COL380

Introduction to
Parallel & Distributed Programming

Send/Recv Synchronization

- Synchronized
  - Send waits for Receive to complete?
  - Synchronization (up to network delay)
- Asynchronous
  - Sender may proceed immediately
    - May or may not need to wait until message is copied out
    - Receiver may also proceed immediately (message arrives later)

Eager vs Rendezvous

Eager

- Send-stub packetizes and transmits
  - (May make local copy first)
- Send-stub signals Done
- Recv-stub continuously accepts
- Delivered when Recv call matched

Rendezvous

- Send-stub transmits envelope info
- May make local copy
- Recv-stub continuously accepts envelope info
- Recv-stub may signal OK (if it has space)
  - Or, wait for matching Recv call
- Recv-stub sets up RDMA with Send-stub
- Data transmitted
- Recv-stub signals Done
- Send-stub signals Done

Send Semantics

- Standard mode: MPI_Send
  - ➡ implementation dependent
- Buffered mode MPI_Bsend
  - ➡ MPI saves a copy of message, Receiver can post later
  - ➡ User provided buffer — See MPI_Buffer_attach
- Synchronous mode MPI_Ssend
  - ➡ Will complete only once a matching receive has started
- Ready mode MPI_Rsend
  - ➡ Send may start only if a matching receive has already been called
  - ➡ Helps performance

- MPI_Send/MPI_Recv are blocking
  - ➡ Recv blocks until output buffer is filled
  - ➡ Send blocks until some 'progress'

Subodh Kumar

Message Semantics

- In order (per pair and tag)
  - ➡ Multi-threaded applications need to be coordinate
- Progress
  - ➡ For a matching send/Recv pair, at least one of these two will complete
- Fairness not guaranteed
  - ➡ A Send or a Recv may starve because all matches are satisfied by others
- Resource limitation can cause deadlocks
- Ready/Synchronous sends requires the least resources
  - ➡ Also used for debugging

Example

```
If (rank == 0)
    Send(to 1);
    Recv(from 1);
else
    Send(to 0);
    Recv(from  0);
```

Non-blocking

- MPI_Isend() / MPI_Irecv()
  - ➡ Non-blocking: Control returns after setup
  - ➡ Blocking and non-blocking Send/Recv match
  - ➡ Still lower Send overhead if Recv has been posted
- All four modes are applicable
  - ➡ Limited impact for buffered and ready modes
- Syntax is similar to Send and Recv
  - ➡ MPI_Request* parameter is added to Isend and replaces MPI_Status* for IRecv

Non-blocking Call

```
int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag,
   MPI_Comm comm, MPI_Request *request)
```

```
int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int
   tag, MPI_Comm comm, MPI_Request *request)
```

Detecting Completion

- MPI_Wait(&request, &status)
  - ➡ status similar to Recv
  - ➡ Blocks as per the blocking version's semantics
    - Send: message was copied out, Recv was started, etc.
    - Recv: Wait for data to fill
  - ➡ Request is freed as a side-effect
- MPI_Test(&request, &flag, &status)
  - ➡ Non-blocking poll
  - ➡ flag indicates whether operation is complete
  - ➡ Request is freed as a side-effect

Also see:

MPI_Waitany, MPI_Waitall, MPI_Waitsome

MPI_Testany, MPI_Testall, MPI_Testsome

Use MPI_Request_get_status to retain request

Later MPI_Request_free

Receiver Peek

- MPI_Probe(source, tag, comm, &flag, &status)
- MPI_Iprobe(source, tag, comm, &flag, &status)
    - Check information about incoming messages without actually receiving them
    - Check message size, e.g.
    - Next (matching) Recv will receive it
- MPI_Cancel(&request)
    - Request cancellation of a non-blocking request (no de-allocation)
    - Itself non-blocking: marks for cancellation and returns
    - Best effort, book-keeping necessary, can later check if cancelled

Persistent Send/Recv

- MPI_Send_init(buf, count, datatype, dest, tag, comm, &request);
- MPI_Start(&request);
- MPI_Start is non-blocking
  - blocking versions do not exist
- There is also MP_Start_all
  - And MPI_Recv_init
  - And MPI_Bsend_init etc.
- Reduces Process interaction with the Communication system

Review Basics

- Send - Recv is point-to-point
  - Call-to-call matching
  - Integer tag to distinguish message streams
  - Wildcard matching: MPI_ANY_SOURCE and MPI_ANY_TAG
- Recv buffer must contain enough space for message
  - Receiving fails otherwise
  - Can query the actual count received (MPI_Get_count)
    - Send determines the actual number sent
  - type parameter determines data<->buffer copying

MPI Data types

MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_LONG_LONG	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t
MPI_BYTE	

Objects of type MPI_Datatype

Subodh Kumar