MPI-2 Overview

Rolf Rabenseifner
rabenseifner@hlrs.de

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Acknowledgements

Parts of this course is based on MPI-2 tutorial on the MPIDC 2000:

MPI-2: Extensions to the Message Passing Interface

MISSISSIPPI STATE UNIVERSITY [1]
HIGH PERFORMANCE COMPUTING LAB
NSF ENGINEERING RESEARCH CENTER

H L R S [2]

Anthony Skjellum[1]
Purushotham Bangalore[1], Shane Hebert[1]
Rolf Rabenseifner[2]

MPI-2 Outlook

- MPI-2, standard since July 18, 1997
- Chapters:
  - Version 1.2 of MPI (Version number, Clarifications)
  - Miscellany (Info Object, Language Interoperability, New Datatype Constructors, Canonical Pack & Unpack, C macros)
  - Process Creation and Management (MPI_Spawn, ...)
  - One-Sided Communications
  - Extended Collective Operations
  - External interfaces (..., MPI and Threads, ...)
  - I/O
  - Language Binding (C++, Fortran 90)
- All documents from http://www.mpi-forum.org/
  (or from www.hlrs.de/mpi/)

MPI-1.2

- New function to obtain version of the MPI Standard implemented
- Compile time information
  - integer MPI_VERSION=1, MPI_SUBVERSION=2
- Runtime information
  - MPI_GET_VERSION(*version*, *subversion*)
- MPI_GET_VERSION can be called before MPI_INIT and after MPI_FINALIZE
- Clarifications and corrections to MPI-1.1
  - pointer to MPI-2 Chapter 10.2.2
    Problems with Fortran Bindings for MPI

MPI 1.2 — Clarifications to MPI 1.1

- MPI_INITIALIZED
  - behavior not affected by MPI_FINALIZE
- MPI_FINALIZE
  - user must ensure the completion of all pending communications (locally) before calling finalize
  - is collective on MPI_COMM_WORLD
  - may abort all processes except “rank==0” in MPI_COMM_WORLD
- Status object after MPI_WAIT/MPI_TEST
- MPI_INTERCOMM_CREATE
- MPI_INTERCOMM_MERGE

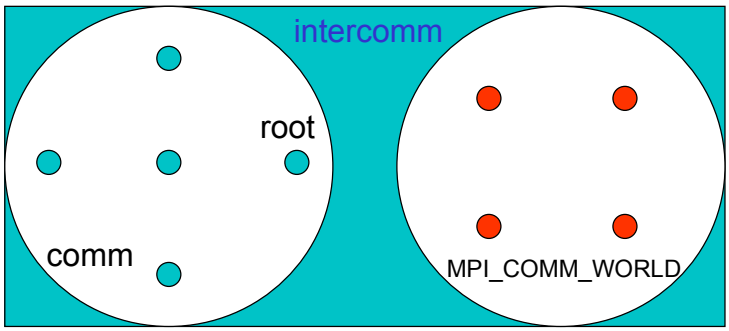
MPI 1.2 — Clarifications to MPI 1.1 (continued)

- Bindings for MPI_TYPE_SIZE
  - output argument in C is of type `int`
- MPI_REDUCE
  - the **`datatype`** and **`op`** for predefined operations must be same for all processes
- MPI_PROBE and MPI_IPROBE
- Attribute callback functions error behavior
- Other minor corrections
  - with nonblocking & persistent communications and MPI_Address:
    - **Fortran problems with data copying and sequence association**
    - **Fortran problems with register optimization**

Dynamic Process Management

- Goals
  - starting new MPI processes
  - connecting independently started MPI processes
- Issues
  - maintaining simplicity, flexibility, and correctness
  - interaction with operating systems, resource manager, and process manager
- Spawn interfaces:
  - at initiators (parents):
    - **Spawning new processes is *collective*, returning an intercommunicator.**
    - **Local group is group of spawning processes.**
    - **Remote group is group of spawned processes.**
  - at spawned processes (children):
    - **New processes have own MPI_COMM_WORLD**
    - **MPI_Comm_get_parent() returns intercommunicator to parent processes**

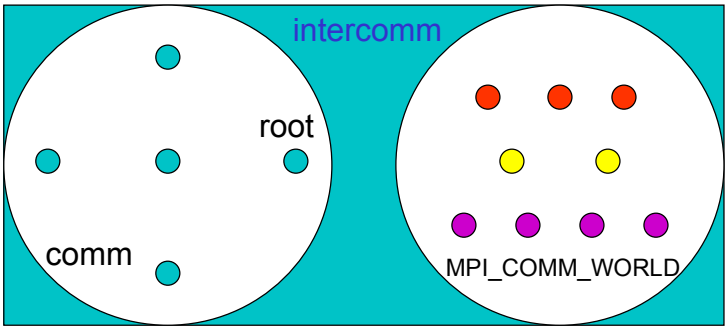
Dynamic Process Management — Get the intercomm, I.



Parents:
MPI_COMM_SPAWN (,
root,comm, ***intercomm***,...)

Children:
MPI_Init(...)
MPI_COMM_GET_PARENT(***intercomm***)

Dynamic Process Management — Get the intercomm, II.



Parents:

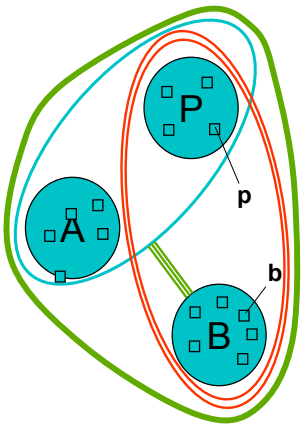
MPI_COMM_SPAWN
_MULTIPLE (3, .,
root,comm, ***intercomm***,...)

Children:

MPI_Init(...)
MPI_COMM_GET_PARENT(***intercomm***)

Dynamic Process Management — Multi-merging, a Challenge

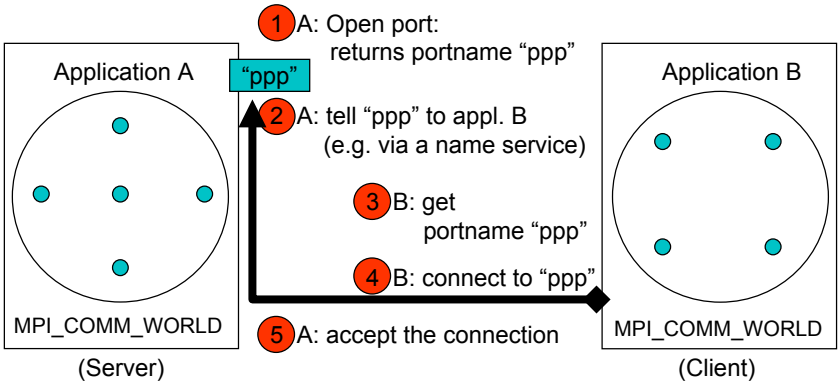
- If a comm. P spawns A and B sequentially, how can P, A and B communicate in a single intracomm?
- The following sequence supports this:
  - P+A merge to form intracomm PA
  - P+B merge to form intracomm PB
  - PA and B create intercomm PA+B
    [using PB as peer, with **p, b** as leaders]
  - PA+B merge to form intracomm PAB
- This is not very easy, but does work



Dynamic Process Management — MPI_Info Object

- An **`MPI_Info`** is an opaque object that consists of a set of (key,value) pairs
  - both key and value are strings
  - a key should have a unique value
  - several keys are reserved by standard / implementation
  - portable programs may use **`MPI_INFO_NULL`** as the info argument, or sets of vendor keys
  - Several sets of vendor-specific keys may be used
- Allows applications to pass environment-specific information
- Several functions provided to manipulate the info objects
- Used in: *Process Creation*, *Window Creation*, *MPI-I/O*

Dynamic Process Management — Establishing Communication



Result: An intercommunicator between both original communicators

Dynamic Process Management — Another way

- Another way to establish MPI communication
- MPI_COMM_JOIN(fd, *intercomm*)
- joins by an intercommunicator
- two independent MPI processes
- that are connected with Berkley Sockets of type SOCK_STREAM

Dynamic Process Management — Singleton INIT

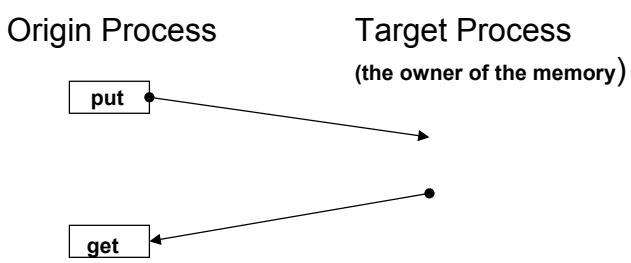
- High quality MPI's will allow single processes to start, call MPI_INIT(), and later join in with other MPI programs
- This approach supports
  - parallel plug-ins to sequential APPs
  - other transparent uses of MPI
- Provides a means for using MPI without having to have the "main" program be MPI specific.

One-Sided Operations

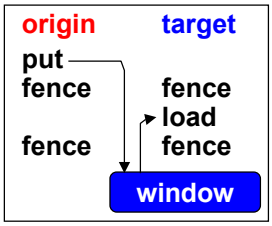
- Goals
  - PUT and GET data to/from memory of other processes
- Issues
  - Synchronization is separate from data movement
  - Automatically dealing with subtle memory behavior: cache coherence, sequential consistency
  - balancing efficiency and portability across a wide class of architectures
    - **shared-memory multiprocessor (SMP)**
    - **clusters of SMP nodes**
    - **NUMA architecture**
    - **distributed-memory MPP's**
    - **workstation networks**
- Interface
  - PUTs and GETs are surrounded by special synchronization calls

One-Sided Operations — Origin and Target

- Communication parameters for both the sender and receiver are specified by one process (origin)
- User must impose correct ordering of memory accesses



One-Sided Operations — An Example

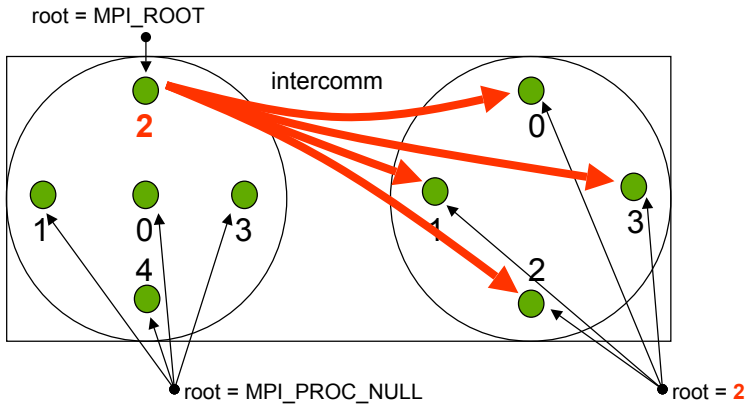


- The target process declares a window with
  - **MPI_WIN_CREATE(base_addr, win_size, disp_unit, info, comm, *win*)**
- Synchronization necessary between
  - remote memory access (RMA)
    - **MPI_PUT**
    - **MPI_GET**
    - **MPI_ACCUMULATE**
  - and local memory access
    - **loads and stores, generated by the compiler**
- Three synchronization methods:
  - **MPI_FENCE (like a barrier)**
  - **Post / start / wait / complete (point-to-point synchronization)**
  - **Lock / unlock (allows passive target communication)**

Extended Collective Operations

- In MPI-1, collective operations are restricted to ordinary (intra) communicators.
- In MPI-2, most collective operations are extended by an additional functionality for intercommunicators
  - e.g., Bcast on a parents-children intercommunicator: sends data from one parent process to all children.
- Provision to specify "*in place*" buffers for collective operations on intracommunicators.
- Two new collective routines:
  - generalized all-to-all
  - exclusive scan

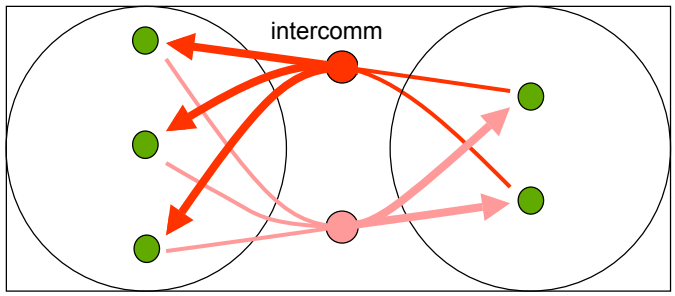
Extended Collective Operations — MPI_Bcast on *inter*comm.



broadcast communication

Extended Collective Operations — MPI_Allgather on *inter*comm.

remember: allgather = gather+broadcast



allgather communication

Extended Collective Operations — "In place" Buffer Specification

The **MPI_IN_PLACE** has two meanings:

- to prohibit the local copy:
  - GATHER(V), SCATTER(V) at root node
  - ALLGATHER(V) at any node
- to overwrite input buffer with the result: (sendbuf=**MPI_IN_PLACE**, input is taken from recvbuf, which is then overwritten)
  - REDUCE at root
  - ALLREDUCE, REDUCE_SCATTER, SCAN at any node

Extended Collective Operations — Generalized All-to-all

- The most general form of all-to-all
- Allows separate specification of count, displacement, and **datatype**
- Displacement is specified in terms of no. of bytes to allow maximum flexibility
- Useful for matrix transpose and corner-turn operations

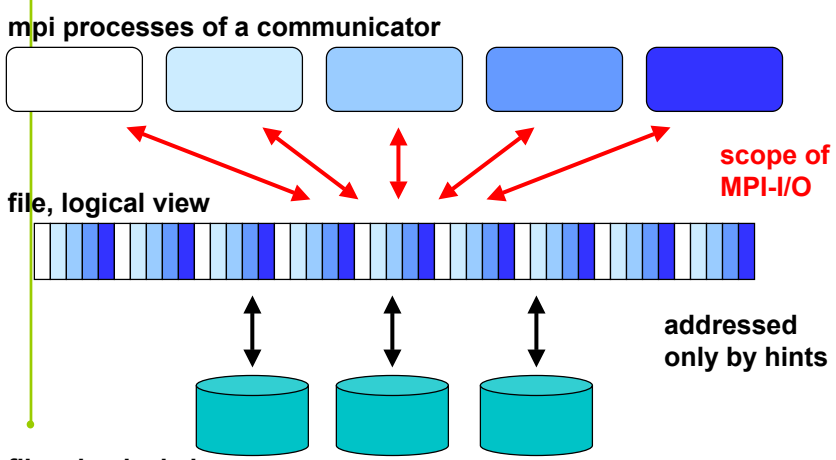
MPI_Alltoall**w**(sendbuf, sendcounts, sdispls, **sendtypes**, *recvbuf*, recvcounts, rdispls, **recvtypes**, comm)

- recvtypes, sendtypes now both arrays

MPI - I/O

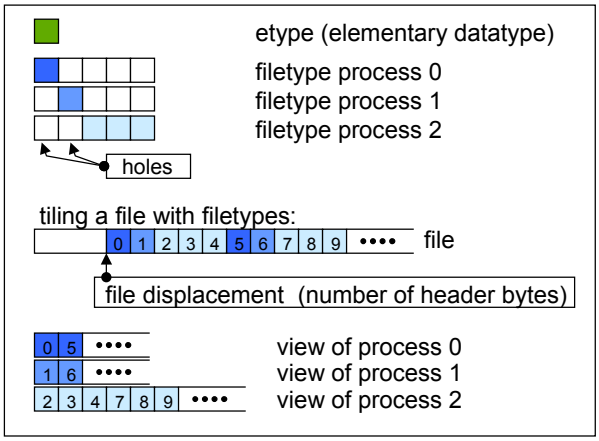
- Goals:
  - reading and writing files in parallel
- Rich set of features:
  - Basic operations: open, close, read, write, seek
  - noncontiguous access in both memory and file
  - logical view via *filetype* and *element-type*
  - physical view addressed by hints, e.g. “striping_unit”
  - explicit offsets / individual file pointers / shared file pointer
  - collective / non-collective
  - blocking / non-blocking or split collective
  - non-atomic / atomic / explicit sync
  - “native” / “internal” / “external32” data representation

MPI - I/O — Logical view / Physical view



MPI - I/O — Definitions

- etype
- filetype
- file
- displacement
- logical view



MPI - I/O — Example with Subarray: Reads and distributes a matrix

```
!!!!    real garray(20,30)                    ! these HPF-like comment lines !
!!!!    PROCESSORS   P(2,  3)                 ! explain the data distribution !
!!!!    DISTRIBUTE garray(BLOCK,BLOCK)        ! used in this MPI program      !
        real larray(10,10) ;     integer (kind=MPI_OFFSET_KIND) disp ;  disp=0

        ndims=2 ; psizes(1)=2 ; period(1)=.false. ; psizes(2)=3 ; period(2)=.false.
        call MPI_CART_CREATE(MPI_COMM_WORLD, ndims, psizes, period,
                                                        .TRUE., comm, ierror)
        call MPI_COMM_RANK(comm, rank, ierror)
        call MPI_CART_COORDS(comm, rank, ndims, coords, ierror)

        gsizes(1)=20 ;  lsizes(1)= 10 ;  starts(1)=coords(1)*lsizes(1)
        gsizes(2)=30 ;  lsizes(2)= 10 ;  starts(2)=coords(2)*lsizes(2)
        call MPI_TYPE_CREATE_SUBARRAY(ndims, gsizes, lsizes, starts,
                        MPI_ORDER_FORTRAN, MPI_REAL, subarray_type, ierror)
        call MPI_TYPE_COMMIT(subarray_type , ierror)

        call MPI_FILE_OPEN(comm, 'exa_subarray_testfile', MPI_MODE_CREATE +
                                MPI_MODE_RDWR, MPI_INFO_NULL, fh, ierror)
        call MPI_FILE_SET_VIEW (fh, disp, MPI_REAL, subarray_type, 'native',
                                MPI_INFO_NULL, ierror)
        call MPI_FILE_READ_ALL(fh, larray, lsizes(1)*lsizes(2), MPI_REAL,
                                status, ierror)
```

Other MPI-2 Features (1)

- Standardized Process Startup: **mpiexec**
- C / C++ / Fortran language interoperability
- Datatypes:
  - New constructors:
    - **MPI_Type_create_darray / ..._subarray / ..._indexed_block**
  - new routines due to incorrect Fortran binding in MPI-1:
    - **INTEGER (KIND=MPI_ADDRESS_KIND) ... in MPI-2**
  - new predefined datatypes:
    - **MPI_WCHAR, MPI_SIGNED_CHAR, MPI_UNSIGNED_LONG_LONG**
- Null values:
  - MPI_Init(NULL,NULL)
  - MPI_STATUS(ES)_IGNORE instead of (&)status

Other MPI-2 Features (2)

- C/C++/Fortran Language interoperability support
  - between languages in same processes
  - messages transferred from one language to another
- (P)MPI_Wtime and ..._Wtick may be implemented as macros in C
- New values VERSION=2, SUBVERSION=0

Deprecated Names/Functions

Deprecated	MPI-2 Replacement
MPI_ADDRESS	MPI_GET_ADDRESS
MPI_TYPE_HINDEXED	MPI_TYPE_CREATE_HINDEXED
MPI_TYPE_HVECTOR	MPI_TYPE_CREATE_HVECTOR
MPI_TYPE_STRUCT	MPI_TYPE_CREATE_STRUCT
MPI_TYPE_EXTENT	MPI_TYPE_GET_EXTENT
MPI_TYPE_UB	MPI_TYPE_GET_EXTENT
MPI_TYPE_LB	MPI_TYPE_GET_EXTENT
MPI_UB	MPI_TYPE_CREATE_RESIZED
MPI_LB	MPI_TYPE_CREATE_RESIZED
…….	…….
…….	…….

Motivation: Incorrect Fortran interface for Aint

Process Startup

- Current MPI implementations provide the “mpirun” as a startup command which is not standard and not portable
- MPI-2 specifies an “mpiexec” startup command (recommended)

```
mpiexec  {-n <maxprocs>  -soft <      >
          -host <      >  -arch <      >
          -wdir <      >  -path <      >
          -file <      >  ……. Command
         } : { …… } : { …… }
```

C++ Language Bindings

- C++ bindings match the new C bindings
- MPI objects are C++ objects
- MPI functions are methods of C++ classes
- User must use MPI create and free functions instead of default constructors and destructors
- Uses shallow copy semantics
  (except MPI::Status objects)
- C++ exceptions used instead of returning error code
- declared within an MPI namespace (**`MPI::...`**)
- C++/C mixed-language interoperability

Fortran 90 Support, I.

- MPI-2 Fortran bindings are Fortran 90 bindings that are “Fortran 77 friendly” (most cases)
- Fortran 90 and MPI have several incompatibilities
  - strong typing vs. choice arguments
  - data copying vs. sequence association
  - special values vs. special constants
  - …..
- MPI-2 standard documents the “do’s” and “don’ts” while using Fortran 90 / 77 features
  - *Chap. 10.2.2* “Problems with Fortran Bindings for MPI”

Problems Due to Data Copying and Sequence Association

- Example 1:

```
real a(100)
call MPI_Irecv( a(1:100:2), MPI_REAL, 50, ...)
```

  - First dummy argument of MPI_Irecv is an assumed-size array (`<type> buf(*)`)
  - a(1:100:2) is copied into a scratch array, which is freed after the end of MPI_Irecv
  - Afterwards, until MPI_Wait, there is no chance to store the results into a(1:100:2)

Problems Due to Data Copying and Sequence Association

- Example 2:

```
real a
call user1(a,rq)
call MPI_Wait(rq,status,ierr)
write (*,*) a
subroutine user1(buf,request)
call MPI_Irecv(buf, ..., request, ...)
end
```

  - the compiler has to guarantee, that it makes no copy of the <u>scalar</u> a
    - **neither in the calling procedure**
    - **nor in the called procedure**
  - check for compiler flags!
  - guarantee is necessary for
    - **MPI_Get_address**
    - **all non-blocking MPI routines**

Fortran Problems with Register Optimization and 1-sided

Source of Process 1	Source of Process 2	Executed in Process 2
bbbb = 777	buff = 999	register_A := 999
call MPI_WIN_FENCE	call MPI_WIN_FENCE	
call MPI_PUT(bbbb into buff of process 2)		stop application thread buff := 777 in PUT handler continue application thread
call MPI_WIN_FENCE	call MPI_WIN_FENCE ccc = buff	ccc := register_A

- Fortran register optimization
- Result ccc=999, but expected ccc=777
- How to avoid: (see MPI-2, Chap. 6.7.3)
  - window memory declared in COMMON blocks
    i.e. MPI_ALLOC_MEM cannot be used
  - declare window memory as VOLATILE
    (non-standard, disables compiler optimization)
  - Calling MPI_Address(buff, idummy_addr, ierror) after 2nd FENCE in process 2

Fortran 90 Support, II.

- Different support levels:
  - Basic Fortran support
    - **mpif.h is valid with free-form and fixed-form**
  - Extended Fortran support
    - **an `mpi` module: `USE mpi` instead of `include mpif.h`**
    - **datatype generation routines for KIND-parameterized Fortran types:**
      - **`MPI_TYPE_CREATE_F90_INTEGER`**
      - **`MPI_TYPE_CREATE_F90_REAL`**
      - **`MPI_TYPE_CREATE_F90_COMPLEX`**
    - **alternative concept [*not appropriate for heterogeneous platforms*]:**
      - **`MPI_SIZEOF`, `MPI_TYPE_MATCH_SIZE`**

Summary of MPI-2

- MPI-2 standard document available since July 1997
- Provides extensions to MPI-1, does not replace MPI-1
- Provides a wide variety of functionality, some still untested
- Implementation of parts of the MPI-2 standard are available
- Nearly full implementations were available by mid-2000 on
  - Fujitsu
  - NEC
  - Hitachi
- Subset examples:
  - Cray: mpt.1.4.0.1 (MPI-I/O [romio], one-sided)
  - MPICH
  - LAM: with dynamic process start
- List of MPI-2 implementations and subsets: www.mpi.nd.edu/MPI2/

Other MPI Activities

- Metacomputing
  - MPI-PACX, MetaMPI, HARNESS, Globus, StaMPI, ...
  - IMPI (Interoperable Massage Passing Interface)
- Real-time MPI
- Additional MPI Language Bindings
  - Java
  - Ada-95
  - COM
- The DARPA Data Reorganization Standard
- MPI on PC clusters

(see www.hlrs.de/mpi/)