MPI-4 BOF

Anthony Skjellum, PhD
University of Tennessee at Chattanooga
tony-skjellum@utc.edu

SC'18

Standardization Topic areas

- Persistent Collective Communication
- Big MPI (64-bit clean MPI)
- Additions for Collective Communication – Nonblocking Constructors/Destructors
- “WITH_INFO” proposals
- “Other” clarifying tickets in the works
- All targeted for the MPI-Next (MPI-4) Release of MPI

Persistent Collective Operations

- Use-case: a collective operation is done many times in an application
- The specific sends and receives represented never change (size, type, lengths, transfers)
- A persistent collective operation can take the time to apply a heuristic and choose a faster way to move that data
- Fixed cost of making those decisions could be high but can be amortized over all the times the operation is used
- Static resource allocation can be done
- Choose fast(er) algorithm, take advantage of special cases
- Reduce queueing costs
- Special limited hardware can be allocated if available
- Choice of multiple transfer paths could also be performed

Basics

- Mirror regular nonblocking collective operations
- For each nonblocking MPI collective, add a persistent variant
- For every MPI_I<coll>, add MPI_<coll>_init
- Parameters are identical to the corresponding nonblocking variant – plus additional MPI_INFO parameter
- All arguments “fixed” for subsequent uses
- Persistent collective operations cannot be matched with blocking or nonblocking collective calls

Example

Nonblocking collectives API

```
for (i = 0; i < MAXITER; i++) {
    compute(bufA);
    MPI_Ibcast(bufA, …, rowcomm, &req[0]);
    compute(bufB);
    MPI_Ireduce(bufB, …, colcomm,
&req[1]);
    MPI_Waitall(2, req, …);
}
```

Persistent collectives API

```
MPI_Bcast_init(bufA, …, rowcomm, &req[0]);
MPI_Reduce_init(bufB, …, colcomm,
&req[1]);
for (i = 0; i < MAXITER; i++) {
    compute(bufA);
    MPI_Start(req[0]);
    compute(bufB);
    MPI_Start(req[1]);
    MPI_Waitall(2, req, …);
}
```

Init/Start

- The init function calls only perform initialization; do not start the operation
- Ex: MPI_Allreduce_init
  - Produces a persistent request (not destroyed by completion)
- Requests work with MPI_Start/MPI_Startall
- Only inactive requests can be started
- MPI_Request_free can free inactive requests

Ordering of Inits and Starts

- Inits must be ordered like all other collective operations
- Persistent collective operations can be started in the same order, or different orders, at all processes
- MPI_Startall can contain multiple operations on the same communicator due to ordering freedom
- A new communicator INFO key will be added that asserts persistent collectives starts will be strictly ordered
- In some cases, this may improve performance
- NB: INFO key incompatible with starting multiple persistent collective operations using MPI_Startall

Big MPI

- Idea: Finally make MPI fully 64-bit clean
- More 2 Gi element transfers "is a pain"
- Solve issue across API: Collectives, Point-to-point, I/O, and RMA
- Started with a significant study and prototyping effort by Jeff Hammond that yielded a proposal for Collective communication
  - Workarounds possible for pt2pt do not work well for collectives
  - v/w-collectives and reductions have the most concerns
- Neighborhood collectives fixed the large-displacement problem (for these new ops)
- Produces _X API variants names that are "64-bit clean"
- MPI_Rank vs. int rank is a controversial part of this discussion
- Alternatives: Alternative/new APIs or polymorphic APIs

Nonblocking Collective Constructors/Destructors

- Idea: Make initialization/de-initialization fully nonblocking
- Support asynchronous libraries better
- Ex: Add nonblocking variants (wherever missing)
  - MPI_COMM_CREATE -> MPI_COMM_ICREATE
  - MPI_COMM_FREE -> MPI_COMM_IFREE
  - MPI_WIN_CREATE -> MPI_WIN_ICREATE
  - MPI_WIN_FREE -> MPI_WIN_IFREE
  - MPI_FILE_CLOSE -> MPI_FILE_ICLOSE
  - …
- Plus other collective operations that lack a nonblocking analog (except we avoid RMA at present)
- Ex: MPI_FILE_SET_INFO -> MPI_FILE_ISET_INFO

Standardization of Persistence

- https://github.com/mpi-forum/mpi-issues/issues/25
- Ticket #25 approved for MPI-4 in September (Barcelona)
- https://github.com/mpi-forum/mpi-issues/issues/83
- Ticket #83 – reread in December (San Jose)
- https://github.com/mpi-forum/mpi-issues/issues/90
- Ticket #90 clarifies text throughout the standard properly to introduce “persistence” in several places where it is not fully mentioned or documented order – to be read again in December, 2018

Standardization of "Big MPI"

- https://github.com/mpi-forum/mpi-issues/issues/80
- Ticket #80 addresses "Large Count support" and 64-bit clean displacements for **collective op**s –read in Barcelona – Sept. 2018 – reread in December 2018
- https://github.com/mpi-forum/mpi-issues/issues/98
- Applies Ticket #80 concepts to I/O chapter – maybe 12/18
- https://github.com/mpi-forum/mpi-issues/issues/99
- Applies Ticket #80 concepts to RMA chapter – to be read in Barcelona – maybe 12/18
- https://github.com/mpi-forum/mpi-issues/issues/100
- Applies Ticket #80 concepts to point-to-point operations – to be read in Barcelona – maybe 12/18

Standardization of "Big MPI" cont'd

- https://github.com/mpi-forum/mpi-issues/issues/97
- Ticket #97 proposes a new MPI_Rank type in all APIs that use ranks – Reading TBD
- Int rank -> MPI_rank rank [globally in API]
- Highly controversial

Standardization of "Nonblocking"

- https://github.com/mpi-forum/mpi-issues/issues/78
- Ticket #78 addresses Communicator, File, and Win nonblocking operations – to be read in Barcelona – Sept. 2018
- https://github.com/mpi-forum/mpi-issues/issues/81
- Ticket #81 addresses nonblocking constructors/destructor for the Dynamic Process Management Chapter – to be read in Barcelona – Sept. 2018
- https://github.com/mpi-forum/mpi-issues/issues/82
- Ticket #82 addresses nonblocking constructors/destructor for the RMA Chapter – to be read in Barcelona – Sept. 2018

WITH_INFO API modifications

- Idea: Make MPI operations that omit "info" have "info"
- Focus:
  - Blocking and Nonblocking collective operations
  - Constructors
- Allow MPI programs to hint to operations more uniformly throughout the API
- A limited number of these functions already are present, such as MPI_DUP_WITH_INFO, MPI_IDUP_WITH_INFO
- Ex:

```
MPI_Ibcast(void* buffer, int count, MPI _Datatype datatype,
  int root, MPI _Comm comm, MPI_Request *request)

MPI_Ibcast_with_info(void* buffer, int count, MPI _Datatype datatype,
  int root, MPI _Comm comm, MPI_Info info, MPI_Request *request)
```

Standardization of "WITH_INFO"

- https://github.com/mpi-forum/mpi-issues/issues/84
- Ticket #84 addresses GRAPH and CART constructors in Topologies chapter – to be read in Barcelona – Sept. 2018 [generalizes 2 APIs]
- https://github.com/mpi-forum/mpi-issues/issues/85
- Ticket #85 "Collective and Topologies Chapters"– blocking and nonblocking operations – to be read in Barcelona – Sept. 2018 [generalizes 31 APIs]

Other [Collective] Tickets

- https://github.com/mpi-forum/mpi-issues/issues/87
- Allow MPI_PROC_NULL as neighbor in (distributed) graph topologies – likely will be read in Barcelona, Sept. 2018
- https://github.com/mpi-forum/mpi-issues/issues/89
- Deprecate MPI_Graph – likely will be read in Barcelona, Sept. 2018 --Advise users to move to MPI_Dist_graph_create() and associated functions instead

Thank You!

Contact:
tony-skjellum@utc.edu