The MPI Message-passing Standard Practical use and implementation (V)

SPD Course

12/10/2012

Massimo Coppola

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

UNIVERSITÀ DI PISA

HPC laboratory

Intracommunicators

COLLECTIVE COMMUNICATIONS

Collectives' Characteristics

- Collective operations are called by ALL processes of a communicator
  - Happen in a communicator like p-to-p
  - Use Datatypes to define message structure
  - Implement complex communication patterns
- Distinct semantics from point-to-point
  - No modes
  - Always blocking
  - No variable-size data
  - No status parameters (would require many…)
  - Limited concurrency
- Still a lot of freedom left to implementers
  - E.g. actual pattern choice, low-level operations
  - Semantics is carefully defined for this aim

Changes! with MPI 3.0

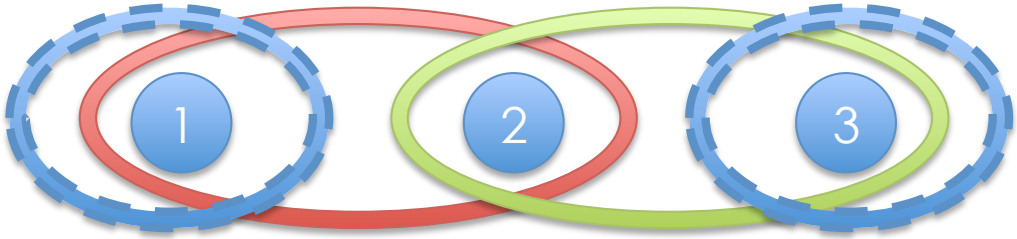
- **MPI standard 3.0 released in September 2012**
  - Collective Communications **can** be non-blocking
  - In this course we will stick to the MPI 2.2 definition
- ***After*** studying the blocking version, it might worth to know about non-blocking collectives
  - implicit serialization within a communicator still holds
  - blocking and non-blocking collectives **do not** match with each other
  - all completion calls (WAIT, TEST) are supported
  - multiple outstanding collectives allowed in same communicator
  - non-blocking behavior can avoid collective-related deadlock across communicators
    - interaction with collective serialization **is** significant
  - it is not allowed to cancel a non-bl. collective

Collective & Communicators

- Independence among separate communicators
- Independence with p-to-point in same comm.
  - Although coll. may be implemented on top of p-to-p.
- Collectives are serialized over a communicator
  - Obvious consequence of the semantics
  - Same actual call order from every process in the communicator
- Serialization **is not** synchronization
  - Blocking behaviour = after the call, local completion is granted and buffer / parameters are free to be reused
  - Globally, the collective may still be ongoing (and vice versa)
  - Example: broadcast on a binary support tree may complete on root process long before it is done
  - Only the MPI_Barrier is granted to synchronize
- Serialization **is** a source of deadlocks

Example of deadlocks and errors

- Serialization **is** a source of deadlocks



BAR BRD BAR

BRD BAR BAR

OK

BAR BAR BAR

BAR BAR BAR

Deadlock!

Collective Primitives

- Many of the primitives you already know
  - Synchronization: Barrier (*also an all-to-all*)
  - One-to-all: Bcast (*broadcast*), Scatter *
  - All-to-one: Gather *, Reduce
  - All-to-all: AllGather *, AllToAll *, AllReduce, ReduceScatter
  - Other (*comp.*): Scan (*parallel prefix*), Exscan
- agreement on parameters among all proc.s
  - Who is the root
  - Transferred data
  - More constraints on the typemaps, not only signatures

Collective Primitives

- Agreement on data to be transferred
  - Buffers defined at each process must match in size
  - Sometimes used for reading AND writing
- User-defined datatypes and type signatures are allowed
  - Typemaps should be compatible as always
  - *Writing* typemaps shall not be redundant
    - No ambiguity shall ever arise from typemap access order, which is free choice of the MPI library
  - Generally speaking, collective primitives should not read or write twice the same location
    - Not discussing all cases, refer to the standard

Barrier & Broadcast

- int MPI_Barrier(MPI_Comm comm)
  - Can be applied to intercommunicators

- int MPI_Bcast(void* buffer, int count,
  MPI_Datatype datatype, int root,
  MPI_Comm comm)