MPI Collectives and Topology Workgroup

Atlanta, GA, January 2010

Torsten Hoefler

Agenda

- Topological Collectives
  - A concrete interface proposal
  - Problems
  - Solutions and Discussion

- Graph Topology issues

Motivation

- Stencil is probably the most important scalable communication pattern used!
- Different stencil operations
  - Regular (e.g., Cartesian) – each process is identical
    - e.g., simple CFD
  - Irregular – each process can be different
    - e.g., sparse matrix vector, AMR
- MPI supports optimization for both operations through graph topologies
  - Only allows to optimize mapping (which is important!)
  - Comm. scheduling can improve performance further
    - e.g., DCMF's (BG/P) Multisend interface (IPDPS'10); IPDPS'09

Overview and Terminology

- Establish directed communication graph between nodes
  - Each node has set of incoming and outgoing neighbors
    - IN = set of incoming neighbors (receive from)
    - OUT = set of outgoing neighbors (send to)
  - Topology can be optimized statically (offline)
    - Reorder ranks for network
    - Establish/optimize routing tables for minimum congestion
    - Optimize communication schedules for neighborhood collectives!
  - Similar to persistent requests
    - Buffers are not bound statically though (more flexible, network-centric)

Neighbor Gather

- MPI_Neighbor_gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
  - Gather data from all incoming neighbors to local buffer
    - single item in sendbuf
    - |IN| items in recvbuf
  - Broadcast single item to all OUT processes
  - Receive item from IN[i] at position i

  - Both can be optimized as a tree if |IN| is large enough
  - Intelligent scheduling

Neighbor Alltoall

- MPI_Neighbor_alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
  - Exchange of all data
    - |OUT| items in sendbuf
    - |IN| items in recvbuf
  - Send item i in sendbuf to OUT[i]
  - Receive item I in recvbuf from IN[i]

  - Dissemination- or Bruck-like algorithms
  - Communication scheduling

Neighbor Reduce

- MPI_Neighbor_reduce(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, op, comm)
  - Reduce items from neighbors
    - Single item in sendbuf
    - Single item in recvbuf
  - Apply op to all items from IN
  - Send item to OUT

  - Enables tree-based implementations
  - Intelligent scheduling

Vector Versions

- MPI_Neighbor_gatherv()
- MPI_Neighbor_alltoallv()
- MPI_Neighbor_alltoallw()
- MPI_Neighbor_reducev()
  - Special case!
  - Different connected neighborhoods have different numbers of elements
  - User needs to supply the correct values (non-trivial)!

Discussion

- Similar to persistent point-to-point
  - Persistent communication pattern
    - Build your own collective ☺
  - Non-persistent data
    - Does not conflict with persistent point-to-point
    - Should be discussed by persistence WG (is it active?)
- Nonblocking variants are easy to add
  - MPI_Ineighbor_*(…, request)

Example 1: Regular Stencil (MPI-2.2 p285)

```
int ndims=2, dims[2];
MPI_Dims_create(&comm, &ndims, &dims);
MPI_Comm comm_cart;
Int periods[2] = {1,1};
MPI_Cart_create(&comm, &ndims, &dims, &periods, 1, &comm_cart);
double sbuf[4], rbuf[4]; // exchange single double with each neighbor
for(int i=0; i<100; ++i) {
    prepare_buffers(u, sbuf, rbuf);
    MPI_Request req;
    MPI_Ineighbor_alltoall(sbuf, 1, MPI_DOUBLE, rbuf, 1, MPI_DOUBLE,
            comm_cart, &req)
    relax(u);
    MPI_Wait(&req, MPI_STATUS_IGNORE);
}
```

Example 2: Irregular Stencil (MPI-2.2 p272)

```
// create graph topology (See example 7.3)
int in, out, weighted;
MPI_Dist_graph_neighbors_count(gcomm, &in, &out, &weighted);
int ine[in], oute[out], inw[in], outw[out];
MPI_Dist_graph_neighbors(gcomm, in, ine, inw, out, oute, outw);
double sbuf[out], rbuf[in]; MPI_Request req;
for(int i=0; i<100; ++i) {
   prepare_buffers(u, in, ine, out, oute, sbuf, rbuf);
   MPI_Ineighbor_alltoall(sbuf, 1, MPI_DOUBLE, rbuf, 1,
         MPI_DOUBLE, gcomm, &req)
   relax(u);
   MPI_Wait(&req, MPI_STATUS_IGNORE);
}
```

Biggest Problem!

- Most current MPI programs process incoming data as soon as it arrives
  - Synchronizing this with a collective adds additional and unnecessary overhead!
  - Will not be adopted by power-users if this problem persists!
- Do we see more problems?

Solutions!

- Streaming completion Version 1:
  - Also applicable to dense collectives ☺
    - Gather, Allgather, Alltoall
  - MPI_Cwaitany(count, request, rank)
- Streaming completion Version 2:
  - MPI_Sneighbor_alltoall (…, requests)
    - |IN| requests
    - Normal MPI requests

- Streaming completion also for send neighbors?
  - If not, we need an extra request for them

Example 3: Streaming Irregular Stencil

```
// same preamble as in example 2
MPI_Request reqs[in+1]; // one for all outgoing edges
for(int i=0; i<100; ++i) {
   prepare_buffers(u, in, ine, out, oute, sbuf, rbuf);
   MPI_Sneighbor_alltoall(sbuf, 1, MPI_DOUBLE, rbuf, 1,
          MPI_DOUBLE, gcomm, &reqs)
   relax(u);
   for(int j=0; j<in+1; j++) {
          int idx;
          MPI_Waitany(in+1, reqs, &idx, MPI_STATUS_IGNORE);
          if(idx!=in) merge(u, idx)
   }
}
```

Concrete Proposal

- Option 1: Addition to Collectives Chapter 5
  1. Suboptimal because Topologies are Chapter 7!
- Option 2: Addition to Topology Chapter 7
  1. Defining text in Section 7.6
  2. Examples in Section 7.7
- Option 3: Addition to Topology Chapter 7
  1. Defining text in Section 7.7
  2. Examples in Section 7.7
- Two examples:
  1. Regular stencil
  2. Irregular stencil/Arbitrary graph

MPI_Cart_create (#194/#195)

- See tickets (D’oh)!