

MPI I - Part 1

Nicholas Mc Guire
OpenTech EDV Research GmbH
Mistelbach - AUSTRIA

Schedule

- Basic communication
- Some definitions
- Point-to-Point
- Resource Issues
- Program "Safty"

Basic Communication

The very basic communication model is where the MPI starts off:

- Point-to-Point Communication
- Blocking operations

Understanding this very basic communication model and how it works will help clarify many of the advanced features!

IPC vs. MPI

Is MPI reinventing the wheel ?

- IPC - Process centric
- MPI - Data centric

MPI is a change in paradigm - trying to apply IPC concepts result in poor code.

Implications of Data-centricity

- High communication latency prohibit process centric
- Data loss is the key disaster scenario
- Conceptually MPI can be seen as operating on a single

MPI design is focused on data management issues.

Communication terms

- blocking: communication function complete - buffers
- non-blocking: return before completion - don't touch
- local: no communication required - state depends on
- non-local: status depends on other mpi process - com
required

Operations

- Send: initiator of communication - push data
- Recv: end-point of communication - waits for data
- Scheduling: users schedules communication tasks and executing them

Don't assume too much about the data transfer in MPI

Point-to-Point Semantics

MPI follows a "PUSH" not a "PULL" data model.

- Ordering: Non-Overtaking within a given communication.
- Fairness: MPI is not fair !
- Progress: MPI guarantees - at least one of the Send operations will complete.
- Resource: Ready and Sync sends require preallocated resources, other modes may consume dynamic resources.

Determinism

Requirements for deterministic Point-to-Point communication

- Single threaded (i.e. LAM)
- No use of MPI_ANY_SOURCE in Receive
- prevent MPI_CANCEL, MPI_WAIT_ANY

System level determinism exhibits lower performance but on the side of the program designer.

Send Modes

MPI provides four modes of blocking send operations.

- MPI_RSend: Ready send
- MPI_BSend: Buffered send
- MPI_SSend: Synchronous send
- MPI_Send: standard send (implementation dependant)

Receive

Receive modes:

- non-overtaking
- only one receive mode
- receives are blocking

Send Phases

System perspective of data transmission:

- Pull data from send buffer
- transfer data
- Pull data from network stack to receive buffer

Communication engine

- default mechanism not standardized
- Blocking means "locally safe"
- Default non-local
- Buffering is not specified
- Programs should not assume specific behavior unless requested.

Management

- System resource insecurity
- MPI permits explicit buffering
- Lack of buffering:
 - BSend fails
 - Receive blocks indefinitely

MPI Communication Safety

- Safe programs don't rely on system buffering
- Test of safety:

MPI_Send -> MPI_SSend

--> No Deadlock

--> Portable over MPI implementation

Caveats

- Mixing Modes
- MPI_ANY_SOURCE
- Implicitly relying on system buffering

1. Conclusion

- focus on resource management - you must know wh
- program design must include explicit communication
- you must be aware if you operation is safe or unsafe
- MPI is not IPC - don't apply the wrong paradigm