SAND2022-6299C

MPI Sessions, Persistent Collectives, and Partitioned Communication

Matthew Dosanjh
Sandia National Laboratories

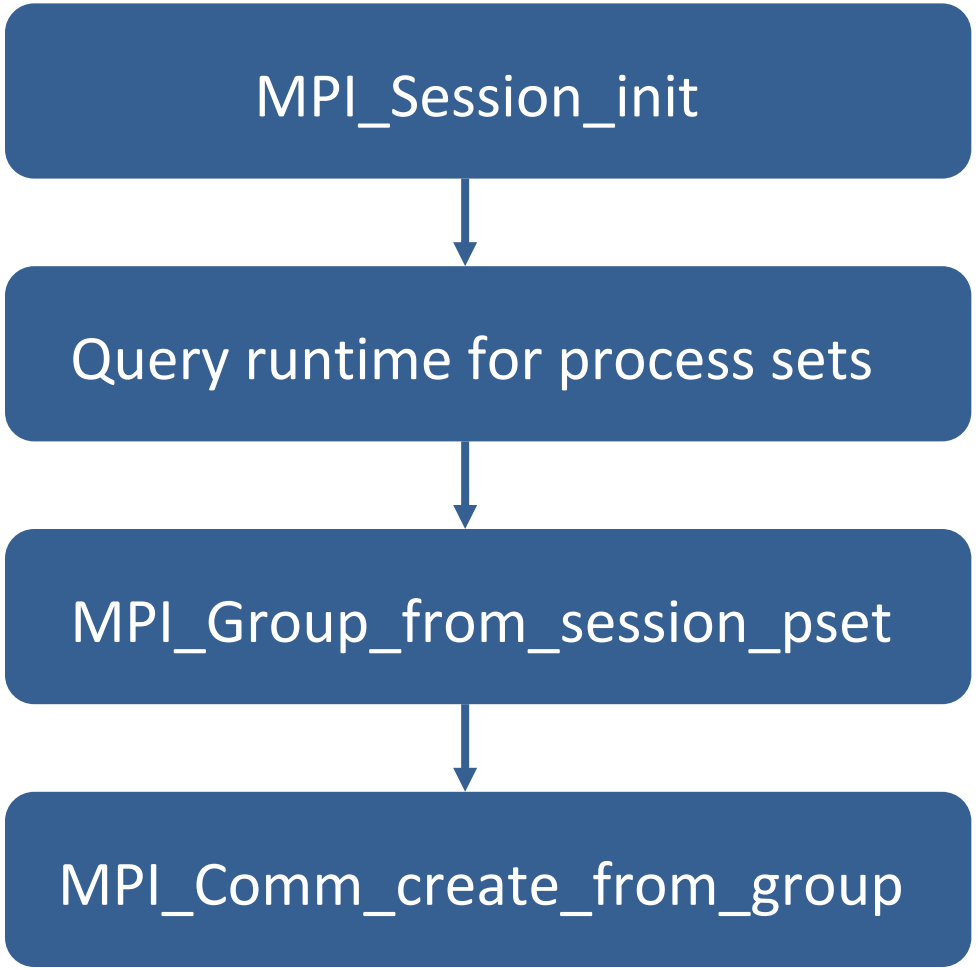
Based on slides by
Howard Prichard
Los Alamos Laboratory

EXASCALE COMPUTING PROJECT

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

MPI Sessions

- In the MPI 4.0 standard
- Sessions API supported in MPICH 4 release stream and Open MPI 5.0 release stream (not yet released)
- Complete implementation of MPI 4 Sessions API
- Requires PMIx 4 or newer (for Open MPI)
- Set of examples are available at https://github.com/hppritcha/mpi_sessions_tests



MPI Persistent Collectives

- Released in the MPI 4 standard at Barcelona MPI Forum Meeting (9/18)
- Aims to accelerate applications with repetitive collective operations
- Initialization call for a persistent collective operation is **non-local,** all members of the communicator being supplied to the initialization operation must eventually invoke this initialization call
- The info argument to these initialization calls can be used to specify MPI implementation specific optimizations.

MPI Persistent Collectives

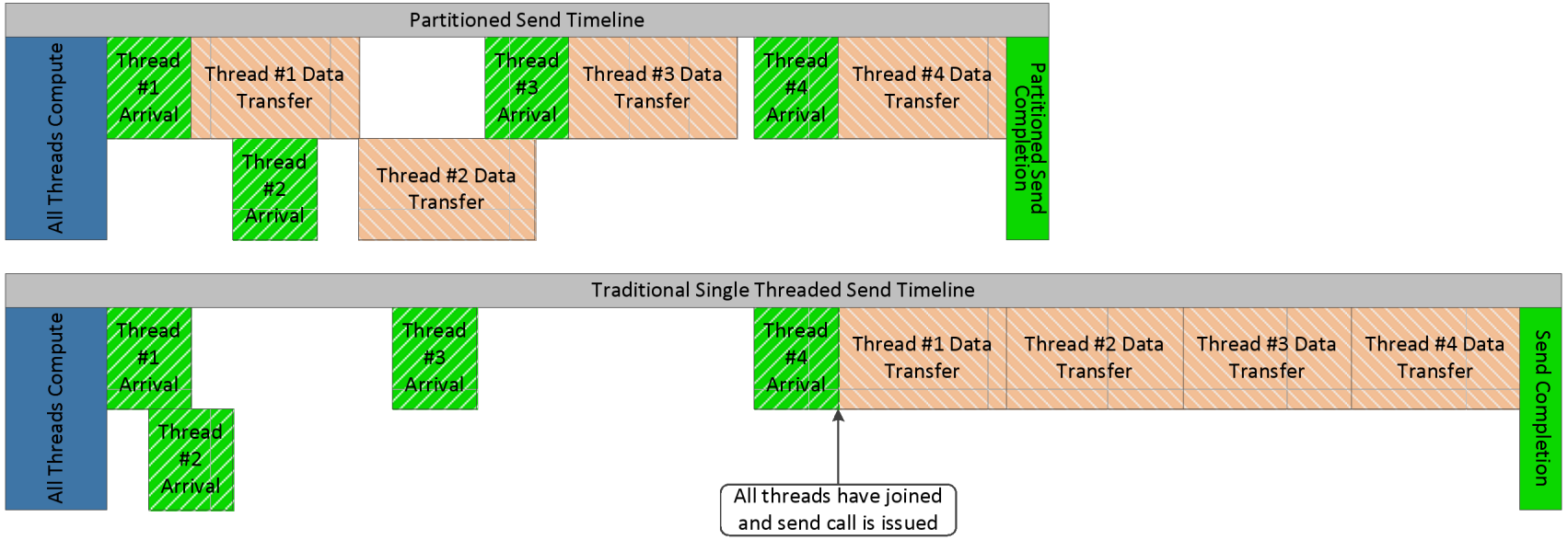
- Starting/completion semantics
- MPI_*_init creates a *request* and may be used zero or more times to start the corresponding collective operation using *MPI_Start* or *MPI_Startall*.
- This request must be inactive before starting a persistent collective operation
  - Starting a persistent collect operation makes it *active*
  - Only one outstanding collective operation on a request
- Loosens ordering constraints
  - Ordering matters for MPI_*_init not MPI_Start*.
- *MPI_Wait, MPI_Test,* etc. completes the operation but does not free the request

Partitioned Communication

- Share “ownership” of a buffer between MPI and user code
  - Similar to RMA Windows
- Uses a persistent buffer ‘partitioned’ into smaller chunks
- User informs MPI when data is ‘ready’ to send.
  - User code must manage memory epochs
  - Once a partition is marked ready, it’s data can not be changed
- Thread agnostic with a minimal synchronization overhead

Early Bird Communication

- New type of overlap
- Data can be transferred before the last compute thread is finished
- Can consolidate some messaging overhead
- Targeted for accelerator use in a future version of the MPI standard



Example (send side)

```
MPI_Psend_init(message, send_partitions, 1, send_type, dest, tag,
          info, MPI_COMM_WORLD, &request);
MPI_Start(&request);
#pragma omp parallel for shared(request) num_threads(NUM_THREADS)
for (int i=0; i<send_partitions; i++)
{
   /* compute and fill partition #i, then mark ready: */
   MPI_Pready(i, &request);
}
while(!flag)
{
   /* Do useful work */
   MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
   /* Do useful work */
}
MPI_Request_free(&request);
```

Example (receive side)

```
MPI_Precv_init(message, recv_partitions, recv_partlength, MPI_DOUBLE,
          source, tag, info, MPI_COMM_WORLD, &request);
MPI_Start(&request);
#pragma omp parallel for shared(request) num_threads(NUM_THREADS)
for (int j=0; j<recv_partitions; j+=2)
{
   int part1_complete = 0;
   int part2_complete = 0;
   while(part1_complete == 0 || part2_complete == 0)
   {
      /* test partition #j and #j+1 */
      MPI_Parrived(&request, j, &flag);
      if(flag && part1_complete == 0)
      {
         part1_complete++;
         /* Do work using partition j data */
      }
      if (j+1 < recv_partitions) {
         MPI_Parrived(&request, j+1, &flag);
         if(flag && part2_complete == 0)
         {
            part2_complete++;
            /* Do work using partition j+1 */
         }
      }
      else {
          part2_complete++;
      }
   }
}
```

```
.
.
.
while(!flag)
{
   /* Do useful work */
   MPI_Test(&request, &flag, MPI_STATUS_IGNORE);
   /* Do useful work */
}
MPI_Request_free(&request);
```

Partitioned communication (MPI 4.0) implementation status

- General implementation layered on top of MPI
  - https://github.com/tonyskjellum/MPIPCL
- Available in MPICH since v4.0a1 release
- Available in Main branch of OpenMPI