MPI-3 Persistent WG (Point-to-point Persistent Channels)

December 7, 2010

Tony Skjellum, Coordinator

MPI Forum

Purpose of the Persistent WG

- More performance
- More scalability
- Impact pt2pt positively
- Impact collective positively
- Eventually look at 1-sided :-)
- Small new # of functions; concepts
- Exploit temporal locality and program regularity, exploit opportunity for early binding
- Ex: Want to cut critical path for send/receive to far below “500 instructions”

What we discussed in Chicago, October Meeting

- Ability to accomplish a lot with 1 or 2 new API calls only (MPI_Bind(), MPI_Rebind()); a few more for more functionality
- Ability to enhance MPI scalability with such functions for regular communications
- Setup/teardown
- Number of messages in flight (1)
- Reactivating the next message in flight
- Range of “persistence”.. Fully early to fully late, with ability to rebind
- Extensions to collective are obvious and we want to include these (but let’s formalize pt2pt version first)
- No slides from October (computer crash)

MPI_Bind/Rebind (2 functions)

- Purpose: reduce the critical path for regular, repetitive transmissions as much as possible
- Build persistent channels
- Define syntax and semantics for OOB point-to-point transmissions that pinch off from communicators
- Two functions, plus one variant:
  - MPI_Bind(), MPI_Rebind()
  - Recommended nonblocking too: MPI_Ibind(), MPI_Irebind()
- Enhanced semantics of Start, Startall, Wait, Waitall, etc for these kinds of requests

MPI_Bind starts with standard persistent requests

- These don't change (including all variants Ssend, Rsend, but no Bsend):
  - int MPI_Send_init(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *S_request)
  - int MPI_Recv_init(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *R_request)
- These first stage requests will be transformed into persistent requests
- These first stage requests can be reused
- Both sides must do bind in a bounded time of each other; matching is the sequential post order per proc.

Protocol and Rendezvous

- There is a bind protocol:
  - Send side: MPI_Bind(S_Request, &PS_Request, info, comm);
  - Recv side: MPI_Bind(R_Request, &PR_Request, info, comm);
- This is OOB from other communication on the communicator comm, and matches exactly as the original pt2pt send/recv would match
- Once bound, the communication space is independent of comm, and no ordering guarantees exist between this pair and other communication on comm
- S_Request, R_Request still exist at this point, and are not related to the Bind; they can be reused for normal communication on the communicator, or as input to further bind calls
- Info: anything special we want to indicate in terms of algorithmic issues, specialized network hints, etc

Protocol and Rendezvous, 2

- Non blocking variants possible too:
  - Send side: MPI_Ibind(S_Request, &PS_Request, info, comm, &EphReq);
  - Recv side: MPI_Ibind(R_Request, &PR_Request, info, comm, &EphReq);
- Blocking and nonblocking binds can match
- The “Ephemeral Requests” are simply waited on for completion like any other pt2pt operation.

Using the bound requests

- The sender wants to send, then he/she does
  - MPI_Start(PS_Request, &status);
- The receiver wants to receive, then he/she does
  - MPI_Start(PR_Request, &status);
- You can wait/test (in future, mprobe)
- Sender side: You cannot Start() again until waited.
- Receiver side: Must have received data before restart.
- Only one message in flight
- Rationale: Sender side test should be lightweight (no polling) to allow for lightweight test for completion
- Receive side: Waitsome or Test should allow lightweight (non polling or short polling) approaches to allow for quick test for completion

Auto Addressing Buffers: Managing abstractions like autoincrement send/receive

- A special form of early binding allows for the incrementation of a buffer by a fixed length, mod N, so that a circular send and/or receive buffer is enabled, with a single channel. {N=stride to next buffer, not buffer len}
- For a DMA underlying implementation, this would be a “modify” and a “kick” instead of just a “kick”. Still can be quite fast compared to full weight send, receive, particularly because the channel could algorithmically set this up in advance
- This use case requires adding more syntax, but it is very valuable to capture
- *New datatype approach is a clever way to get this feature FYI. Need full concept, but the use case is valuable.*

Bundles of pairwise transfers

- To allow expression multiple messages in flight (parallel channels), applications could make sets of bound requests for a given pair of {sender,receiver}
- A set of requests is enumerated list of persistent requests made by bind. They can allow a user program (e.g., with ascending tags), to order a set of transmissions, round robin, and therefore ensure full bandwidth transmission between pairs, rather than "stop and wait" behavior as a single in flight channel would imply.
- We can leave all that up to the user, each request has to be bound.
- Optionally, we could provide something to support bundles

Bundles con’d

- Because there is a relatively high cost for bind, we can offer additional API for MPI_Mbind():
  - Send side: MPI_Mbind(S_Request, &PS_Request_Array, N_requests, info, comm);
  - Recv side: MPI_Mbind(R_Request, &PR_Request_Array, info, comm);
  - This array of requests can be used in any way by the sender/receiver, with order of matching pairwise
  - Right now, we require N_requests to be same on sender and receiver; we will explore non-one-to-one later
- Not to be overly lugubrious, but we could just have this API, and use N_requests=1, to keep # of functions to a tiny number

Rebind Protocol and Rendezvous

- There is a rebind protocol too:
  - int MPI_Rebind(void *buf, int count, MPI_Datatype datatype, int dest, int tag, [need enum of xfer protocol], MPI_Comm comm, MPI_Info info, INOUT MPI_Request *S_request)
  - int MPI_Rebind(void *buf, int count, MPI_Datatype datatype, int source, int tag, [need enum of xfer protocol], MPI_Comm comm, MPI_Info info, INOUT MPI_Request *R_request)

- Use same comm as in original bind.
- This is OOB from other communication on the communicator comm, and matches exactly as the original pt2pt send/recv would match
- Once bound, the communication space is independent of comm, and no ordering guarantees exist between this pair and other communication on comm
- This should be cheaper than a Bind(), but not necessarily.
- Mrebind() on subsets of the original set of channels made by Mbind() is allowed too.

Clean up

- MPI_Request_free() is not our favorite way of ending channels
- Recommend:
  - MPI_Bind_free(MPI_Request *req, int N_requests)
  - These can be any N channels made over any communicators with any pairs, and is loosely collective between the pairs only
  - MPI_Ibind_free() suggested also! We always prefer non-blocking :-)
- The original communicator or communicators used to form this channel could have already been destroyed before this happens, that's OK

MPI_Info in Bind/Rebind

Plan of Action

- Formal proposal - First 2010 Meeting
- Need implementation work - Need to decide who, when, how (Tony and volunteers?)
- Define implementation
- Define goal for when we try are ready for first reading.
- Should implementation prove out performance gain
  - Not required, per Rich
  - Required, per Tony (we won't get votes if not fast)