Workshop Parallel Programming

MPI–2

Rolf Rabenseifner
28.1.1998

RUS
Rechenzentrum
Universität
Stuttgart

MPI−2 Documents

The MPI−2 Forum has released

- MPI−2: Extensions to the Message−Passing Interface (July 18, 1997)

  It includes:

  - **MPI 1.2** with
    - minor additions, e.g. new data types
    - clarifications, e.g. MPI_Finalize
  - **MPI 2.0** with
    - new features,
    - e.g. dynamic process start,
      MPI−I/O

- MPI−2 Journal of Development (July 18, 1997)
  - not mandatory functions
  - ideas for MPI 3

General rules:

- No modifications to MPI 1.1
- Only additional functionality and clarifications

RUS
Rechenzentrum
Universität
Stuttgart

MPI 1 — the Basis

Functionality

- point−to−point communication
- collective communication
- process groups & communicators
- process topologies
- environmental management
  - implementation information
  - error handling
- language binding
  - C
  - Fortran 77

Actual release

- MPI 1.1 — June 12, 1995

RUS
Rechenzentrum
Universität
Stuttgart

New MPI–2 functionality

- dynamic process creation and management
- one sided communication
- extended Collective Operations
- external interfaces
- I/O
- miscellany
- C++ binding
- Fortran 90 binding

Dynamic Process Creation and Management

Spawning Processes

spawning **new additional** MPI processes

- new group with own MPI_COMM_WORLD
- intercommunicator between old and new

Interfaces for new additional MPI processes

- in the parent:

```
MPI_COMM_SPAWN(command, argv,maxprocs,
    info, root, comm,
    intercomm, array_of_errcodes)
```

```
MPI_COMM_SPAWN_MULTIPLE(count,
    array_of_commands, array_of_argv,
    array_of_maxprocs, array_of_info,
    root, comm,
    intercomm, array_of_errcodes)
```

- in the child:

```
MPI_Init(...)
```

```
MPI_COMM_GET_PARENT(intercomm)
```

(*output arguments in* ***bold italic***)

Dynamic Process Creation and Management (cont.)

Establishing Communication between MPI processes

- between 2 sets of MPI processes
  that do not share an existing communicator
- server: MPI_OPEN_PORT &
  MPI_COMM_ACCEPT
  client: MPI_COMM_CONNECT
- Simple name service
- Releasing Connections

Interfaces (*output arguments in **bold italic***)

MPI_OPEN_PORT (info, ***port_name***)

MPI_COMM_ACCEPT (port_name, info, root,
comm, ***newcomm***)

MPI_COMM_CONNECT(port_name, info, root,
comm, ***newcomm***)

MPI_PUBLISH_NAME (service_name, info,
port_name)

MPI_LOOKUP_NAME (service_name, info,
port_name)

MPI_COMM_DISCONNECT (comm)

RUS
Rechenzentrum
Universität
Stuttgart

Dynamic Process Creation and Management (cont.)

Another way to establish MPI communications

`MPI_COMM_JOIN(fd, `***`intercomm`***`)`

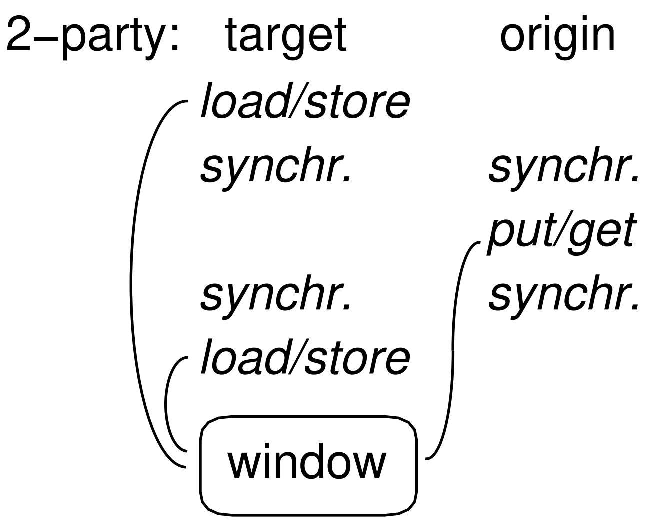
- joins by an intercommunicator
- two independent MPI processes
- that are connected with Berkeley Sockets of type SOCK_STREAM

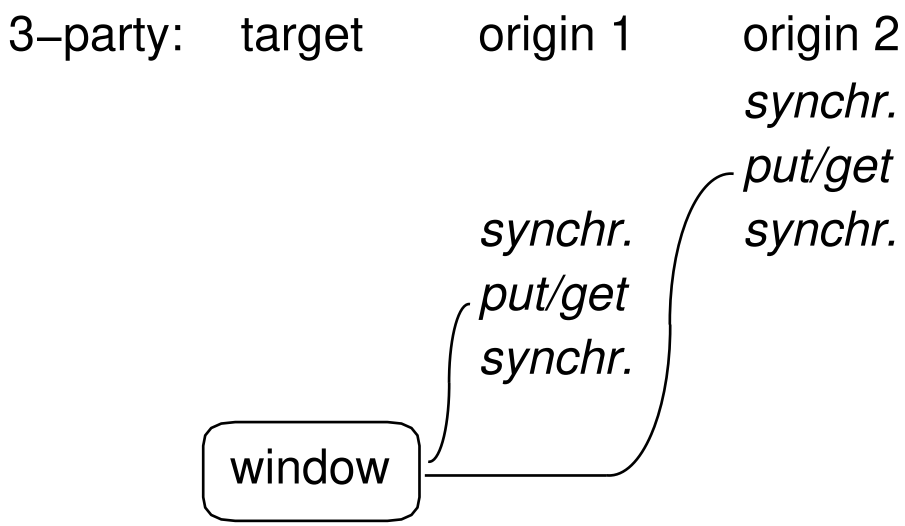
Other routines

- predefined attribute MPI_UNIVERSE_SIZE (of MPI_COMM_WORLD) indicates the total number of processes that are expected
- Singleton MPI_INIT
  - in high–quality impl., but not mandatory
  - allows start of MPI applications without mpirun
- attribute MPI_APPNUM is the command–line number from MPI_SPAWN_MULTIPLE

RUS
Rechenzentrum
Universität
Stuttgart

One Sided Communications





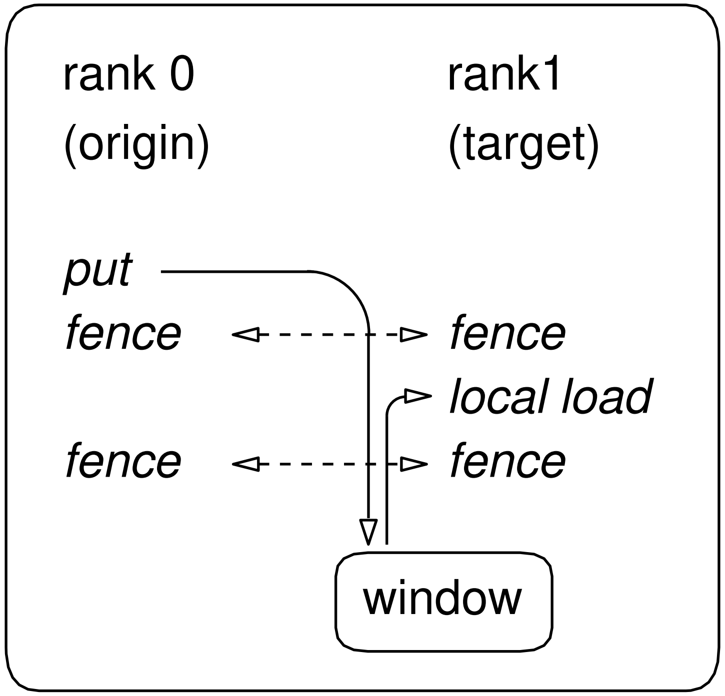
Principles for Remote Memory Access (RMA)

- origin process specifies all origin **and** target arguments
- separation of
  - communication
    MPI_GET = remote read
    MPI_PUT = remote write
    MPI_ACCUMULATE = remote update
    from/to RMA windows
  - synchronization
    fence (barrier)
    post/start/complete/wait for 2–party RMA
    lock/unlock (shared/excl.) for 3–party RMA
- cache coherence functions are called automatic.
- special window allocation not mandatory (except for lock/unlock) but necessary for efficiency
- Progress: RMA calls should not block, except for synchronizing with barrier, start, wait and lock

RUS
Rechenzentrum
Universität
Stuttgart

One Sided Communications (Cont.)

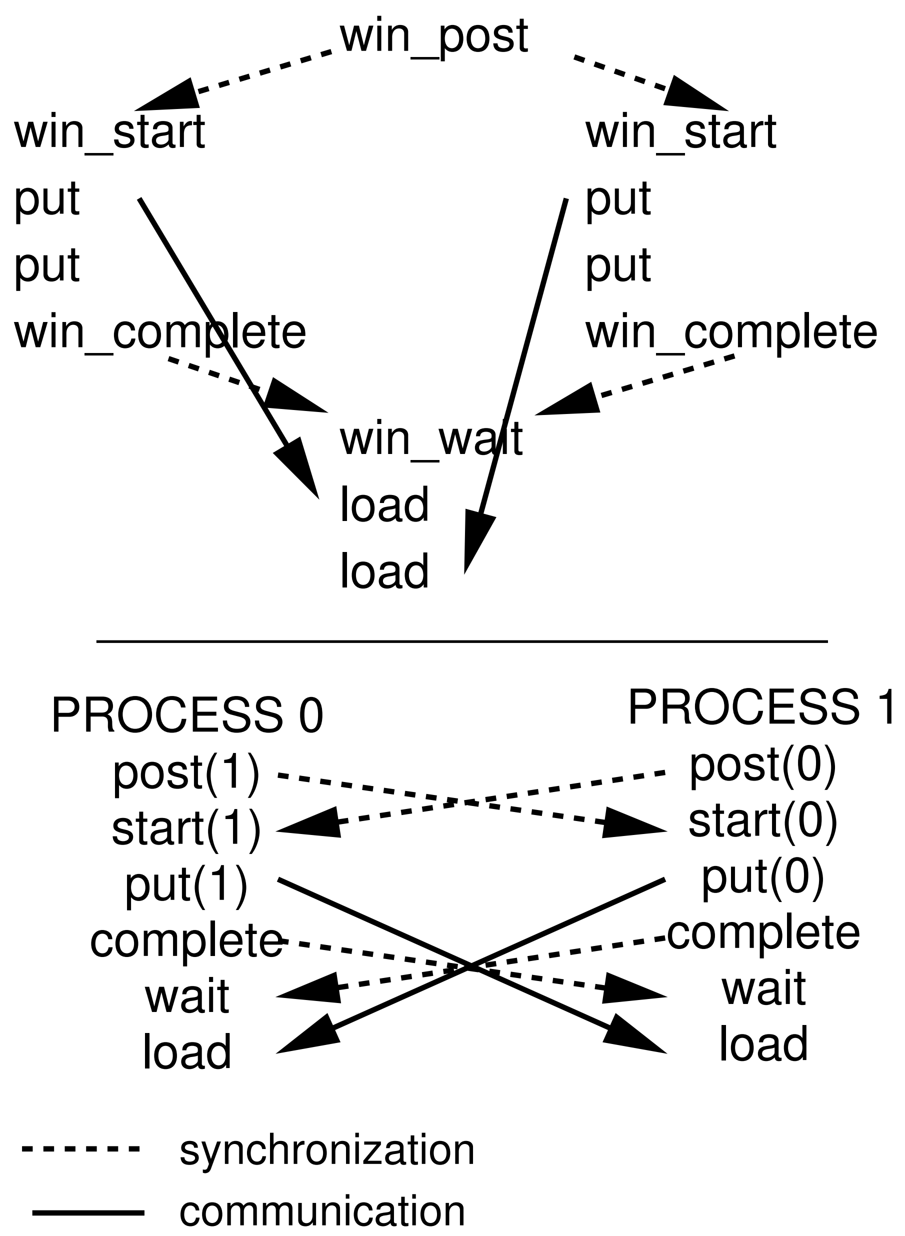
Example with fence (barrier) synchronization



(*output arguments in **bold italic***)

```
float *base;
MPI_Alloc_mem(100*sizeof(float), info,
                 comm, base);
MPI_Win_create(base,100*sizeof(float),
     sizeof(float), info, comm, win);
if (myrank == 0)
  MPI_Put(buf,70,MPI_FLOAT, /*origin*/
          1,30,70,MPI_FLOAT,/*target*/
          win);
MPI_Win_fence(asserts, win);
if (myrank == 1)
   consuming the data in the window:
      base[30] ... base[99];
MPI_Win_fence(asserts, win);
```

One Sided Communications (Cont.)



2–Party Synchronization

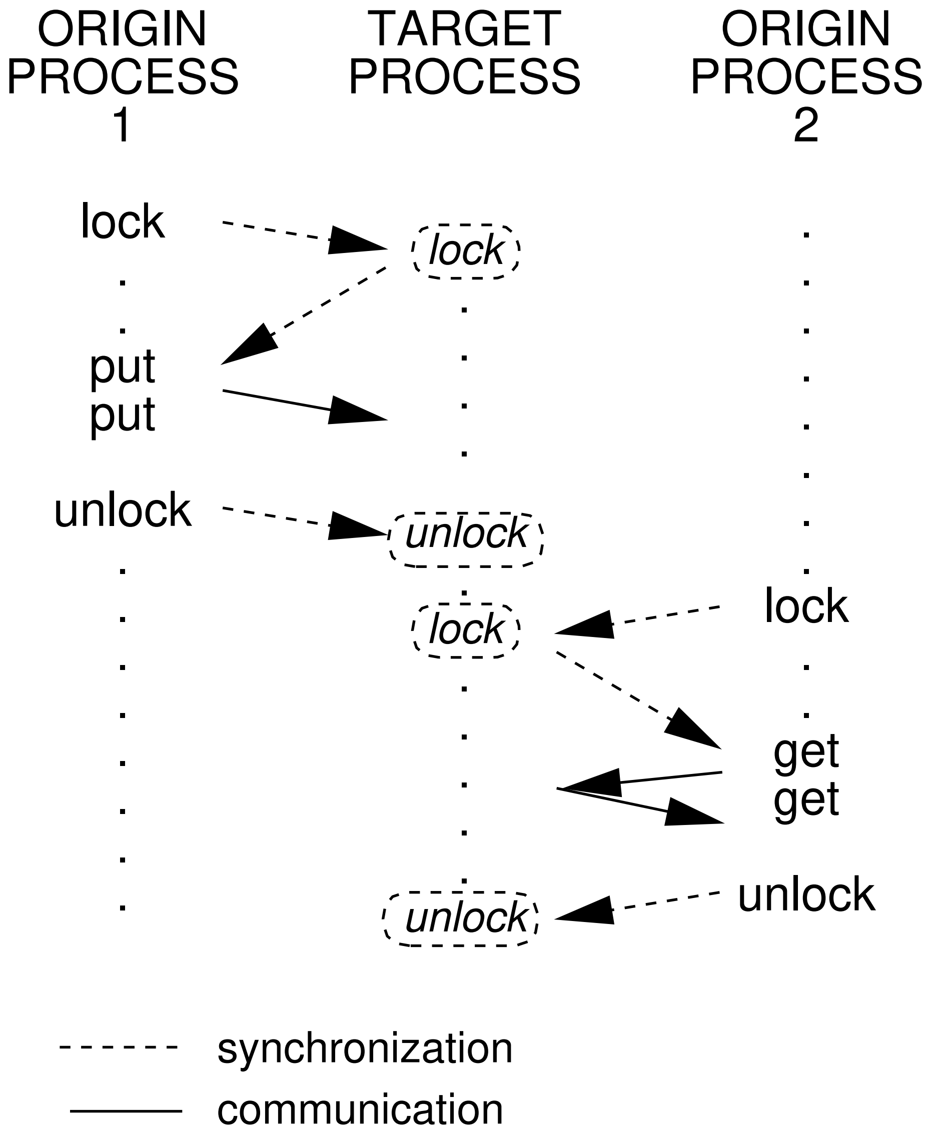
- Put, Get, Accumulate are finished
  - locally after Win_complete, and
  - at the target after Win_wait
- local buffer should not be reused before RMA call locally finished
- communication partners must be known
- no atomicity for overlapping puts
- symmetric communication possible
- assertions may improve efficiency
- allows max. efficiency with double buffering applications (example 6.10)

Interface:

```
MPI_WIN_POST (group, asserts, win)
MPI_WIN_START(group, asserts, win)
MPI_WIN_COMPLETE(win)
MPI_WIN_WAIT(win)
```


RUS Rechenzentrum Universität Stuttgart

One Sided Communications (Cont.)



3–Party Synchronization

- Does not guarantee a needed sequence
- Window should be allocated with MPI_ALLOC_MEM
- agent necessary on systems without (virtual) shared memory

RUS
Rechenzentrum
Universität
Stuttgart

Extended Collective Operations

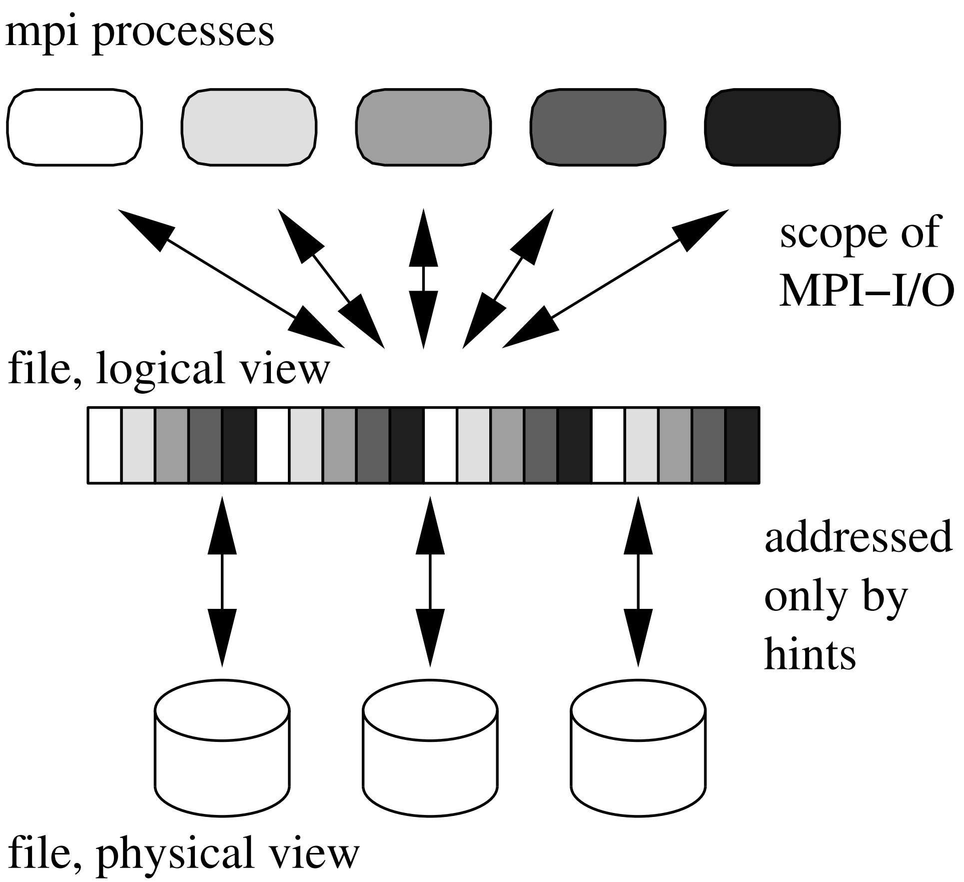
- New **inter**communicator Constructors:
  - MPI_COMM_CREATE
  - MPI_COMM_SPLIT
- Extension of all MPI collective operations (except MPI_SCAN) to **inter**communicators
- The `sendbuf` may be equivalent to the recvbuf in
  `(ALL)GATHER(V), (ALL)SCATTER(V),`
  `(ALL)REDUCE,REDUCE_SCATTER, SCAN`
  by specifiying `MPI_IN_PLACE` as `sendbuf`
  e.g. `MPI_SCATTER(MPI_IN_PLACE,...,`
  `send_recv_buf,...)`
- new `MPI_ALLTOALLW, MPI_EXSCAN`

External Interfaces

Mainly **for library writers** and for implementing **MPI–I/O** on top of MPI 1.1 with MPI–2 Misc and External

- Generalized Requests
- Associating Information with Status
- Naming Objects (`MPI_COMM_SET_NAME/_GET_NAME, ..._TYPE_..., ..._WIN_...`)
- Allowing User Errors (`MPI_COMM_CALL_ERRHANDLER, ..._WIN_..., ..._FILE_...`)
- Decoding MPI Datatypes
- MPI and Threads
- Attribute Caching on MPI Handles (`COMM, TYPE, WIN`)
  - old routine names are deprecated
  - Fortran binding: `attr_val` is address–sized integer

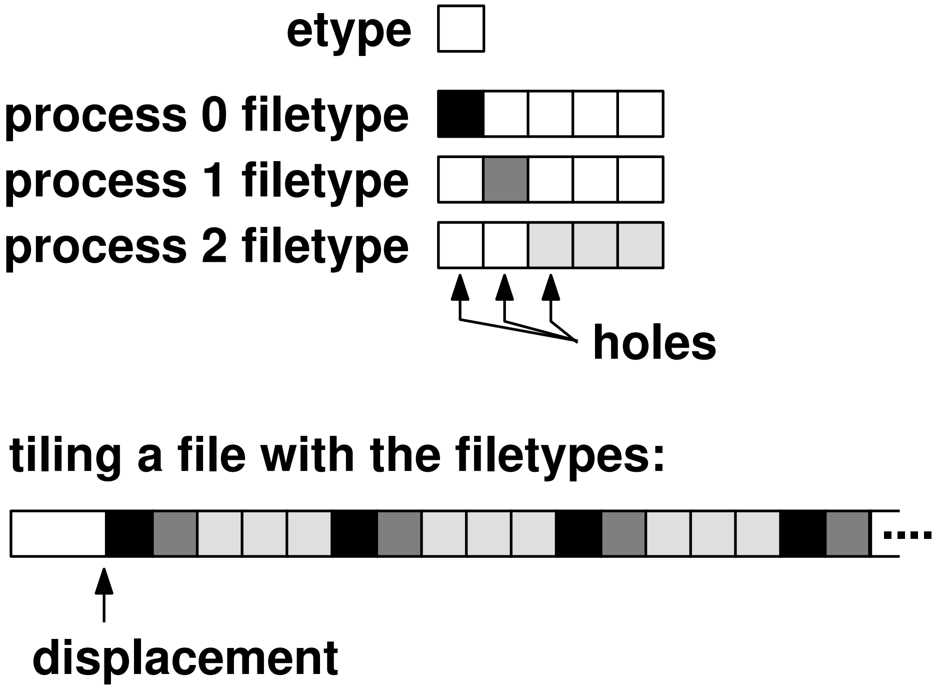
MPI – I/O



- Files are opened by the MPI processes of a communicator
- logical view / physical view
- 4 orthogonal aspects = 24 functions
  - access – read
    – write
  - positioning – explicit offsets
    – individual file pointers
    – shared file pointers
  - synchronism – blocking
    – nonblocking or split coll.
  - coordination – noncollective
    – collective

RUS
Rechenzentrum
Universität
Stuttgart

MPI–I/O (cont.)



etype:

elementary datatype,
is the unit of data access and positioning,
it can be any MPI predefined or derived datatype.

filetype:

the basis for partitioning a file among processes,
defines a template for accessing the file.

displacement:

number of header bytes

view:

each process has its own view, defined by:
a displacement, an etype, and a filetype.
The filetype is repeated, beginning at the displacement.

RUS
Rechenzentrum
Universität
Stuttgart

MPI 1.2 Miscellany

- Version Number
  - o compile–time:
    ```
    #define MPI_VERSION    1
    #define MPI_SUBVERSION 2
    PARAMETER (MPI_VERSION    = 1) ...
    const int MPI::VERSION = 1      ...
    ```
  - o runtime:
    `MPI_GET_VERSION(`***`version,subversion`***`)`
- Clarifications to `INITIALIZED`, `FINALIZED`, `WAIT/TEST` after `ISEND`, `INTERCOMM_CREATE`, `INTERCOMM_MERGE`, `TYPE_SIZE`, `REDUCE` , attribute callback functions, `PROBE/IPROBE`
- Fortran Problem with Register Optimization, an *Advice to users*
- Minor Corrections (Typos)

RUS
Rechenzentrum
Universität
Stuttgart

MPI−2 Miscellany

- Portable MPI Process Startup: `mpiexec`
  - similar to existing `mpirun`
  - same functionality as `MPI_SPAWN(_MULTIPLE)`
- `MPI_Init(NULL,NULL)` also allowed
- Version 2, subversion 0
- New constructor `MPI_TYPE_CREATE_INDEXED_BLOCK`
- Treatment of MPI_Status
  - `MPI_STATUS_IGNORE` for `status` arguments
  - Non−destructive Test of MPI_Status:
    `MPI_REQUEST_GET_STATUS(req,flag,status)`
- Error Class for Invalid Keyval: `MPI_ERR_KEYVAL`
- User function at process termination (via attributes and delete callback function on `MPI_COMM_SELF`)
- MPI_FINALIZED(***flag***)
- Info object handling (for SPAWN, 1−SIDED, MPI−I/O)
- Language Interoperability (Fortran 77 & 90 **with** C & C++)
  - 32/64 bit `MPI_ADDRESS` Fortran problem:
    datatype INTEGER = 32, but addresses = 64 bits.
    Solutions
    - Constant `MPI_ADDRESS_KIND`
    - Additional functions `MPI_GET_ADDRESS`, ...
  - Transfer of handles with explicit wrappers

MPI–2 Miscellany (cont.)

- New error handler interface for `COMM, WIN, FILE`
- New type constructors with explicit addresses:

  `MPI_TYPE_CREATE_HVECTOR`
  `MPI_TYPE_CREATE_HINDEXED`
  `MPI_TYPE_CREATE_STRUCT`

  with `INTEGER(KIND=MPI_ADDRESS_KIND)`
- Extent and bounds of datatypes:

  `MPI_TYPE_UB`, `MPI_TYPE_LB` and `MPI_TYPE_EXTENT`
  are replaced by
  `MPI_TYPE_CREATE_RESIZED` + `MPI_TYPE_GET_EXTENT`
- True extent: `MPI_TYPE_GET_TRUE_EXTENT`
- New datatype constructors:
  subarray: `MPI_TYPE_CREATE_SUBARRAY`,
  distributed array: `MPI_TYPE_CREATE_DARRAY`
- New Datatypes: `MPI_WCHAR`,
  `MPI_SIGNED_CHAR` (not translated as character),
  `MPI_UNSIGNED_LONG_LONG`
- Canonical data packing:
  `MPI_PACK_EXTERNAL, MPI_UNPACK_EXTERNAL,`
  `MPI_PACK_EXTERNAL_SIZE`
- **Change to MPI 1.2:**
  `MPI_WTIME`, `MPI_WTICK` may be C macros.

RUS
Rechenzentrum
Universität
Stuttgart

C++ Binding

Principles

- small set of classes with a lightweight functional interface to MPI
- classes = fundamental MPI object types
- one–to–one mapping between MPI functions and their C++ bindings
- declared within an `MPI` namespace (`MPI::...`)
- C++ / C mixed–language interoperability

Classes

```
namespace MPI {
 class Comm {...};  class Datatype  {...};
 class Group{...};  class Errhandler{...};
 class Op   {...};  class Request   {...};
 class Status{...}; class Exception {...};
};
```

error handling with exceptions

Constants

```
MPI::SUCCESS  ...
```

Typdefs

```
MPI::Aint
```

Functions

```
namespace MPI {
  void Comm::Send(const void* buf,int count,
        const Datatype& datatype,...) const
  void Init (int& argc, char**& argv)
  void Init ()
  ...
```

RUS
Rechenzentrum
Universität
Stuttgart

Fortran 90 Binding

Principles

- Fortran 77 MPI bindings for Fortran 90 programs
- Fortran 90 module instead of `mpif.h`
- Require that **choice arguments** not cause Fortran 90 compilation or linkage errors
- Coexistence of Fortran 77 and Fortran 90 MPI programs
- Explain carefully the problems that users might encounter when trying to use Fortran 90 with MPI, **also important for MPI 1.1 with Fortran 77**
- Different support levels:
  - Basic Fortran support:
    - mpif.h is always valid
  - Extended Fortran support:
    - an `mpi` module
    - datatype generation routines for KIND−parameterized Fortran types:
      `MPI_TYPE_CREATE_F90_INTEGER`
      `MPI_TYPE_CREATE_F90_REAL`
      `MPI_TYPE_CREATE_F90_COMPLEX`
    - `MPI_SIZEOF, MPI_TYPE_MATCH_SIZE`

RUS
Rechenzentrum
Universität
Stuttgart

Fortran 90 Binding

(cont.)

The big problem:

MPI datatypes descibe memory layout of user data

Fortran 90 does not define the memory layout and data need not be statically in the memory

In detail:

- `MPI_ADDRESS` senseless
- `MPI_I...`:
  actual buffer argument can be a temporary copy
- Array sections

```
real x(200)
call bar(x(1:200:2))
...
subroutine bar(x)
real x(:)
call MPI_Irecv(x, 100, MPI_Real,
                                ... , request)
...
call MPI_Wait(request)
```

- Derived datatypes – memory layout unknown

Comments

I/O This will be the major chapter but when shall we get high−performance implementations?

Fortran please read the problem−section although you use only Fortran 77 subset

1−sided never better than point−to−point messages?

spawn does it help any PVM user to come to MPI?

C++ okay?

misc the useful `MPI_TYPE_SIMPLE_STRUCT` went to the Journal of Development (JOD)

colletive nonblocking "split" interface only for I/O routines!

Thread for coexistence of OpenMP & MPI enough?

Interoperability between different MPI implementations is not part of the standard !!

Further information:

http://www.hlrs.de/structure/support/models/mpi/

http://www.hlrs.de/structure/support/models/mpi/mpi−20.ps.gz

http://www.hlrs.de/structure/support/models/mpi/mpi−20.ps.Z