Supporting large counts in MPI

Jeff Hammond, Intel Labs
(thanks to George, Rajeev, Pavan for comments)

Background

- MPI-3 largely punted on large-count support
- A handful of MPI_Foo_x routines make rudimentary large-count support possible
- Forum asserted that “just use datatypes” is a sufficient solution for users.
- Obviously, no one bothered to verify the aforementioned assertion...

BigMPI

Purpose: provide a high-level library that supports large counts that can be used as drop-in replacement for existing MPI routines and *test the Forum's assertion that datatypes are sufficient for large-count support.*

https://github.com/jeffhammond/BigMPI

Functions that were easy

- all variants of send and receive
- bcast, gather, scatter, allgather, alltoall (blocking and nonblocking)
- RMA (put, get, accumulate, get_accumulate)

Each of these functions follows the pattern shown on the next slide...

```c
int MPIX_Send_x(const void *buf, MPI_Count count, MPI_Datatype datatype,
                int dest, int tag, MPI_Comm comm)
{
    int rc = MPI_SUCCESS;

    if (likely (count <= bigmpi_int_max )) {
        rc = MPI_Send(buf, (int)count, datatype, dest, tag, comm);
    } else {
        MPI_Datatype newtype;
        MPIX_Type_contiguous_x(count, datatype, &newtype);
        MPI_Type_commit(&newtype);
        rc = MPI_Send(buf, 1, newtype, dest, tag, comm);
        MPI_Type_free(&newtype);
    }
    return rc;
}
```

The magical large-count datatype

Please see src/type_contiguous_x.c in BigMPI.

In the general case, MPI_Type_create_struct is required, although BigMPI tries to factorize the count into C integers so we can use MPI_Type_vector.

Ticket 423 would improve user experience because it addresses the common case of large counts of built-ins.

Reductions were not easy

- User-defined ops required for user-defined types. This would be fixed by tickets 34+338.
- How do I support MPI_IN_PLACE inside of my user-defined reduction? I want to use MPI_Reduce_local…
- Blocking reductions can use the cleaver.

V-collectives are hard

- All V-collectives turn into ALLTOALLW because different counts implies different large-count types.
- ALLTOALLW displacements given in bytes are C int and therefore *it is impossible* to offset more than 2GB into the buffer.
- NEIGHBOR_ALLTOALLW to the rescue?!?!

Using NEIGHBOR_ALLTOALLW

- MPI_Neighbor_alltoallw displacements are MPI_Aint not int. **This is good.**
- Neighborhood collectives require special communicators that must be created for each call (and possibly cached).
- Must allocate new argument vectors and, in the case of “!alltoall”, we wastefully splat the same value in all locations.

V-collectives using P2P and RMA

- One can follow the definition in MPI to implement all of the V-collectives using P2P.
- RMA (with win_fence synchronization) also works for the V-collectives.
- Allgatherv using nproc calls to Bcast also works.
- Large-count definitely outside of recursive doubling regime so little to optimize...

V-collectives - nonblocking issues

None of the aforementioned solutions works for nonblocking because:

- What request do we return in the case of P2P or RMA?
- Cannot free argument vectors until complete.
- Any solution involving generalized requests is untenable for users. BigMPI might use it.

Neighborhood collectives

- Scalar collectives are easy.
- V-collectives: map to ALLTOALLW
- Same problem as before with nonblocking regarding the allocated argument vectors.
- If not for MPI_Aint displacements in ALLTOALLW, we would have to drop into P2P and MPICH generalized requests.

Summary of tickets

- 34+338 (+339?) required for sane reductions.
- 423 is sugar but it greatly reduces user pain and is trivial to implement.
- 430 is required to support nonblocking v-collectives in a straightforward way.
- MPICH-style generalized requests required (https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/Proposal should be ticket-ized)