call to force buffer halo exchange, when force buffer ops
patches accepted.

Implements part of #2890
Associated with #2915

Change-Id: I8e6473481ad4d943df78d7019681bfa821bd5798

**Revision 44f607d7 - 09/16/2019 03:12 PM - Alan Gray**

GPU halo exchange

Activate with GMX_GPU_DD_COMMS and GMX_USE_GPU_BUFFER_OPS environment
variable.

Class to initialize and apply coordinate buffer halo exchange
functionality directly on GPU memory space.

Currently only supports direct cuda memcpy, and relies on thread MPI being in use.

Updated gpucomm testing matrices to cover non-GPU case.

Limitation: still only supports thread MPI, 1D data decomposition and only coordinate halo exchange

Implements part of #2890
Associated with #2915

Change-Id: I8e6473481ad4d943df78d7019681bfa821bd5798

## History

#### #1 - 04/02/2019 05:37 PM - Alan Gray

Szilard wrote (copying from issue 2891):

> How do we provide fallbacks for when i) no MPI is used ii) no CUDA-aware MPI is used?
>
> - For the former, with tMPI I assume we can have a GPUDirect-based fallback.

Yes, we need to make threadMPI CUDA-aware. Jon did some work on this a while ago - I need to review what is there.

> - For the latter, how do we detect that we have a CUDA-aware MPI? What happens if we don't and the proposed code is invoked?

We can detect at runtime if OpenMPI is CUDA-aware with an env variable
https://www.open-mpi.org/faq/?category=runcuda#mpi-cuda-aware-support
I assume similar for other MPI libs. I suggest that, in the first instance, we tie this in with detection of the GMX_PURE_MULTI_GPI env variable such that we follow this schedule if and only if CUDA-aware MPI has also been detected.

#### #2 - 04/02/2019 06:19 PM - Szilárd Páll

Alan Gray wrote:

> Szilard wrote (copying from issue 2891):
>
> > How do we provide fallbacks for when i) no MPI is used ii) no CUDA-aware MPI is used?
> >
> > - For the former, with tMPI I assume we can have a GPUDirect-based fallback.
>
> Yes, we need to make threadMPI CUDA-aware. Jon did some work on this a while ago - I need to review what is there.

OK. Keep us posted. I think it should be straightforward, but let's identify if there is some additional infrastructure that's needed (I expect not, we should be able to just branch on whether tMPI + CUDA is used).

> > - For the latter, how do we detect that we have a CUDA-aware MPI? What happens if we don't and the proposed code is invoked?
>
> We can detect at runtime if OpenMPI is CUDA-aware with an env variable
> https://www.open-mpi.org/faq/?category=runcuda#mpi-cuda-aware-support
> I assume similar for other MPI libs.

Sure, but let's keep in mind that production functionality we need to have a clear message to users when is not supported and when is it not. We have little no experience with CUDA-aware MPI, so we need to start exploring early how will this be brought to a release (including documentation, build/compile/runtime checks, etc.)

> I suggest that, in the first instance, we tie this in with detection of the GMX_PURE_MULTI_GPI env variable such that we follow this schedule if and only if CUDA-aware MPI has also been detected.

This functionality will work and may be useful for any multi-GPU run whether only (most) forces or also constrains/update is offloaded. Hence, as soon as the PP-PME comm and X/F exchange are implemented, it should allow us to start evaluating it.

#### #3 - 04/03/2019 02:59 PM - Szilárd Páll

*- Related to Feature #2891: PME/PP GPU communications  added*

#### #4 - 05/31/2019 03:07 PM - Alan Gray

*- Description updated*

#### #5 - 06/12/2019 11:01 AM - Alan Gray

*- Description updated*

#### #6 - 06/12/2019 05:25 PM - Szilárd Páll

Alan Gray wrote:

> TODO
>
> - ~~Investigate if GPU halo exchange inter-GPU sync in thread-MPI case can be made more lightweight with cuStreamWaitValue32 (or similar).~~ (done with MPI exchange of pointers to events, with remote event enqueued to stream).

I need to review the new solution, but at first it does not seem to be equally efficient: the cuStreamWaitValue32 suggestion would allow enqueueing a dependency that can be resolved with no involvement of the CPU (the remote rank could write to the memory location to notify the receiver).

#### #7 - 08/01/2019 04:50 PM - Szilárd Páll

Having looked at the details of the proposed features to employ direct communication, here is a summary of the aspects we need to further consider while integrating the proposed prototypes into the main code-base.

### Communication setup flavors

First off, along the current staged communication, the following flavors of communication setup are relevant to consider:

- single rank / tMPI using GPU direct
  Pros:
  - lightweight receiver notification
  - receiver does asyncs dependency resolution using GPU streams,
  - one extra explicit CUDA API call, but MPI will have at least two internal CUDA API calls with higher launch overhead
  Cons:
  - only single-node (given no plans for PME decomp not a major limitation*)
  Questions:
  -

- multi-rank (real) MPI
  Pros:
  - multi-node scalability (only relevant with with RF* -- use-case: cg simulations)
  - less proprietary API code (?)
  Cons:
  - requires wait on the sender host code before communication
  - receiver is blocked in MPI until the data arrives to the GPU --> can't use async dependencies
  Questions:
  - unclear performance behavior in real-wold use-cases on common hardware

### Communication source-target cases:

Either source or target can be CPU or GPU memory; while current prototype assumes only GPU<->GPU transfers, CPU<->GPU (or GPU<->CPU) setups should be considered. In particular:

- current code produces new coordinates on the host, so both x PP->PME and halow exchange should be CPU->GPU;
- with GPU force reductions f PME->PP and halo exchange will be best done GPU<->GPU
- virial steps do reductions on the CPU, so for those steps *in the current code* f PME->PP and halo exchange GPU->CPU would make more sense (this can however be reconsidered TODO ref the GPU virial issue)

I suggest to make flags and pre-compute these at initialization to avoid composing conditions in multiple places which risks inconsistencies and it is harder to understand and maintain.

### Notes

- We should take a closer look at the force exchange case in particular both non-virial and virial steps as this brings the additional complexity related to virial reduction
- We need a good picture of the performance behaviour of the proposed schemes; as of now, without PME decomposition the real-MPI case seems of less use, in particular due to inherent technical limitations (if we'd have a mixed mode PME i.e. spread offloaded, this would become a much more relevant target for a lot of existing hardware).

#### #8 - 08/15/2019 03:11 PM - Alan Gray

@Szilard Given our discussion yesterday, would you like me to remove the CUDA-aware MPI path from each the comms patches, and upload these changes separately as WIP patches to go on the backburner for now?

#### #9 - 08/16/2019 05:09 PM - Szilárd Páll

Alan Gray wrote:

> @Szilard Given our discussion yesterday, would you like me to remove the CUDA-aware MPI path from each the comms patches, and upload these changes separately as WIP patches to go on the backburner for now?

Yes, please. I suggest splitting the tMPI and real MPI use-cases. Smaller commits are easier to understand and review, so adding new functionality step-by-step is always preferred.

I'd suggest even smaller incremental changes, e.g. the PP/PME direct comm change like adds a lot of logic to arrange for and sync on X H2D copy on the source side. Instead, it could well do a CPU->(remote) GPU transfer first, then in follow-up (possibly after the coordinates become resident on-GPU) add the x in GPU memory on the source side (GPU->GPU) case.

#### #10 - 08/16/2019 06:09 PM - Szilárd Páll

Szilárd Páll wrote:

> Alan Gray wrote:
>
> > @Szilard Given our discussion yesterday, would you like me to remove the CUDA-aware MPI path from each the comms patches, and upload these changes separately as WIP patches to go on the backburner for now?
>
> Yes, please. I suggest splitting the tMPI and real MPI use-cases. Smaller commits are easier to understand and review, so adding new functionality step-by-step is always preferred.

Also, decoupling features will allow prioritizing one thing over another and for that it would also be useful to avoid long dependency chains.

#### #11 - 02/14/2020 12:39 PM - Alan Gray

*- Parent task changed from #2816 to #3370*

#### #12 - 02/14/2020 12:48 PM - Alan Gray

*- Status changed from New to In Progress*