MPI-3.0 Overview

Rolf Rabenseifner

Acknowledgements

- Some detailed slides are provided by the
  - Ticket authors,
  - Chapter authors, or
  - Chapter working groups.
- Richard Graham, chair of MPI-3.0.
- Torsten Hoefler (additional example about new one-sided interfaces)

Goal & Scope of MPI-3.0

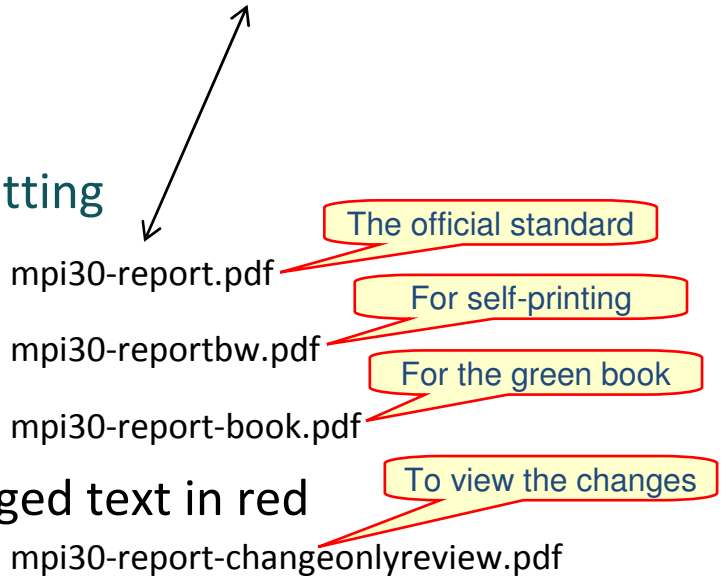
- Goal:
  - To produce new versions of the MPI standard that better serves the needs of the parallel computing user community
- Scope:
  - Additions to the standard that are needed for better **platform** and **application** support.
  - These are to be consistent with MPI being a library providing process group management and data exchange. This includes, but is not limited to, issues associated with scalability (performance and robustness), multi-core support, cluster support, and application support.
  - And of course,
    all needed corrections to detected bugs / ambiguities / inconsistencies
  - Backwards compatibility may be maintained —
    Routines may be deprecated or deleted.

MPI-3.0 – the pdf files

- www.mpi-forum.org
  - → MPI documents
    - → MPI 3.0 document as PDF

      http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
    - → Hardcover (green book)

      http://www.hlrs.de/mpi/mpi30/
    - → Versions of MPI 3.0 with alternate formatting
      - Official MPI 3.0 standard
      - MPI 3.0 standard (black and white)
      - MPI 3.0 standard (book version)
      - MPI 3.0 standard with new or changed text in red



Improved Collective Communication

- Scalable sparse collectives on process topologies
  - MPI_(I)Neighbor_allgather(v) / alltoall(v/w):
  - MPI_Aint displs in (I)Neighbor_alltoallw (instead of int)

→Slide 14

21

21

- Nonblocking collectives
  - Communication: MPI_Ibarrier, MPI_Ibcast, …
  - MPI_Icomm_dup

→Slide 15

13

15

Background information, see:

- #Item in MPI-3.0, Change-Log, Annex B.1.1 (E1-E7) and B.1.2 (1-42)
  The item numbers in this document are based on MPI-3.0, Sep. 21, 2012:

Improvements to one-sided communication support

New features

→Slides 16-21

- **Routines: MPI_Rget / Rput / Raccumulate / Get_accumulate / Rget_accumulate / Fetch_and_op / Compare_and_swap MPI_Win_allocate / create_dynamic / attach / detach / lock_all / unlock_all / flush / flush_all / flush_local / flush_local_all / sync**
- **New operation MPI_NO_OP**
- **New attribute** MPI_WIN_CREATE_FLAVOR **= MPI_WIN_FLAVOR_CREATE / ALLOCATE / DYNAMIC**
- **New attribute** MPI_WIN_MODEL **= MPI_WIN_SEPARATE / UNIFIED**

- Shared memory RMA window (see also slide on “hybrid” →Slide 7) 24
  - **Routine: MPI_Win_allocate_shared, MPI_Win_shared_query** →Slides 21-23
  - **Flavor attribute: MPI_WIN_FLAVOR_SHARED**

Support for clusters of SMP nodes (hybrid support)

- Topology aware communicator creation →Slides 21+23 17
  - **MPI_Comm_split_type with split_type=MPI_COMM_TYPE_SHARED**
- Shared memory RMA window (see also slide on "one-sided" →Slide 6) 24
  - **Routine: MPI_Win_allocate_shared, MPI_Win_shared_query** →Slides 21+23
  - **Flavor attribute: MPI_WIN_FLAVOR_SHARED**
- Thread-safe probe → new probe routine and message object
  - **MPI_(I)Mprobe, MPI_(I)Mrecv** →Slide 25 11
  - **MPI_Message_f2c / c2f** 11

Other Forum Activities

- New Fortran bindings: mpi_f08 module →Slides 26-38 26-42
- Non-collective communicator formation: MPI_Comm_create_group 16
  - Tags are in a special tag-space.
  - MPI_INTERCOMM_CREATE has its own tag-space →Slide 39 18
- New Datatype Creation Routine: MPI_Type_create_hindexed_block →Slide 40 12
- Large counts: 6,8
  - type MPI_Count / INTEGER(KIND=MPI_COUNT_KIND) →Slide 41
  - datatype MPI_COUNT
  - New routines: MPI_Type_: size_x / get_extent_x / get_true_extent_x, MPI_Get_elements_x, MPI_Status_set_elements_x
- MPI_Init, MPI_Init_thread, and MPI_Finalize were clarified. 22
  - New predefined info object **MPI_INFO_ENV** holds arguments from mpiexec or MPI_COMM_SPAWN

New Tool Interface

- Main ticket 25

→Slides 42-47

  - **MPI_T_init_thread / finalize / enum_get_info/item,**
    MPI_T_cvar_: get_num/info / handle_alloc/free / read / write,
    MPI_T_pvar_: get_num/info / session_create/free / handle_alloc/free / start / stop / read / write / reset / readreset,
    MPI_T_category_: get_num/info/cvars/pvars/categories / changed
  - **MPI_T routines are callable before MPI_Init & after MPI_Finalize** 23
  - **MPI_Get_library_version** 5
  - **MPI_Comm_dup_with_info /set_info / get_info, MPI_Win_set/get_info**
    - **MPI_COMM_DUP must also duplicate info hints.** 14
- MPIR (independent document, not part of the MPI standard) ---
  - **"The MPIR Process Acquisition Interface"**
  - a commonly implemented interface
    primarily used by debuggers to interface to MPI parallel programs

Deprecated functionality *moved* to Chapter 16. Removed Interfaces

- Removed MPI-1.1 functionality (deprecated since MPI-2.0): 1
  - Routines: MPI_ADDRESS, MPI_ERRHANDLER_CREATE / GET / SET, MPI_TYPE_EXTENT / HINDEXED / HVECTOR / STRUCT / LB / UB
  - Datatypes: MPI_LB / UB
  - Constants MPI_COMBINER_ HINDEXED/HVECTOR/STRUCT _INTEGER
  - Removing deprecated functions from the examples and definition of MPI_TYPE_GET_EXTENT
- C++ Bindings are removed (deprecated since MPI-2.2) →Slide 47 2
  - C++ applications are supported through the MPI C language binding
  - Special C++ types are supported through additional MPI predefined datatypes E1
    - MPI_CXX_BOOL — bool
    - MPI_CXX_FLOAT_COMPLEX — std::complex<float>
    - MPI_CXX_DOUBLE_COMPLEX — std::complex<double>
    - MPI_CXX_LONG_DOUBLE_COMPLEX — std::complex<long double>

Minor Corrections and Clarifications

- Consistent use of [] for input and output arrays 7
  - Exception: MPI_INIT and MPI_INIT_THREAD: char ***argv
- Add const keyword to the C bindings. “IN” was clarified. 3
- MPI_STATUSES_IGNORE can be used in MPI_(I)(M)PROBE 9
- MPI_PROC_NULL behavior for MPI_PROBE and MPI_IPROBE 10
- MPI_UNWEIGHTED should not be NULL 4
- MPI_Cart_map with num_dims=0 20
- MPI_MAX_OBJECT_NAME used in MPI_Type/win_get_name 19
- New wording in reductions:
  Multi-language types MPI_AINT, MPI_OFFSET, MPI_COUNT ---
- MPI_TYPE_CREATE_RESIZED should be used for “arrays of struct” 32
  - The MPI alignment rule cannot guarantee to calculate the same alignments as the compiler

Six MPI-2.2 errata items, see Change-Log Annex B.1.1

- C++ corrections were added as MPI-2.2 errata, because MPI-2.2 is the last version that includes the MPI C++ language binding.
  - **MPI_CXX_BOOL** and **MPI_CXX_ FLOAT/DOUBLE/LONG_DOUBLE _COMPLEX** are added to the list of predefined datatype handles (in C and Fortran) E2
  - The nonstandard C++ types Complex<...> were substituted by the standard types **std::complex<...>**. E2
  - Minor corrections, e.g., MPI name-space is completely reserved E1, E4, E6, E7
- MPI_C_COMPLEX was added to the *Complex* reduction group E3
- The MPI_C_BOOL "external32" representation is 1-byte E5

The MPI-3.0 Change-Log, Annex B.1.2 has two parts:

- Items 1-25: General changes
- Times 26-42: Changes related to Fortran

Details about most & important topics

nn

Background information, see:
#Item in MPI-3.0, Change-Log, B.1.2 (1-42)
The item numbers in this document are based on MPI-3.0. Sep. 21, 2012.

- Slide 14: Sparse and scalable irregular collectives
- Slide 15: Nonblocking collectives
- Slides 17-20: One-sided communication – enhancements
- Slides 21-23: Shared memory extensions (on clusters of SMP nodes)
- Slide 25: Mprobe for hybrid programming on clusters of SMP nodes
- Slides 26-38: Fortran interface
- Slide 39: Group-Collective Communicator Creation
- Slide 40: MPI_TYPE_CREATE_HINDEXED_BLOCK
- Slide 41: Large Counts
- Slides 42-46: New tools interface
- Slide 47: Removing C++ bindings from the Standard

Sparse Collective Operations on Process Topologies

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
  - MPI_NEIGHBOR_ALLGATHER(V)
  - MPI_NEIGHBOR_ALLTOALL(V,W)
  - Also nonblocking variants
  - Accepted for MPI-3 in Sept. 2011
  - Reference implementation exists
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
  - Exception: s/rdispls in MPI_NEIGHBOR_ALLTOALLW are MPI_Aint
- Allow for optimized communication scheduling and scalable resource binding

Courtesy of Torsten Hoefler and Richard Graham

Nonblocking Collective Communication and MPI_ICOMM_DUP

- Idea
  - Collective initiation and completion separated
  - Offers opportunity to overlap computation and communication
  - Each blocking collective operation has a corresponding nonblocking operation
  - May have multiple outstanding collective communications on the same communicator
  - Ordered initialization
- Voted into the MPI-3.0 draft standard in June 2009,
  - Reference Implementation (LibNBC) stable
  - Several implementations pending
- Parallel MPI I/O: The split collective interface may be substituted in the next version of MPI

Courtesy of Torsten Hoefler and Richard Graham

MPI One-Sided Communication Interface

24

Courtesy of the MPI-3 One-sided working group

Background of MPI-2 One-Sided Communication

- MPI-2's one-sided communication provides a programming model for put/get/update programming that can be implemented on a wide variety of systems
- The "public/private" memory model is suitable for systems without local memory coherence (e.g., special memory in the network; separate, non-coherent caches between actors working together to implement MPI One-Sided)
- The MPI-2 interface, however, does not support some other common one-sided programming models well, which needs to be fixed
- Good features of the MPI-2 one-sided interface should be preserved, such as
  - Nonblocking RMA operations to allow for overlap of communication with other operations
  - Support for non-cache-coherent and heterogeneous environments
  - Transfers of noncontiguous data, including strided (vector) and scatter/gather
  - Scalable completion (a single call for a group of processes)

Goals for the MPI-3 One-Sided Interface

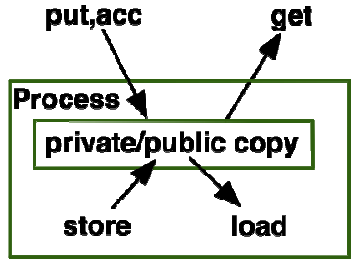
- Address the limitations of MPI-2 RMA by supporting the following features:
  - In order to support RMA to arbitrary locations, no constraints on memory, such as symmetric allocation or collective window creation, should be required
  - RMA operations that are imprecise (such as access to overlapping storage) must be permitted, even if the behavior is undefined
  - The required level of consistency, atomicity, and completeness should be flexible
  - Read-modify-write and compare-and-swap operations are needed for efficient algorithms

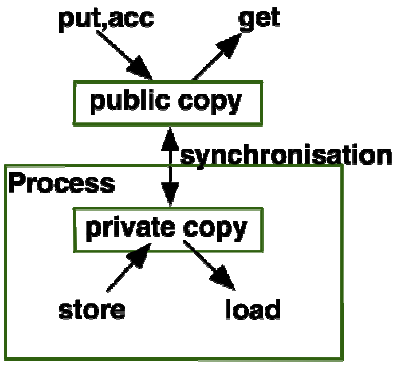
Major New Features in the MPI-3 One-sided Interface

- New types of windows (MPI-2 had only MPI_Win_create)
  - MPI_Win_allocate – returns memory allocated by MPI; permits symmetric allocation
  - MPI_Win_allocate_shared – creates a window of shared memory that enables direct load/store accesses with RMA semantics to other processes in the same shared memory domain (e.g., the same node)
  - MPI_Win_create_dynamic – allows any memory to be attached to the window dynamically as needed
- New atomic read-modify-write operations
  - MPI_Get_accumulate, MPI_Fetch_and_op, MPI_Compare_and_swap
- New synchronization and completion calls, including:
  - Wait and test on request-based one-sided operations: MPI_Rput/get/…
  - Completion of pending RMA operations within passive target access epochs (MPI_Win_flush and variants)

Major New Features – cont'd

- Query for new attribute to allow applications to tune for cache-coherent architectures
  - Attribute MPI_WIN_MODEL with values
    - MPI_WIN_UNIFIED on cache-coherent systems
    - MPI_WIN_SEPARATE otherwise





- Relaxed rules for certain access patterns
  - Results undefined rather than erroneous; matches other shared-memory and RDMA approaches
- Ordering of Accumulate operations
  - Change: ordering provided by default
  - Can be turned off for performance, using a new info key

Figures: Courtesy of Torsten Hoefler

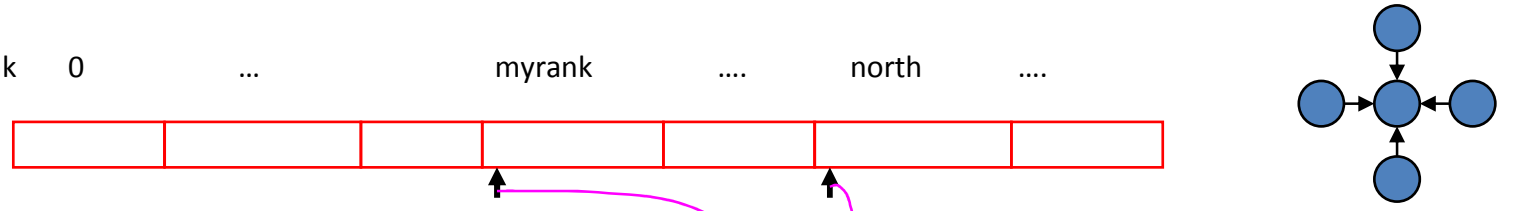
Shared Memory Extensions to MPI

- Step 1: Topology-aware communicator creation
  - Allows you to create a communicator whose processes can create a shared memory region
  - MPI_Comm_split_type(comm, comm_type, key, info, split_comm)
  - More generally: it splits a communicator into subcommunicators split_comm of a certain type:
    - MPI_COMM_TYPE_SHARED: shared memory capability
    - Other implementation specific types are possible: rack, switch, etc.
      - But not yet standardized.

Shared Memory Extensions to MPI – Cont'd

- Step 2: Creation and manipulation of shared memory (based on MPI RMA semantics)
- MPI_WIN_ALLOCATE_SHARED(size, info, split_comm, baseptr, win)
  - Collective call that allocates memory of least size bytes that is shared among all processes in comm
  - Returns locally allocated region pointed to by baseptr that can be used for load/store access on the calling process
  - "Principle of least surprise" (default)
    - Memory is consecutive across ranks
    - Allows for inter-rank address calculations
    - i.e., rank i's first Byte starts right after rank i-1's last Byte
  - Consistent view of shared memory can be created in the RMA unified memory model by using window synchronization functions or by calling MPI_WIN_FLUSH()

Five-point stencil communication in a shared memory window



Executed at myrank

collective call

```
MPI_Comm_split_type(comm, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, &shmcomm);

MPI_Win_allocate_shared(size*sizeof(double), info, shmcomm, &mem, &win);
MPI_Win_shared_query(win, north, &sz, &northptr);
// ... south, east, west directions

for(iter=0; iter<niters; ++iter) {
  MPI_Win_fence(0, win); // start new access and exposure epoch
  if(north != MPI_PROC_NULL) // the "communication"
    for(int i=0; i<bx; ++i) a2[ind(i+1,0)] = northptr[ind(i+1,by)];
  // ... south, east, west directions
  update_grid(&a1, &a2); // apply operator and swap arrays
}
```

collective call

local call

Synchronization

Direct load access to remote window portion

Example: Courtesy of Torsten Hoefler

Status of the MPI-3 One-sided Interface

- Passed final 2nd (final) vote at the January 2012 meeting
- Example implementation on top of Portals-4 available
  - thanks to Brian Barrett of Sandia National Labs
- Other implementations in progress

Thread-safe probe: MPI_(I)MPROBE & MPI_(I)MRECV

- MPI_PROBE & MPI_RECV together are not thread-safe:
  - Within one MPI process, thread A may call MPI_PROBE
  - Another tread B may steal the probed message
  - Thread A calls MPI_RECV, but may not receive the probed message
- New thread-safe interface:
  - MPI_IMPROBE(source, tag, comm, flag, message, status) or
  - MPI_MPROBE(source, tag, comm, message, status)

  together with
  - MPI_MRECV(buf, count, datatype, message, status) or
  - MPI_IMRECV(buf, count, datatype, message, request)

Message handle, e.g., stored in a thread-local variable

MPI 3.0 Fortran Bindings

- A high-level summary for non-Fortran programmers
- Details for Fortran programmers

Brief overview of the requirements for new MPI 3.0 Fortran bindings

- Requirements
  - comply with Fortran standard (for the first time)
  - enhance type safety
  - suppress argument checking for choice buffers
  - guarantee of correct asynchronous operations
  - for user convenience
    - provide users with convenient migration path
    - allow some optional arguments (e.g., ierror)
    - support sub-arrays
  - for vendor convenience
    - allow vendors to take advantage of the C interoperability standard

Slide: Courtesy of Jeff Squyres and Craig Rasmussen

Three methods of Fortran support

- USE mpi_f08 new 26
  - This is the only Fortran support method that is consistent with the Fortran standard (Fortran 2008 + TR 29113 and later).
  - This method is highly recommended for all MPI applications.
  - Mandatory compile-time argument checking & unique MPI handle types.
  - Convenient migration path.
- USE mpi
  - This Fortran support method is **inconsistent** with the Fortran standard, and its use is therefore **not recommended**.
  - It exists only for backwards compatibility.
  - new Mandatory compile-time argument checking (but all handles match with INTEGER). 39
- INCLUDE 'mpif.h' 40
  - The use of the include file mpif.h is **strongly discouraged** starting with MPI-3.0.
  - Does not guarantees compile-time argument checking.
  - Does not solve the optimization problems with nonblocking calls,
  - and is therefore inconsistent with the Fortran standard.
  - It exists only for backwards compatibility with legacy MPI applications.

The mpi_f08 Module

Mainly for implementer's reasons.
Not relevant for users.

- Example:

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

MPI_Wait(request, status, ierror) BIND(C)
    TYPE(MPI_Request), INTENT(INOUT) :: request
    TYPE(MPI_Status) :: status
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran compatible buffer declaration allows correct compiler optimizations — 28

Unique handle types allow best compile-time argument checking — 27

INTENT → Compiler-based optimizations & checking — 38

Status is now a Fortran structure, i.e., a Fortran derived type — 30

OPTIONAL ierror: MPI routine can be called without ierror argument — 29

Major changes

- Support method:

26

  USE mpi or INCLUDE 'mpif.h' → **USE mpi_f08**

- Status 30

  INTEGER, DIMENSION(MPI_STATUS_SIZE) :: status
  → **TYPE(MPI_Status) :: status**

  status(MPI_SOURCE) → **status%MPI_SOURCE**

  status(MPI_TAG) → **status%MPI_TAG**

  status(MPI_ERROR) → **status%MPI_ERROR**

Additional routines and declarations are provided for the language interoperability of the status information between see MPI-3.0, Section 17.2.5

- **C,**
- **Fortran mpi_f08, and**
- **Fortran mpi (and mpif.h)**

Major changes, continued

Same names as in C

TYPE, BIND(C) :: MPI_Comm
INTEGER :: MPI_VAL
END TYPE MPI_Comm

27

- Unique handle types, e.g.,
  - INTEGER new_comm → **TYPE(MPI_Comm) :: new_comm** (new)
- Handle comparisons, e.g.,
  - req .EQ. MPI_REQUEST_NULL → req .EQ. MPI_REQUEST_NULL

No change through overloaded operator

- Conversion in mixed applications:
  - Both modules (mpi & mpi_f08) contain the declarations for all handles.

```
SUBROUTINE a
USE mpi
INTEGER :: splitcomm
CALL MPI_COMM_SPLIT(…, splitcomm)
CALL b(splitcomm)
END
```

```
SUBROUTINE b(splitcomm)
USE mpi_f08
INTEGER :: splitcomm
TYPE(MPI_Comm) :: splitcomm_f08
CALL MPI_Send(…, MPI_Comm(splitcomm) )
! or
splitcomm_f08%MPI_VAL = splitcomm
CALL MPI_Send(…, splitcomm_f08)
END
```

```
SUBROUTINE a
USE mpi_f08
TYPE(MPI_Comm) :: splitcomm
CALL MPI_Comm_split(…, splitcomm)
CALL b(splitcomm)
END
```

```
SUBROUTINE b(splitcomm)
USE mpi
TYPE(MPI_Comm) :: splitcomm
INTEGER :: splitcomm_old
CALL MPI_SEND(…, splitcomm%MPI_VAL )
! or
splitcomm_old = splitcomm%MPI_VAL
CALL MPI_SEND(…, splitcomm_old)
END
```

Major changes, continued

- SEQUENCE **and BIND(C)** derived application types can be used as buffers in MPI operations.
- Alignment calculation of basic datatypes:
  - In MPI-2.2, it was undefined in which environment the alignments are taken.
  - There is no sentence in the standard.
  - **It may depend on compilation options!**
  - In MPI-3.0, still undefined, but recommended to use a BIND(C) environment.
  - Implication **(for C and Fortran!)**:
    - If an array of structures (in C/C++) or derived types (in Fortran) should be communicated, it is recommended that
    - the user creates a portable datatype handle and
    - applies additionally MPI_TYPE_CREATE_RESIZED to this datatype handle.

Other enhancements

- Unused ierror

  INCLUDE ‘mpif.h’

  ! wrong call:

  CALL MPI_SEND(…., MPI_COMM_WORLD)

  ! → terrible implications because ierror=0 is written somewhere to the memory

- With the new module

  USE mpi_f08

  ! Correct call, because ierror is **optional**:

  CALL MPI_SEND(…., MPI_COMM_WORLD)

Other enhancements, continued

- With the mpi & mpi_f08 module:

  new

  – Positional and **keyword-based** argument lists 33

    - CALL MPI_SEND(sndbuf, 5, MPI_REAL, right, 33, MPI_COMM_WORLD)
    - CALL MPI_SEND(**buf**=sndbuf, **count**=5, **datatype**=MPI_REAL, **dest**=right, **tag**=33, **comm**=MPI_COMM_WORLD)

    The keywords are defined in the language bindings. Same keywords for both modules.

  – Remark: Some keywords are changed since MPI-2.2 33

    - For consistency reasons, or
    - To prohibit conflicts with Fortran keywords, e.g., type, function.

Major enhancement with a full MPI-3.0 implementation

- The following features require Fortran 2003 + TR 29113
  - Subarrays may be passed to nonblocking routines

    - This feature is available if the LOGICAL compile-time constant MPI_SUBARRAYS_SUPPORTED == .TRUE.
  - Correct handling of buffers passed to nonblocking routines,

    - if the application has declared the buffer as ASYNCHRONOUS within the scope from which the nonblocking MPI routine and its MPI_Wait/Test is called,
    - and the LOGICAL compile-time constant MPI_ASYNC_PROTECTS_NONBLOCKING == .TRUE.
  - These features **must** be available in MPI-3.0 if the target compiler is Fortran 2003+TR 29113 compliant.
    - For the mpi module and mpif.h, it is a question of the quality of the MPI library.

28

37

Minor changes

- MPI_ALLOC_MEM, MPI_WIN_ALLOCATE, MPI_WIN_ALLOCATE_SHARED and MPI_WIN_SHARED_QUERY return a base_addr. 35
  - In MPI-2.2, it is declared as INTEGER(KIND=MPI_ADDRESS_KIND) and may be usable for non-standard Cray-pointer, see Example 8.2 of the use of MPI_ALLOC_MEM
  - In MPI-3.0 in the mpi_f08 & mpi module, these routines are overloaded with a routine that returns a TYPE(C_PTR) pointer, see Example 8.1
- The buffer_addr argumet in MPI_BUFFER_DETACH is incorrectly defined and therefore unused. 31
- Callbacks are defined with explicit interfaces PROCEDURE(MPI_...) BIND(C) 41+42
- A clarification about comm_copy_attr_fn callback, see MPI_COMM_CREATE_KEYVAL: 34
  - Returned flag in Fortran must be LOGICAL, i.e., .TRUE. or .FLASE.

Detailed description of problems, mainly with the old support methods, or if the compiler does not support TR 29113:

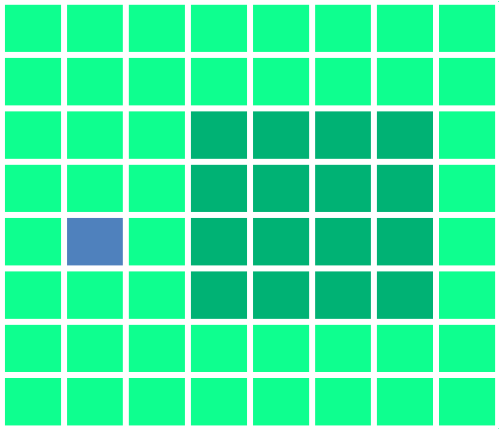
- 17.1.8 Additional Support for Fortran Register-Memory-Synchronization
- 17.1.10 Problems With Fortran Bindings for MPI
- 17.1.11 Problems Due to Strong Typing
- 17.1.12 Problems Due to Data Copying and Sequence Association with Subscript Triplets
- 17.1.13 Problems Due to Data Copying and Sequence Association with Vector Subscripts
- 17.1.14 Special Constants
- 17.1.15 Fortran Derived Types
- 17.1.16 Optimization Problems, an Overview
- 17.1.17 Problems with Code Movement and Register Optimization
  - Nonblocking Operations
  - One-sided Communication
  - MPI_BOTTOM and Combining Independent Variables in Datatypes
  - Solutions
  - The Fortran ASYNCHRONOUS Attribute
  - Calling MPI_F_SYNC_REG (new routine, defined in Section 17.1.7)

  - A User Defined Routine Instead of MPI_F_SYNC_REG
  - Module Variables and COMMON Blocks
  - The (Poorly Performing) Fortran VOLATILE Attribute
  - The Fortran TARGET Attribute
- 17.1.18 Temporary Data Movement and Temporary Memory Modication
- 17.1.19 Permanent Data Movement
- 17.1.20 Comparison with C

Status

- Passed final 2nd (final) vote at the March 2012 meeting
- Status of reference implementation
  - an initial implementation of the MPI 3.0 Fortran bindings are available in Open MPI
  - a full implementation will not be available until compilers implement new Fortran syntax added specifically to support MPI
    - need ASYNCHRONOUS attribute for nonblocking routines
    - need TYPE(*), DIMENSION(..) syntax to support subarrays
      - e.g. MPI_Irecv(Array(3:13:2), ...)

Group-Collective Communicator Creation



- MPI-2: Comm. creation is collective
- MPI-3: New group-collective creation
  - Collective only on members of new comm.
- Avoid unnecessary synchronization
  - Enable asynchronous multi-level parallelism
- Reduce overhead
  - Lower overhead when creating small communicators
- Recover from failures
  - Failed processes in parent communicator can't participate
- Enable compatibility with Global Arrays
  - In the past: GA collectives implemented on top of MPI Send/Recv

Courtesy of Jim Dinan and Richard Graham

MPI_TYPE_CREATE_HINDEXED_BLOCK

- MPI_TYPE_CREATE_HINDEXED_BLOCK is identical to MPI_TYPE_CREATE_INDEXED_BLOCK,
  except that block displacements in array_of_displacements are specied in bytes, rather than in multiples of the oldtype extent:

MPI_TYPE_CREATE_HINDEXED_BLOCK(count, blocklength, array_of_displacements, oldtype, newtype)

IN	count	length of array of displacements (non-negative integer)
IN	blocklength	size of block (non-negative integer)
IN	array_of_displacements	**byte** displacement of each block (array of integer)
IN	oldtype	old datatype (handle)
OUT	newtype	new datatype (handle)

Large Counts

- MPI-2.2
  - All counts are int / INTEGER
  - Producing longer messages through derived datatypes may cause problems
- MPI-3.0
  - New type to store long counts:
    - MPI_Count / INTEGER(KIND=MPI_COUNT_KIND)
  - Additional routines to handle “long” derived datatypes:
    - MPI_Type_size_x, MPI_Type_get_extent_x, MPI_Type_get_true_extent_x
  - “long” count information within a status:
    - MPI_Get_elements_x, MPI_Status_set_elements_x
  - Communication routines are not changed !!!
  - Well-defined overflow-behavior in existing MPI-2.2 query routines:
    - count in MPI_GET_COUNT, MPI_GET_ELEMENTS, and
      size in MPI_PACK_SIZE and MPI_TYPE_SIZE
      is set to MPI_UNDEFINED when that argument would overflow.

6

8

Tool Interfaces for MPI-3

- Goals of the tools working group
  - Extend tool support in MPI-3 beyond the PMPI interface
  - Document state of the art for de-facto standard APIs

Courtesy of the MPI-3 Tools working group

The MPI Performance Interface (MPI_T)

- Goal: provide tools with access to MPI internal information
  - Access to configuration/control and performance variables
  - MPI implementation agnostic: tools query available information
- Information provided as a set of variables
  - Performance variables (design similar to PAPI counters)
    Query internal state of the MPI library at runtime
  - Configuration/control variables
    List, query, and (if available) set configuration settings

Examples of Performance Vars.

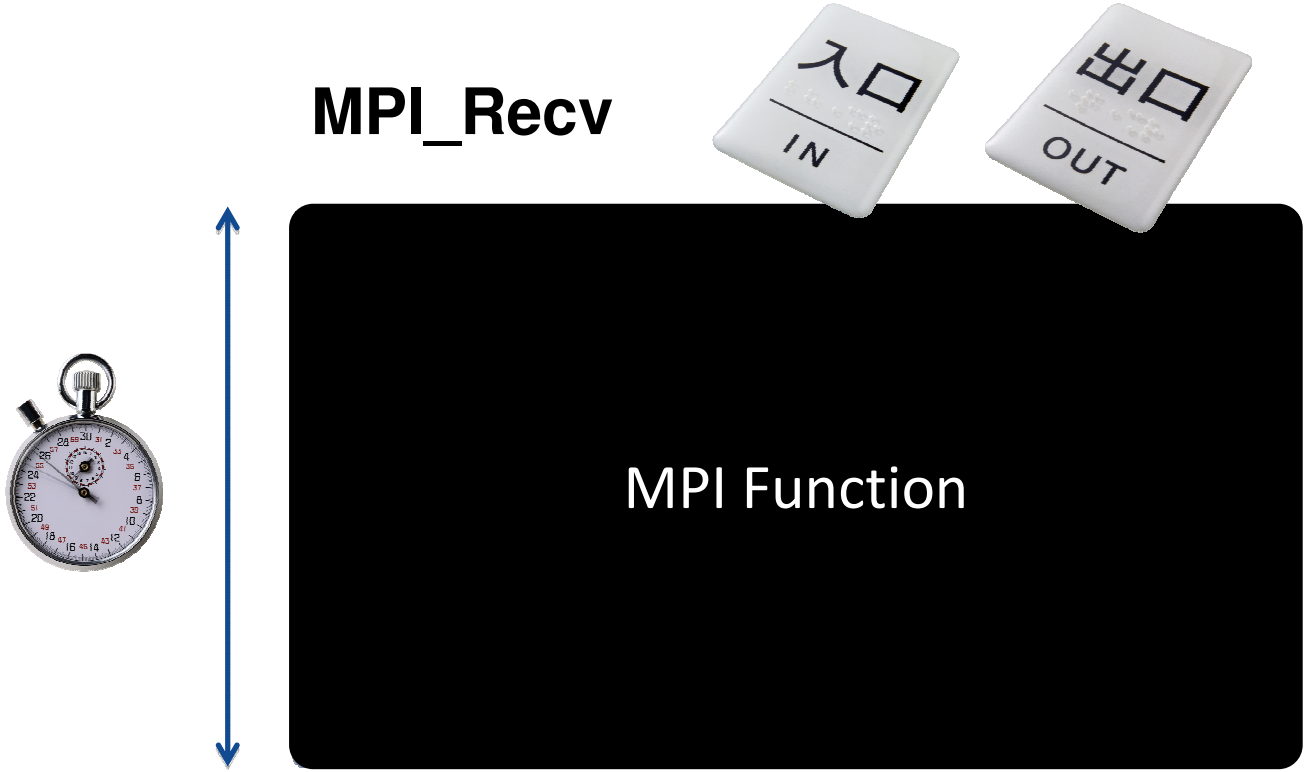
- Number of packets sent
- Time spent blocking
- Memory allocated

Examples for Control Vars.

- Parameters like Eager Limit
- Startup control
- Buffer sizes and management

- Complimentary to the existing PMPI Interface

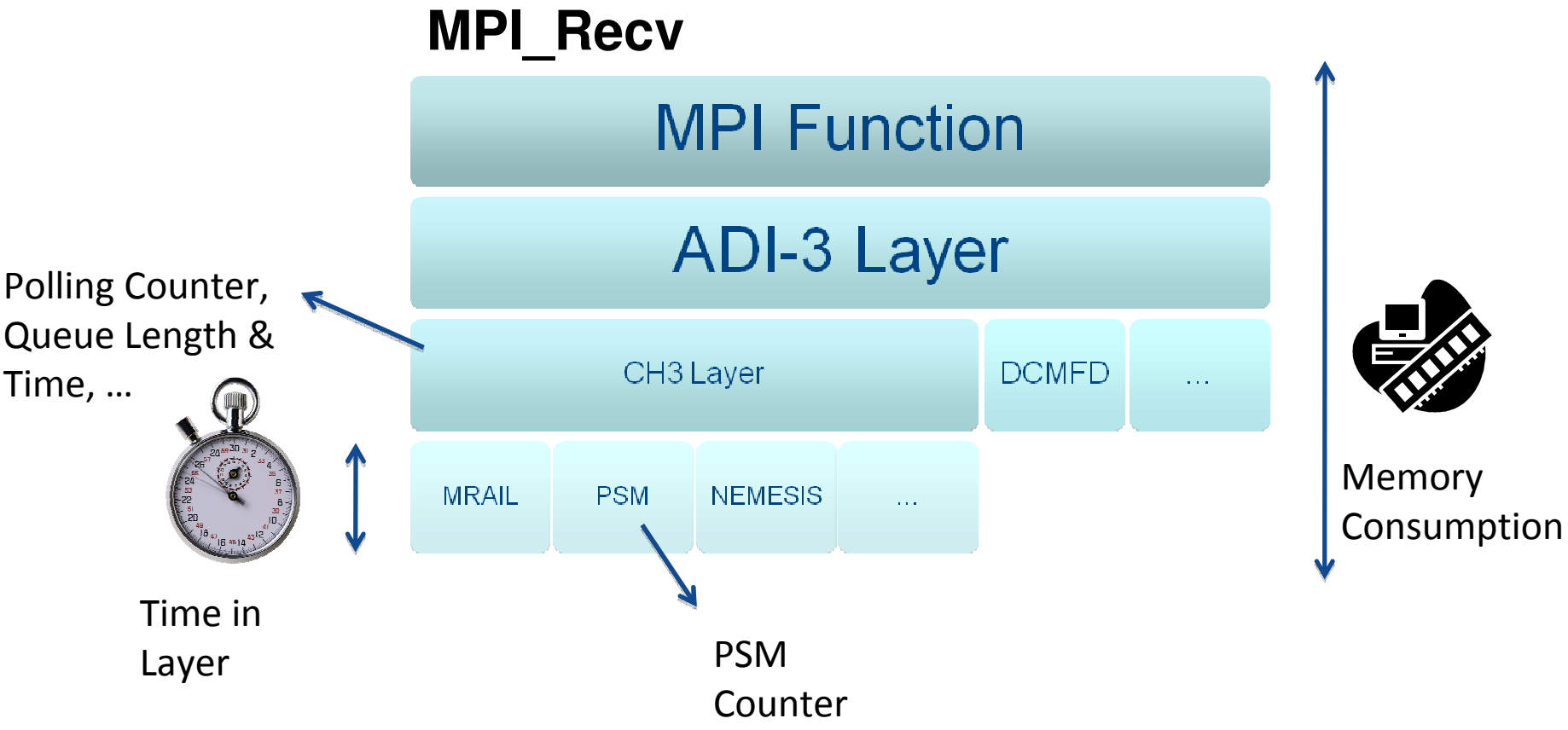
Granularity of PMPI Information



+ Information is the same for all MPI implementations
– MPI implementation is a black box

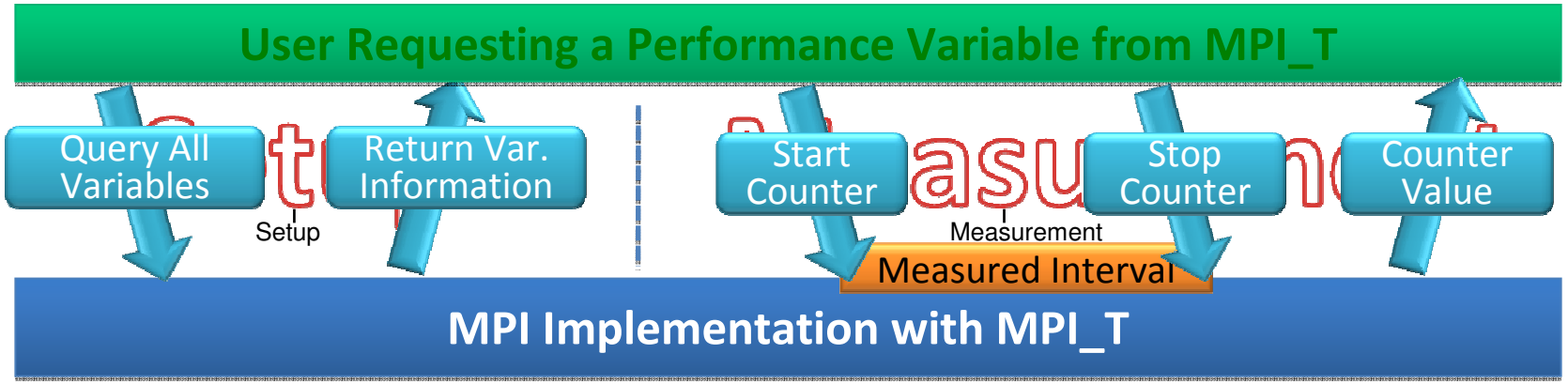
Granularity of MPI_T Information

Example: MVAPICH2



Some of MPI_T's Concepts

- Query API for all MPI_T variables / 2 phase approach
  - Setup: Query all variables and select from them
  - Measurement: allocate handles and read variables



  - Other features and properties
    - Ability to access variables before MPI_Init and after MPI_Finalize
    - Optional scoping of variables to individual MPI objects, e.g., communicator
    - Optional categorization of variables

Removing C++ bindings from the Standard

- Technical aspects of deprecated in MPI 2.2:
  - Supports MPI namespace
  - Support for exception handling
  - Not what most C++ programmers expect
- Decision in MPI-3.0:
  - Remove the C++ support from the standard
  - Use the C bindings – what most C++ developers do today
  - Preserve/add additional MPI predefined datatype handles in C and Fortran to support C++ types that are not provided by C
  - Preserve the MPI:: namespace and names with the meaning as defined in MPI-2.2 + MPI-2.2 errata, see MPI-3.0 Annex B.1.1
  - Perhaps provide the current bindings as a standalone library sitting on top of MPI,
  - or as part of MPI-3.0 libraries.

Further information

- www.mpi-forum.org → the official standard
- https://svn.mpi-forum.org/
  - View tickets (see headline boxes) → Custom query (right below headline boxes)
    - https://svn.mpi-forum.org/trac/mpi-forum-web/query
      → Filter → Version = MPI-3.0 **or** MPI-2.2-errata
- http://meetings.mpi-forum.org/
  - At a glance → All meeting information
    - http://meetings.mpi-forum.org/Meeting_details.php
  - MPI-3.0 Wiki
    - http://meetings.mpi-forum.org/MPI_3.0_main_page.php
    - Chapter Working groups:
      http://meetings.mpi-forum.org/mpi3.0_chapter_wgs.php

Thank you for your interest