# MPI Tags issue in IPPL

**Sonali Mayani**

Paul Scherrer Institut (PSI)

September 10, 2024

# Issue (on IPPL Github)

Issue: `https://github.com/IPPL-framework/ippl/issues/282`

The tags found in src/Communicate/Tags.h for MPI communication start overlapping after a certain run size (i.e. number of MPI processes used), which causes correctness issues in code which relies on communication and uses these tags. For example, this issue became visible when running the test/solver/TestGaussian.cpp on more than 512 nodes on Perlmutter, each having 4 GPUs.

This is now temporarily fixed by increasing the absolute distance between the tags which may be used at the same time, by commit b27fa15ed95a322873500d70a57e5df58e32a04f.

However, this is still an issue that we will run into for bigger runs which may reach this overlap limit, therefore, a permanent solution for it should be found.

# Extent of the issue

- The issue was first found on Perlmutter GPUs when using more than 512 nodes ($\approx$ 2000 GPUs).

  $\rightarrow$ Fixed by increasing interval between `SOLVER_SEND` and `SOLVER_RECV` (before: 1000, now: 5000).

  $\rightarrow$ **Branch**: `scaling_study_vico_paper`.

- Happened again on Eiger, when using more than 64 nodes (64*128 $\approx$ 8000 MPI ranks).

  $\rightarrow$ Fixed by increasing interval between `SOLVER_SEND` and `SOLVER_RECV` (before: 5000, now: 16000).

  $\rightarrow$ **Branch**: `mpitags_fix_solverscaling`.

- Checked CPU scalings on Perlmutter again.
  $\rightarrow$ Same issue as Eiger.

# MPI Tags in Solver: Communication

```
// send
std::vector<MPI_Request> requests(0);

for (int i = 0; i < ranks; ++i) {
    if (lDomains2[i].touches(ldom1)) {
        auto intersection = lDomains2[i].intersect(ldom1);

        solver_send(mpi::tag::SOLVER_SEND, mpi::tag::OPEN_SOLVER, 0, i, intersection,
                    ldom1, nghost1, view1, fd_m, requests);
    }
}

// receive
const auto& lDomains1 = layout_mp->getHostLocalDomains();
int myRank            = Comm->rank();

for (int i = 0; i < ranks; ++i) {
    if (lDomains1[i].touches(ldom2)) {
        auto intersection = lDomains1[i].intersect(ldom2);

        mpi::Communicator::size_type nrecvs;
        nrecvs = intersection.size();

        buffer_type buf =
            Comm->getBuffer<memory_space, Trhs>(mpi::tag::SOLVER_RECV + myRank, nrecvs);

        Comm->recv(i, mpi::tag::OPEN_SOLVER, fd_m, *buf, nrecvs * sizeof(Trhs), nrecvs);
        buf->resetReadPos();

        unpack(intersection, view2, fd_m, nghost2, ldom2);
    }
}
```

# MPI Tags in Solver: solver_send

```
// Buffer message indicates the domain intersection (x, y, z, xy, yz, xz, xyz).
ippl::mpi::Communicator::buffer_type<memory_space> buf =
    ippl::Comm->getBuffer<memory_space, Tf>(BUF_MSG + id * 8 + i, nsends);

int tag = TAG + id;

ippl::Comm->isend(i, tag, fd, *buf, requests.back(), nsends);
buf->resetWritePos();
```

- BUF_MSG = mpi::tag::SOLVER_SEND
- id = 0
- i = i
- TAG = mpi::tag::OPEN_SOLVER

$\Longrightarrow$ buffer_ID = mpi::tag::SOLVER_SEND + i

$\Longrightarrow$ tag = mpi::tag::OPEN_SOLVER

# MPI Tags in Solver: Receive

```
// receive
const auto& lDomains1 = layout_mp->getHostLocalDomains();
int myRank            = Comm->rank();

for (int i = 0; i < ranks; ++i) {
    if (lDomains1[i].touches(ldom2)) {
        auto intersection = lDomains1[i].intersect(ldom2);

        mpi::Communicator::size_type nrecvs;
        nrecvs = intersection.size();

        buffer_type buf =
            Comm->getBuffer<memory_space, Trhs>(mpi::tag::SOLVER_RECV + myRank, nrecvs);

        Comm->recv(i, mpi::tag::OPEN_SOLVER, fd_m, *buf, nrecvs * sizeof(Trhs), nrecvs);
        buf->resetReadPos();

        unpack(intersection, view2, fd_m, nghost2, ldom2);
    }
}
```

$\Longrightarrow$ buffer_ID = mpi::tag::SOLVER_RECV + myRank

$\Longrightarrow$ tag = mpi::tag::OPEN_SOLVER

# MPI Tags in Solver: Summary

- It seems like the problem is actually not the tag itself but the fact that it is used to get the buffer ID.
- The tag of the messages is always the same (`mpi::tag::OPEN_SOLVER`).
- From `src/Communciate/Buffers.hpp`: "If a buffer is requested with the same ID as a buffer that has been previously allocated, the same buffer will be used."
- So probably the issue is that once the buffer ID of the send overlaps with the receive (i.e. when `i` > interval size between `mpi::tag::SOLVER_SEND` and `mpi::tag::SOLVER_RECV`), then the sending rank writes over a buffer that is actually being used to receive by another rank. This messes up the communication.

# Testing whether this is the issue

Change the following in `solver_send`:

```
    // Buffer message indicates the domain intersection (x, y, z, xy, yz, xz, xyz).
    ippl::mpi::Communicator::buffer_type<memory_space> buf =
-       ippl::Comm->getBuffer<memory_space, Tf>(BUF_MSG + id * 8 + i, nsends);
+       ippl::Comm->getBuffer<memory_space, Tf>(i, nsends);
```

This means that if

$$\texttt{i < mpi::tag:SOLVER\_RECV},$$

then there is no problem (and `mpi::tag:SOLVER_RECV = 25000` so that is okay for 128*128 ranks $\approx$ 16'000).

$\rightarrow$ This works. Ran 10 jobs with this, all of them ran correctly, whereas with the original code, 2 out of 5 runs were incorrect.

$\Longrightarrow$ This is indeed the issue.