MPI-3.0 Overview

Gian Franco Marras[1]

[1]CINECA - SuperComputing Applications and Innovation Department - SCAI, Via Magnanelli 6/3, 40033 Casalecchio di Reno, Bologna (Bo),
g.marras@cineca.it

February 11-15, 2013

Outline

1. Introduction
   - MPI documents
   - Implementations of the MPI-3.0 standard

2. Background of MPI-3.0
   - Fortran 2008 binding
   - Nonblocking Collectives
   - Neighborhood Collective Communication
   - Extension to one-side operations
     - Extending MPI with Integrated Shared Memory
     - RMA interface for shared memory

MPI-3.0: pdf files

www.mpi-forum.org

- MPI documents
  - MPI 3.0 document as PDF
  - Versions of MPI 3.0 with alternate formatting
  - Errata for MPI 3.0

Implementations of the MPI standard

Open MPI

openmpi-1.7: http://www.open-mpi.org/software/ompi/v1.7/

- Added MPI-3 functionality:
    - MPI_GET_LIBRARY_VERSION
    - Matched probe
    - MPI_TYPE_CREATE_HINDEXED_BLOCK
    - Non-blocking collectives
    - MPI_INFO_ENV support
    - Fortran '08 bindings

MPICH

mpich-3.0.2: http://www.mpich.org/downloads/

Background of MPI-3.0

Overview of new features in MPI-3

- Nonblocking versions of collective operations;
- Neighborhood collective communication;
- Extensions to one-sided operations;
- Added tools interface;
- New Fortran 2008 binding;
- Removed deprecated C++ bindings;
- Removed many of the deprecated routines and MPI objects.

Background of MPI-3.0

Overview of new features in MPI-3

- Nonblocking versions of collective operations;
- Neighborhood collective communication;
- Extensions to one-sided operations;
- Added tools interface;
- New Fortran 2008 binding;
- Removed deprecated C++ bindings;
- Removed many of the deprecated routines and MPI objects.

Language binding

Three methods of Fortran support:

- **INCLUDE** 'mpif.h'

  strongly discouraged, because this method neither guarantees compile-time argument checking nor provides sufficient techniques to solve the optimization problems with non-blocking calls.

- **USE** mpi

  inconsistent with the Fortran standard, therefore not recommended;

- **USE** mpi_f08

  It's consistent with the Fortran Standard, Fortran 2008 + TS 29113 and later (NEW!);

Fortran 2008 binding

- An additional set of bindings for the latest Fortran specification;
- Guarantees compile-time argument checking;
- Declares each argument with an `INTENT` of `IN`, `OUT` or `INOUT` ad defined in this standard;
- Declares all `ierror` output arguments as `OPTIONAL`;
- Uses the `ASYNCHRONOUS` attribute to protect the buffers of non-blocking operations;
- Non contiguous sub-array can be used as buffers in non-blocking routines (`MPI_SUBARRAY_SUPPORTED=.TRUE.`);
- Fixes many other issues with the old Fortran 90 bindings.

MPI-2.2

```
    MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG,
&                         COMM, REQUEST, IERROR)
        <type>      BUF(*)
        INTEGER     COUNT, DATATYPE, DEST, TAG,
&                         COMM, REQUEST, IERROR
```

MPI-3.0

```
MPI_Isend(buf, count, datatype, dest, tag, &
              comm, request, ierror) BIND(C)
TYPE(*),DIMENSION(..),INTENT(IN),ASYNCHRONOUS :: buf
INTEGER,                  INTENT(IN)  :: count, dest, tag
TYPE(MPI_Datatype),       INTENT(IN)  :: datatype
TYPE(MPI_Comm),           INTENT(IN)  :: comm
TYPE(MPI_Request),        INTENT(OUT) :: request
INTEGER, OPTIONAL,        INTENT(OUT) :: ierror
```

Nonblocking Collective Communication

Collective communication

- Collection of pre-defined optimized routines.

Nonblocking communication:

- Deadlock avoidance;
- Overlapping communication and computation.

Three Types:

- Synchronization (Barrier);
- Data Movement (Scatter, Gather, Alltoall, Allgather);
- Reductions (Reduce, Allreduce, Scan);

Collective Communication

Nonblocking Collective Communication

- Non blocking variants of all collectives, they return an *MPI_Request* object:
  - MPI_Ibcast(<bcast args>, MPI_Request *req);
- Semantic:
  - Function returns no matter what;
  - No guaranteed progress;
  - The user must call *MPI_Test*/*MPI_Wait* or their variants to complete the operation;
  - Out-of order completion.
- Restrictions:
  - No tags, in-order matching;
  - Multiple non-blocking collectives may be outstanding, but they must be called in the same order on all processes;
  - No matching with blocking collectives.

Neighborhood Collectives

Neighborhood Collectives Communication

- New functions *MPI_Neighbor_allgather*, *MPI_Neighbor_alltoall*, and their variants define collective operations among a process and its neighbors;
- Neighbors are defined by an MPI Cartesian or graph virtual process topology that must be previously set;
- These functions are useful, for example, in stencil computations that require nearest-neighbor exchanges;

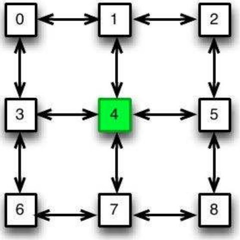
Neighborhood Collectives

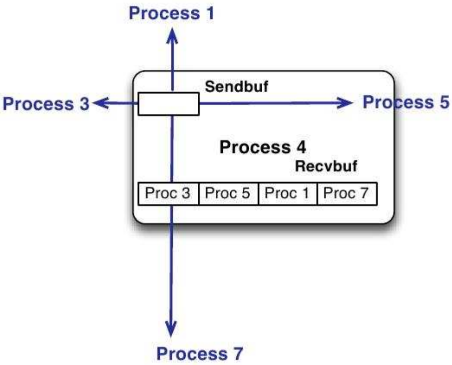
MPI_Neighbor_allgather

```
int MPI_Neighbor_allgather(
      const void *sendbuf, int sendcount,
      MPI_Datatype sendtype, void *recvbuf,
      int recvcount, MPI_Datatype recvtype,
                              MPI_Comm comm)
```

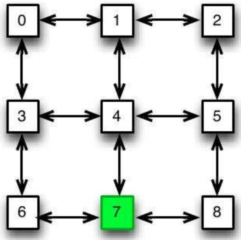
- The central process sends the same message to all neighbors;
- The neighbors receive the same message;
- Similar to MPI_Gather.
- Vector e non-blocking versions for full flexibility.

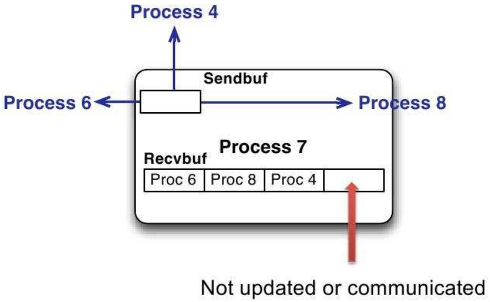
MPI_Neighbor_allgather





MPI_Neighbor_allgather





Neighborhood Collectives

MPI_Neighbor_alltoall

```
int MPI_Neighbor_alltoall(
        const void *sendbuf, int sendcount,
        MPI_Datatype sendtype, void *recvbuf,
        int recvcount, MPI_Datatype recvtype,
                                MPI_Comm comm)
```

- The central process sends outdegree distinct messages;
- The neighbors receive distinct messages;
- Similar to MPI_Alltoall.
- Vector, w and non-blocking versions for full flexibility.

Improved RMA interface

- Substantial extensions to the MPI-2 RMA interface;
- New window creation routines:
  - MPI_Win_allocate: MPI allocates the memory associated with the window;
  - MPI_Win_create_dynamic: Creates a window without memory attached. User can dynamically attach and detach memory to/from the window by calling MPI_Win_attach and MPI_Win_detach;
  - MPI_Win_allocate_shared: Creates a window of shared memory (within a node) that can be used for direct load/store accesses in addition to RMA operations.
- New atomic read-modify-write operations;
  - MPI_Get_accumulate;
  - MPI_Fetch_and_op (simplified version of Get_accumulate);
  - MPI_Compare_and_swap .

Improved RMA interface

- Substantial extensions to the MPI-2 RMA interface;
- New window creation routines:
  - MPI_Win_allocate: MPI allocates the memory associated with the window;
  - MPI_Win_create_dynamic: Creates a window without memory attached. User can dynamically attach and detach memory to/from the window by calling MPI_Win_attach and MPI_Win_detach;
  - MPI_Win_allocate_shared: Creates a window of shared memory (within a node) that can be used for direct load/store accesses in addition to RMA operations.
- New atomic read-modify-write operations;
  - MPI_Get_accumulate;
  - MPI_Fetch_and_op (simplified version of Get_accumulate);
  - MPI_Compare_and_swap .

Extending MPI with Integrated Shared Memory

Introduction

- MPI's remote memory access (RMA) interface defines one-sided communication operations, data consistency, and synchronization models for accessing memory regions that are exposed through MPI windows.
- The MPI-3 RMA interface extends MPI-2's separate memory model with a new unified model, which provides relaxed semantics that can reduce synchronization overheads and allow greater concurrency in interacting with data exposed in the window.
- The unified model was added in MPI-3 RMA to enable more efficient one-sided data access in systems with coherent memory subsystems.
- The public and private copies of the window are logically identical, and updates to either "copy" automatically propagate.
- Explicit synchronization operations can be used to ensure completion of individual or groups of operations.

Shared Memory Windows

Using the RMA interface for shared memory

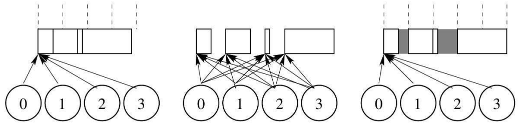
- In the MPI-2 one-sided communication interface, the user first allocates memory and then exposes it in a window;
- MPI_Win_allocate_shared collectively allocates and maps shared memory across all processes in the given communicator;
- All processes in the given communicator must be in shared memory;
- Load/store operations do not pass through the MPI library.

```
int MPI_Win_allocate_shared(
        MPI_Aint size, int disp_unit,
        MPI_Info info, MPI_Comm comm,
        void *baseptr, MPI_Win *win)
```

Shared Memory Windows

Memory Layout

- Default:
  - Memory is consecutive across ranks;
  - Allow for inter-rank address calculations,i.e., process *i*'s memory starts where process *i-1*'s memory ends;
- Optimizations allowed:
  - With info "alloc_shared_noncontig" may create non-contiguous locations;
  - This can enable better performance and eliminates negative cache and NUMA effects.
  - Each windows segment is aligned to optimize memory access.



(a) Contiguous (b) Noncontig Separate (c) Noncontig Padded

Shared Memory Comm Creation

How do I know which processes share memory?

```
int MPI_Comm_split_type(
        MPI_Comm comm, int split_type,
        int key, MPI_Info info,
                  MPI_Comm * newcomm)
```

- Creates a shared-memory communicator and allocates the entire work array in shared memory;
- Portable;
- `split_type = MPI_COMM_TYPE_SHARED`
- Splits communicator into maximum shared memory islands;

Shared Memory Windows address query

How do I query the process address for remote memory segments created?

```
int MPI_Win_shared_query(
        MPI_Win win, int rank,
        MPI_Aint *size, int *disp_unit,
                            void *baseptr)
```

- MPI_Win_allocate_shared does not guarantee the same virtual address across ranks;
- This function can return different process-local addresses for the same physical memory on different processes;
- MPI_Win_shared_query provides a query mechanism for determining the base address in the current process and size of another process's region in the shared-memory segment.