MPI-3.0 and MPI-3.1 Overview

Rolf Rabenseifner

Lecture at “Recent Advances in Parallel Programming Languages”, Scientific Workshop @ LRZ, June 8, 2015

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.**

Goal & Scope of MPI-3.1

- Provide **small additions** to MPI 3.0 and integrate them together with identified **errata** items into a new version of the MPI standard.

Goal & Scope of MPI-4.0

- The MPI 4.0 standardization efforts aim
  - at adding new techniques, approaches, or concepts to the MPI standard

  that will help MPI
  - address the need of current and next generation applications and architectures.
- In particular, the following additions are currently being proposed and worked on:
  - Extensions to better support **hybrid programming** models
  - Support for **fault tolerance** in MPI applications
- Additionally, several working groups are working on new ideas and concepts, incl.
  - **Active messages**
  - **Stream messaging**
  - **New profiling interface**

Acknowledgements

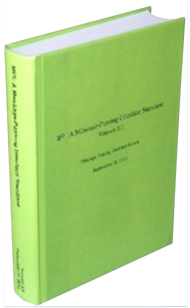
- Some detailed slides are provided by the
  - Ticket authors,
  - Chapter authors, or
  - Chapter working groups.
- Richard Graham, chair of MPI-3.0.
- Torsten Hoefler (additional example about new one-sided interfaces)

MPI-3.0 (Sep. 21, 2012) and MPI-3.1 (June 4, 2015) – the pdf files

- www.mpi-forum.org
  - MPI-3.0 documents

    → MPI 3.0 document as PDF

    http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

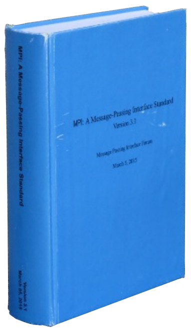


    → Hardcover (green book)

    http://www.hlrs.de/mpi/mpi30/

    \+ errata document

    http://www.mpi-forum.org/docs/mpi-3.0/errata-30.pdf
  - MPI-3.1 documents



    → MPI 3.1 document as PDF

    http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

    → Hardcover (blue book)

    http://www.hlrs.de/mpi/mpi31/ (planned, not yet available)

Change-Logs in MPI-3.1

B Change-Log

MPI-3.1 is mainly an errata release

MPI-3.0 has many important new features

MPI-2.1 with several small new features

MPI-2.0 combined 1.1 + 2.0 to one document

MPI-3.0 – Details about most & important topics

nn

Background information, see:
MPI-3.1, Change-Log, B.2.2 (1-42) & B.2.1 (E1-E6)

- **Major** additions
  - Slide 9: Nonblocking collectives
  - Slide 10: Sparse and scalable irregular collectives
  - Slides 11-15: One-sided communication – enhancements
  - Slides 16-21: Shared memory extensions (on clusters of SMP nodes)
  - Slides 22-35: Fortran interface
  - Slides 36-40: New tools interface
- **Minor** additions
  - Slide 42: Mprobe for hybrid programming on clusters of SMP nodes
  - Slide 43: Group-Collective Communicator Creation
  - Slide 44: MPI_TYPE_CREATE_HINDEXED_BLOCK
  - Slide 45: Large Counts
  - Slide 46: Removing C++ bindings from the Standard
  - Slide 47-48: Other forum activities and minor corrections

MPI-3.1 – Mainly an errata release

- Errata
  - Several errata in the new MPI Tool Information Interface chapter (Section 14.3)
  - New internal backend for the new Fortran interfaces (rewritten Section 17.1.5)
  - Only a few errata to the One-sided chapter (Chapter 11)
    - No errata to the new shared memory interface (Section 11.2.3 and other)

Must be implemented already in MPI-**3.0** libraries!

- **New Functionality and Features** → Slide 49
  - A **General Index** was added: should contain all relevant **MPI terms** (pages 816-819)
  - Intrinsic operators + and - for absolute addresses → substituted by **new functions** **MPI_AINT_ADD and MPI_AINT_DIFF**
  - MPI_INITIALIZED, MPI_FINALIZED, MPI_QUERY_THREAD, MPI_IS_THREAD_MAIN, MPI_GET_VERSION, and MPI_GET_LIBRARY_VERSION → **now without thread-safety restrictions**
  - **same_disp_unit** info key was added for use in RMA window creation routines
  - **Nonblocking collective MPI-I/O** routines added for *explicit addresses* and *individual file pointers*: MPI_FILE_IREAD_AT_ALL + MPI_FILE_IWRITE_AT_ALL and MPI_FILE_IREAD_ALL + MPI_FILE_IWRITE_ALL
    - Corresponding split collective interface was **not** declared as deprecated
  - MPI_T_... tools interface: 3 new routines; 2 new error codes; clarification about NULL parameters

Outline

Background information, see:
MPI-3.1, Change-Log, B.2.2 (1-42) & B.2.1 (E1-E6)

nn

- MPI-3.0 – Major additions
  - **Slide 9: Nonblocking collectives**
  - **Slide 10: Sparse and scalable irregular collectives**
  - Slides 11-15: One-sided communication – enhancements
  - Slides 16-21: Shared memory extensions (on clusters of SMP nodes)
  - Slides 22-35: Fortran interface
  - Slides 36-40: New tools interface
- MPI-3.0 – Minor additions
  - Slide 42: Mprobe for hybrid programming on clusters of SMP nodes
  - Slide 43: Group-Collective Communicator Creation
  - Slide 44: MPI_TYPE_CREATE_HINDEXED_BLOCK
  - Slide 45: Large Counts
  - Slide 46: Removing C++ bindings from the Standard
  - Slide 47-48: Other forum activities and minor corrections
- MPI-3.1
- Implementation Status

Nonblocking Collective Communication and MPI_ICOMM_DUP

13+15

- Idea
  - Collective initiation and completion separated
  - Offers opportunity to overlap computation and communication
  - Each blocking collective operation has a corresponding nonblocking operation: MPI_Ibarrier, MPI_Ibcast, …
  - May have multiple outstanding collective communications on the same communicator
  - Ordered initialization
  - Additional slide → Appendix App.
- Parallel MPI I/O: See MPI-3.1

Courtesy of Torsten Hoefler and Richard Graham

Sparse Collective Operations on Process Topologies

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
  - MPI_(I)NEIGHBOR_ALLGATHER(V)
  - MPI_(I)NEIGHBOR_ALLTOALL(V,W)
- 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

---

- Cartesian topology:
  - Sequence of buffer segments is communicated with:
    - direction=0 source, direction=0 dest, direction=1 source, direction=1 dest, …
  - Defined only for disp=1
  - If a source or dest rank is MPI_PROC_NULL then the buffer location is still there but the content is not touched.

Courtesy of Torsten Hoefler and Richard Graham

Outline

Background information, see:
MPI-3.1, Change-Log, B.2.2 $_{(1-42)}$ & B.2.1 $_{(E1-E6)}$ nn

- MPI-3.0 – Major additions
  - Slide 9: Nonblocking collectives
  - Slide 10: Sparse and scalable irregular collectives
  - **Slides 11-15: One-sided communication – enhancements**
  - **Slides 16-21: Shared memory extensions (on clusters of SMP nodes)**
  - Slides 22-35: Fortran interface
  - Slides 36-40: New tools interface

Courtesy of the MPI-3 One-sided working group

- MPI-3.0 – Minor additions
  - Slide 42: Mprobe for hybrid programming on clusters of SMP nodes
  - Slide 43: Group-Collective Communicator Creation
  - Slide 44: MPI_TYPE_CREATE_HINDEXED_BLOCK
  - Slide 45: Large Counts
  - Slide 46: Removing C++ bindings from the Standard
  - Slide 47-48: Other forum activities and minor corrections
- MPI-3.1
- Implementation status

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

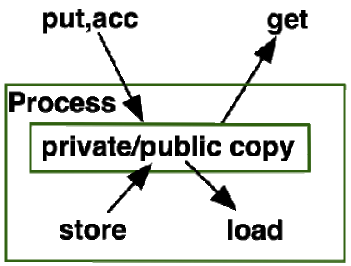
- Address the limitations of MPI-2 RMA by supporting the following features:
  - In order to support RMA to arbitrary locations, no constraints on memory, such as symmetric allocation or collective window creation, should be required
  - RMA operations that are imprecise (such as access to overlapping 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

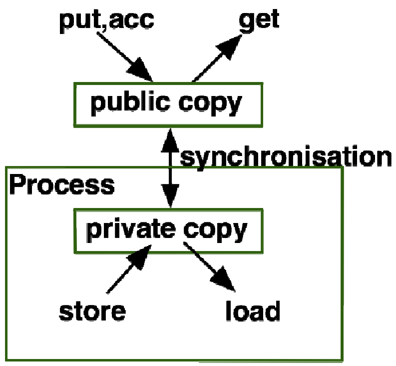
Major New Features in the MPI-3 One-sided Interface

- 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 / attach / detach
    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

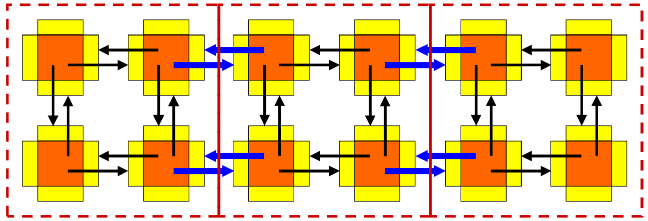
MPI-3 shared memory

- Split main communicator into shared memory islands
  - **MPI_Comm_split_type**
- Define a shared memory window on each island
  - **MPI_Win_allocate_shared**
  - Result (by default):
    contiguous array, directly accessible by all processes of the island
- Accesses and synchronization
  - Normal assignments and expressions
  - No **MPI_PUT/GET** !
  - Normal MPI one-sided synchronization, e.g., **MPI_WIN_FENCE**

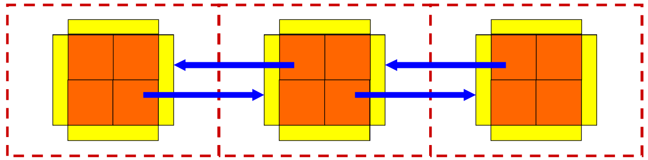
MPI-3.0 shared memory can be used
to **significantly reduce the memory needs**
for **replicated data**.

Hybrid shared/cluster programming models

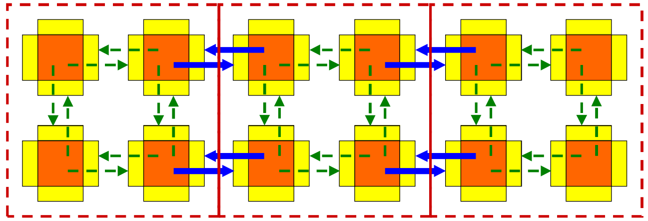
- MPI on each core (not hybrid)
  - Halos between all cores
  - MPI uses internally shared memory and cluster communication protocols



- MPI+OpenMP
  - Multi-threaded MPI processes
  - Halos communica. only between MPI processes

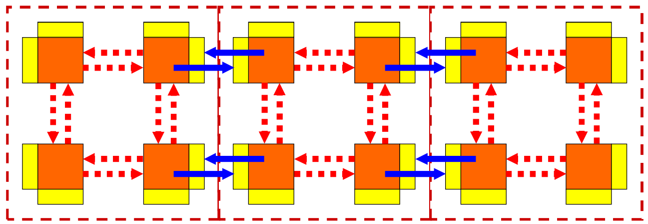


- MPI cluster communication
  + MPI shared memory **communication**
  - Same as “MPI on each core”, but
  - within the shared memory nodes, halo communication through direct copying with C or Fortran statements



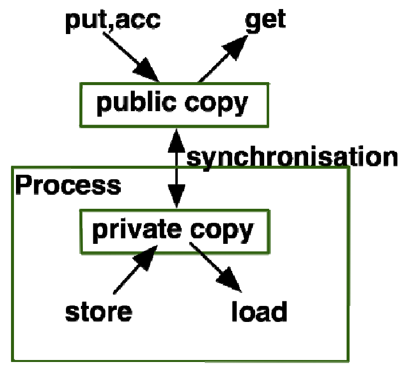
MPI inter-node communication
MPI intra-node communication
Intra-node direct Fortran/C copy
Intra-node direct neighbor access

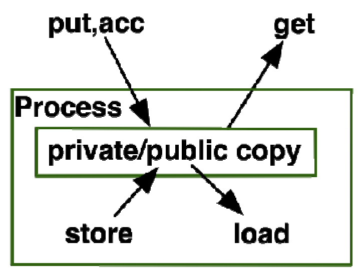
- MPI cluster comm. + MPI shared memory **access**
  - Similar to “MPI+OpenMP”, but
  - shared memory programming through work-sharing between the MPI processes within each SMP node



Two memory models

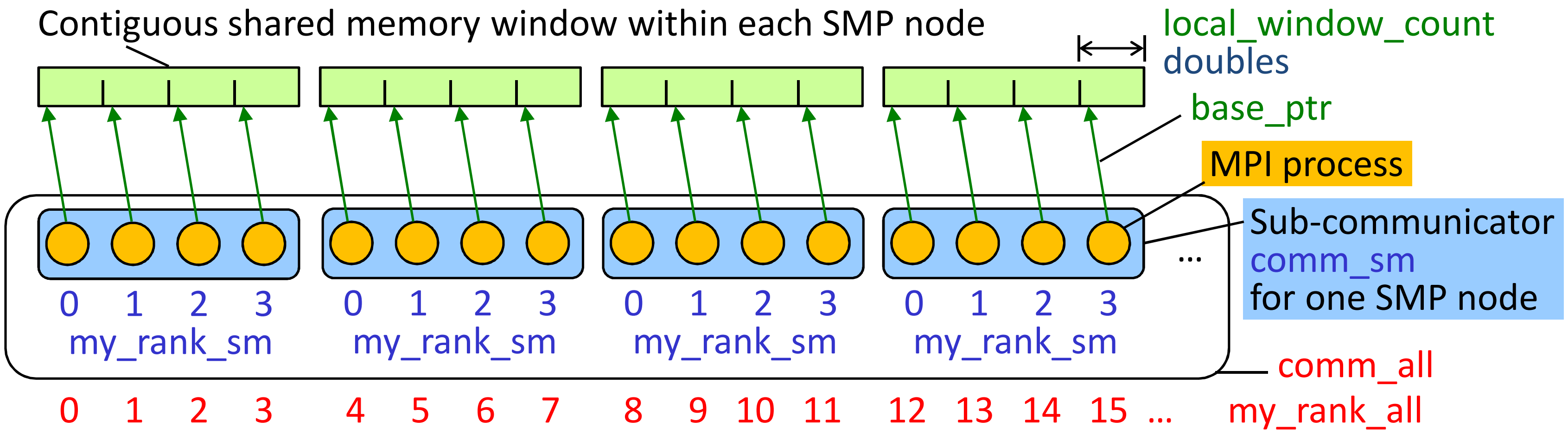
- Query for new attribute to allow applications to tune for cache-coherent architectures
  - Attribute MPI_WIN_MODEL with values
    - MPI_WIN_SEPARATE model
    - MPI_WIN_UNIFIED model on cache-coherent systems





- Shared memory windows always use the MPI_WIN_UNIFIED model
  - Public and private copies are **eventually** synchronized without additional RMA calls
    (MPI-3.0/MPI-3.1, Section 11.4, page 436/435 lines 37-40/43-46)
  - For synchronization **without delay**: MPI_WIN_SYNC()
    (MPI-3.0 errata https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/413)
    (MPI-3.1 Section 11.8, Example 11.21 on pages 468-469)
  - or any other RMA synchronization:
    "A consistent view can be created in the unified memory model (see Section 11.4) by utilizing the window synchronization functions (see Section 11.5) or explicitly completing outstanding store accesses (e.g., by calling MPI_WIN_FLUSH)."
    (MPI-3.0/MPI-3.1, MPI_Win_allocate_shared, page 410/408, lines 16-20/43-47)

Splitting the communicator & contiguous shared memory allocation



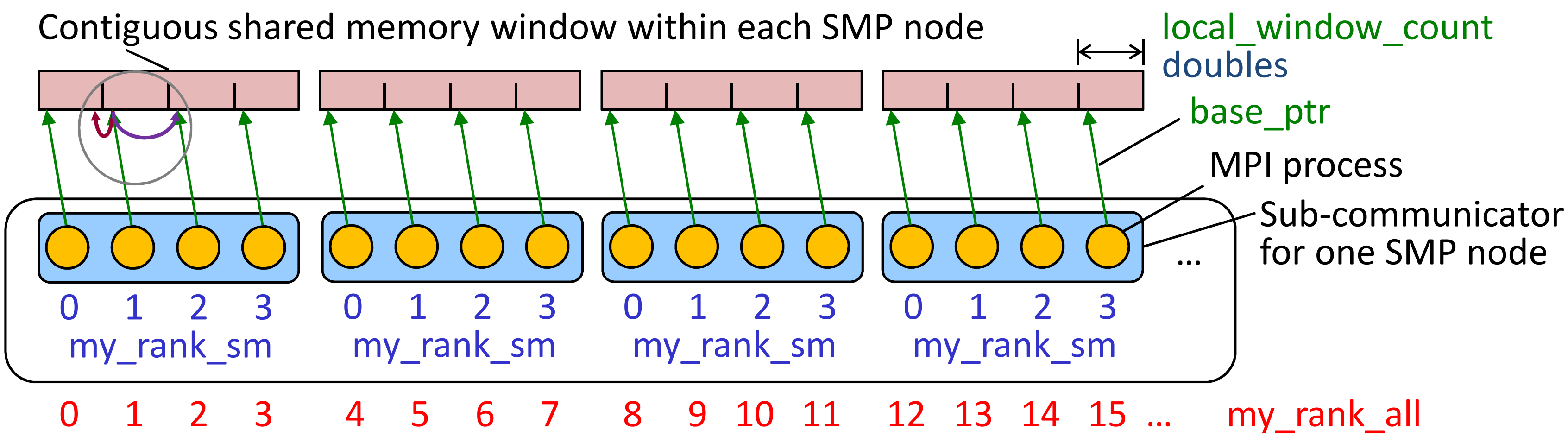

```
MPI_Aint /*IN*/ local_window_count;   double /*OUT*/  *base_ptr;
MPI_Comm  comm_all, comm_sm;          int  my_rank_all, my_rank_sm, size_sm, disp_unit;
MPI_Comm_rank (comm_all, &my_rank_all);
MPI_Comm_split_type (comm_all, MPI_COMM_TYPE_SHARED, 0,
                     MPI_INFO_NULL, &comm_sm);
MPI_Comm_rank (comm_sm, &my_rank_sm);  MPI_Comm_size (comm_sm, &size_sm);
disp_unit = sizeof(double);  /* shared memory should contain doubles */
MPI_Win_allocate_shared (local_window_count*disp_unit, disp_unit, MPI_INFO_NULL,
                         comm_sm, &base_ptr, &win_sm);
```

Sequence in comm_sm as in comm_all

Within each SMP node – Essentials

- The allocated shared memory is contiguous across process ranks,
- i.e., the first byte of rank i starts right after the last byte of rank i-1.
- Processes can calculate remote addresses' offsets with local information only.
- Remote accesses through load/store operations,
- i.e., without MPI RMA operations (MPI_GET/PUT, …)
- Although each process in comm_sm accesses the same physical memory, the virtual start address of the whole array may be different in all processes!
  → **linked lists** only with offsets in a shared array, but **not with binary pointer addresses!**

Shared memory access example




```
MPI_Aint /*IN*/ local_window_count;        double /*OUT*/  *base_ptr;
MPI_Win_allocate_shared (local_window_count*disp_unit,  disp_unit,  MPI_INFO_NULL,
                         comm_sm,  &base_ptr,  &win_sm);
MPI_Win_fence (0, win_sm);  /*local store epoch can start*/
for (i=0; i<local_window_count; i++)  base_ptr[i] = ... /* fill values into local portion */
MPI_Win_fence (0, win_sm);  /* local stores are finished, remote load epoch can start */
if (my_rank_sm > 0)           printf("left neighbor's rightmost value = %lf \n", base_ptr[-1] );
if (my_rank_sm < size_sm-1)   printf("right neighbor's leftmost value = %lf \n",
                                      base_ptr[local_window_count] );
```

Synchronization (before and after each MPI_Win_fence, marked F)

Local stores

Direct load access to remote window portion

F: In Fortran, before and after the synchronization, one must declare the buffer as ASYNCHRONOUS and must add:
IF(.NOT.MPI_ASYNC_PROTECTS_NONBLOCKING) CALL MPI_F_SYNC_REG (buffer)
to guarantee that register copies of buffer are written back to memory, respectively read again from memory.

Outline

Background information, see:
MPI-3.1, Change-Log, B.2.2 $_{(1\text{-}42)}$ & B.2.1 $_{(E1\text{-}E6)}$ nn

- MPI-3.0 – Major additions
  - Slide 9: Nonblocking collectives
  - Slide 10: Sparse and scalable irregular collectives
  - Slides 11-15: One-sided communication – enhancements
  - Slides 16-21: Shared memory extensions (on clusters of SMP nodes)
  - **Slides 22-35: Fortran interface**
    - **A high-level summary for non-Fortran programmers**
    - **Details for Fortran programmers**
  - Slides 36-40: New tools interface
- MPI-3.0 – Minor additions
  - …
  - …
- MPI-3.1
- Implementation status

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'
  - The use of the include file mpif.h is **strongly discouraged** starting with MPI-3.0. new 40
  - Does not guarantee 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 **new**

- 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

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

Removed, see MPI-3.0 errata Sep. 24, 2013 and later

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

Nonblocking Receive and Register Optimization / Code Movement in Fortran

HLRS

- **Fortran source code:**

  MPI_IRECV (*buf*, ..., *request_handle*, *ierror*)
  MPI_WAIT(request_handle, *status*, *ierror*)
  write (*,*) buf

  buf is not part of the argument list

- **may be compiled as**

  MPI_IRECV (*buf*, ..., *request_handle*, *ierror*)
  registerA = buf
  MPI_WAIT(request_handle, *status*, *ierror*)
  write (*,*) **registerA**

  Data may be received in buf during MPI_Wait

  Therefore old data may be printed instead of the newly received data

- **Solution:**
  - ASYNCHRONOUS :: buf
  - *buf* may be allocated in a common block or module data, or
  - IF (.NOT. MPI_ASYNC_PROTECTS_NONBLOCKING) &
    & CALL MPI_F_SYNC_REG(buf)

  In the scope including nonblocking call and MPI_Wait

  Directly after CALL MPI_Wait

MPI_ASYNC_PROTECTS_NONBLOCKING == .TRUE. requires a TS29113 compiler

- **Work-around in older MPI versions:**
  - call **MPI_GET_ADDRESS**(buf, *iaddrdummy*, *ierror*)

    with INTEGER(KIND=MPI_ADDRESS_KIND) iaddrdummy

  i.e., if MPI_F_SYNC_REG and MPI_ASYNC_PROTECTS_NONBLOCKING is not yet available

Major changes

- Support method: 26

  USE mpi or INCLUDE 'mpif.h' → **USE mpi_f08** (new)

- 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-3.0/3.1, Section 17.2.5

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

Major changes, continued

- Unique handle types, e.g.,
  - INTEGER new_comm → **TYPE(MPI_Comm) :: new_comm** (new)

  Same names as in C

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

27

Major changes, continued

new

- 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
    - (1st) the user creates a portable datatype handle and
    - (2nd) 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 29

  USE mpi_f08

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

  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.

    → Use at least MPI-3.0 standard document

Major enhancement with a full MPI-3.0 implementation

- The following features require Fortran 2003 + TR 29113
  - Subarrays may be passed to nonblocking routines new 28
    - This feature is available if the LOGICAL compile-time constant MPI_SUBARRAYS_SUPPORTED == .TRUE.
  - new Correct handling of buffers passed to nonblocking routines, 37
    - 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.

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 argument 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 .FALSE.

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

Implementation

- Initial implementations of the MPI 3.0 Fortran bindings are based on Fortran 2003
  - OpenMPI → MPI-3.0 compliant
- MPICH strategy:
  - MPI-3.0 compliant only with TS 29113-compilers
  - Without TS 29113-compilers
    - All of MPI-3.0 routines available with mpif.h and mpi module
    - MPI module (partially)
      without compile argument checking & keyword-based argument lists
      → Not MPI-3.0 compliant

Outline

Background information, see:
MPI-3.1, Change-Log, B.2.2 (1-42) & B.2.1 (E1-E6)

nn

- MPI-3.0 – Major additions
  - Slide 9: Nonblocking collectives
  - Slide 10: Sparse and scalable irregular collectives
  - Slides 11-15: One-sided communication – enhancements
  - Slides 16-21: Shared memory extensions (on clusters of SMP nodes)
  - Slides 22-35: Fortran interface
  - **Slides 36-40: New tools interface**
    - 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

- MPI-3.0 – Minor additions
  - …
- MPI-3.1
- Implementation status

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

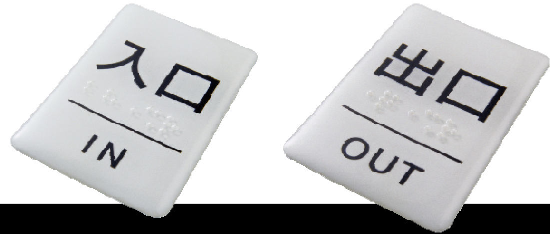
Examples for Control Vars.

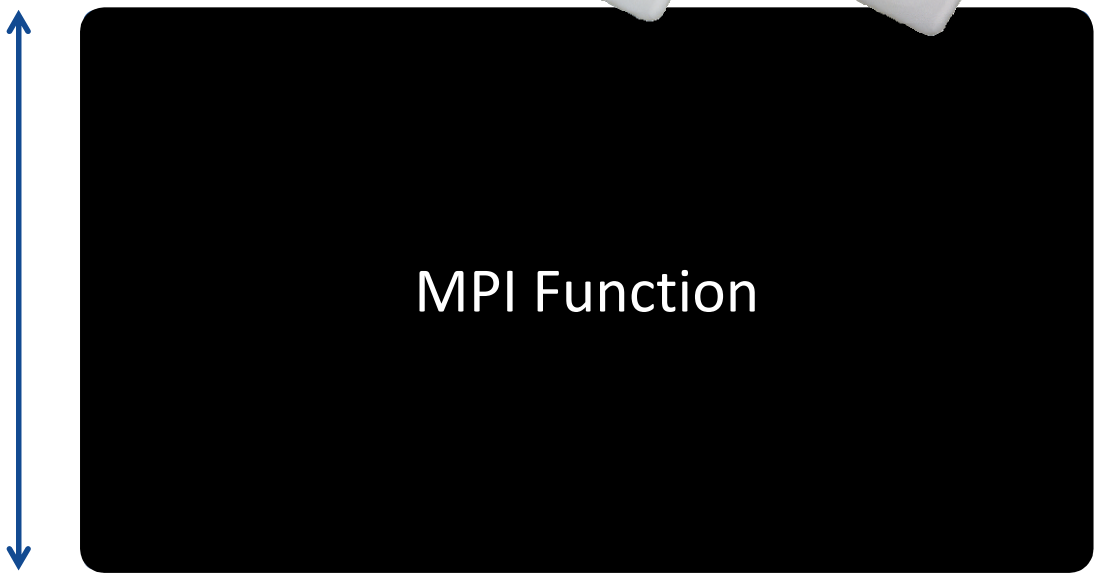
- Parameters like Eager Limit
- Startup control
- Buffer sizes and management

- Complimentary to the existing PMPI Interface

Granularity of PMPI Information

MPI_Recv

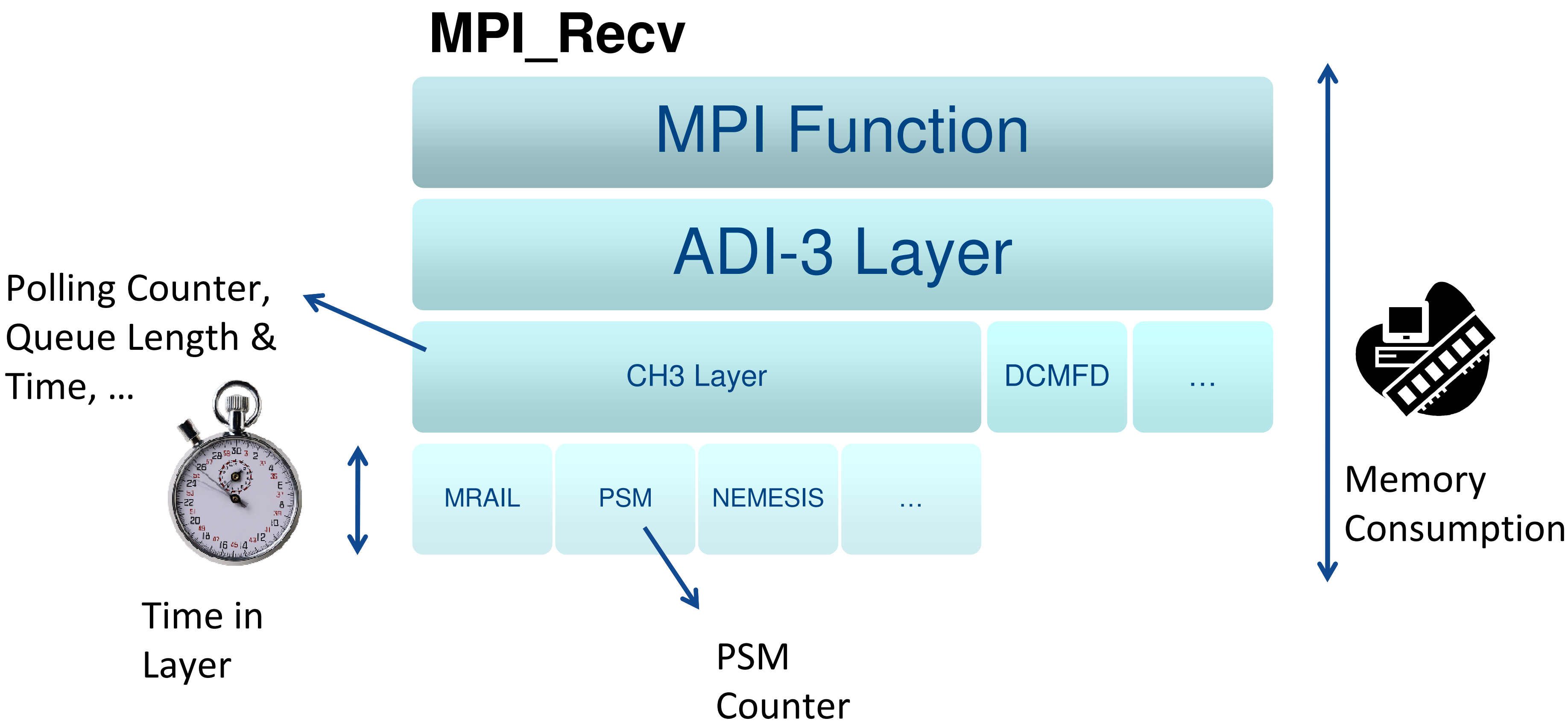




- \+ Information is the same for all MPI implementations
- – MPI implementation is a black box

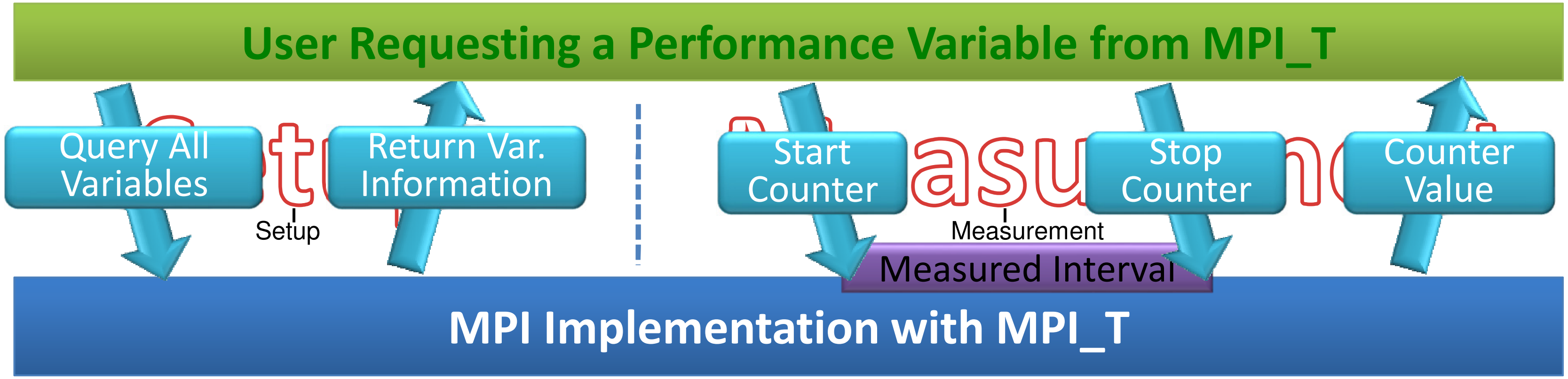
Granularity of MPI_T Information

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

Outline

Background information, see:
MPI-3.1, Change-Log, B.2.2 (1-42) & B.2.1 (E1-E6)

nn

- MPI-3.0 – Major additions
  - Slide 9: Nonblocking collectives
  - Slide 10: Sparse and scalable irregular collectives
  - Slides 11-15: One-sided communication – enhancements
  - Slides 16-21: Shared memory extensions (on clusters of SMP nodes)
  - Slides 22-35: Fortran interface
  - Slides 36-40: New tools interface
- MPI-3.0 – Minor additions
  - **Slide 42: Mprobe for hybrid programming on clusters of SMP nodes**
  - **Slide 43: Group-Collective Communicator Creation**
  - **Slide 44: MPI_TYPE_CREATE_HINDEXED_BLOCK**
  - **Slide 45: Large Counts**
  - **Slide 46: Removing C++ bindings from the Standard**
  - **Slide 47-48: Other forum activities and minor corrections**
- MPI-3.1
- **Implementation status**

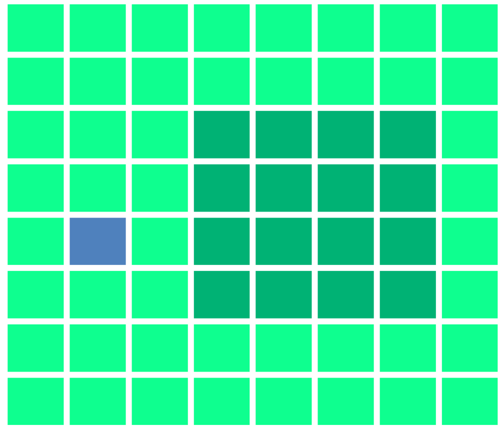
Thread-safe probe: MPI_(I)MPROBE & MPI_(I)MRECV

- 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

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: 6
    - 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 !!! 8
  - 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.

Removing C++ bindings from the Standard

2

- MPI-2 C++ Interface:
  - Not what most C++ programmers expect
  - Deprecated in MPI-2.2 / removed in MPI-3.0
- 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
- Special C++ types are supported through additional MPI predefined datatypes (in C and Fortran) 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>
- 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.

Other Forum Activities

- 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
- 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
- Removed MPI-1.1 functionality stored in new Chapter 16 (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

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
- The MPI_C_BOOL "external32" representation is 1-byte E5

MPI-3.1 – Mainly an errata release

- Errata
  - Several errata in the new MPI Tool Information Interface chapter (Section 14.3)
  - New internal backend for the new Fortran interfaces (rewritten Section 17.1.5)
  - Only a few errata to the One-sided chapter (Chapter 11)
    - No errata to the new shared memory interface (Section 11.2.3 and other)

Must be implemented already in MPI-**3.0** libraries!

- **New Functionality and Features**
  - A **General Index** was added: should contain all relevant **MPI terms** (pages 816-819)
  - Intrinsic operators + and - for absolute addresses
    → substituted by **new functions MPI_AINT_ADD and MPI_AINT_DIFF**
  - MPI_INITIALIZED, MPI_FINALIZED, MPI_QUERY_THREAD, MPI_IS_THREAD_MAIN, MPI_GET_VERSION, and MPI_GET_LIBRARY_VERSION → **now without thread-safety restrictions**
  - **same_disp_unit** info key was added for use in RMA window creation routines
  - **Nonblocking collective MPI-I/O** routines added for *explicit addresses* and *individual file pointers*: MPI_FILE_IREAD_AT_ALL + MPI_FILE_IWRITE_AT_ALL and MPI_FILE_IREAD_ALL + MPI_FILE_IWRITE_ALL
    - Corresponding split collective interface was **not** declared as deprecated
  - MPI_T_... tools interface: 3 new routines; 2 new error codes; clarification about NULL parameters

Address calculations in MPI-3.1 and later

Instead of intrinsic integer address operators + and –

New absolute address := existing absolute address + relative displacement

- C/C++: *MPI_Aint* **MPI_Aint_add**(MPI_Aint base, MPI_Aint *disp*)
- Fortran: *INTEGER(KIND=MPI_ADDRESS_KIND)* **MPI_Aint_add**(base, disp)
  INTEGER(KIND=MPI_ADDRESS_KIND) :: base, disp

Relative displacement := absolute address 1 – absolute address 2

- C/C++: *MPI_Aint* **MPI_Aint_diff**(MPI_Aint addr1, MPI_Aint *addr2*)
- Fortran: *INTEGER(KIND=MPI_ADDRESS_KIND)* **MPI_Aint_diff**(addr1, addr2)
  INTEGER(KIND=MPI_ADDRESS_KIND) :: addr1, addr2

Examples: (MPI-3.0 / MPI-3.1, Example 4.8, page 103 / 102 and Example 4.17, pp 125-127)

Fortran — New in MPI-3.1

```
REAL a(100,100)
INTEGER(KIND=MPI_ADDRESS_KIND) iaddr1, iaddr2, disp;  INTEGER ierror
CALL MPI_GET_ADDRESS( a(1,1), iaddr1, ierror)     ! The address of a(1,1) is stored in iaddr1
CALL MPI_GET_ADDRESS( a(10,10), iaddr2, ierror)
disp = MPI_Aint_diff(iaddr2, iaddr1)              ! MPI-3.0 & former: disp = iaddr2–iaddr1
```

C — New in MPI-3.1

```
float a[100][100];  MPI_Aint iaddr1, iaddr2, disp;
MPI_Get_address( &a[0][0], &iaddr1);  // the address value &a[0][0] is stored into variable iaddr
MPI_Get_address( a,        &iaddr1);  // same result, because a represents the address of
                                      //   the first element, i.e. &a[0][0]
MPI_Get_address( &a[9][9], &iaddr2);
disp = MPI_Aint_diff(iaddr2, iaddr1);  // MPI-3.0 & former: disp = iaddr2–iaddr1
```

Status of MPI-3.0 Implementations

	MPICH	MVAPICH	Open MPI	Cray MPI	Tianhe MPI	Intel MPI	IBM BG/Q MPI [1]	IBM PE MPICH	IBM Platform	SGI MPI	Fujitsu MPI	MS MPI
NB collectives	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	*
Neighborhood collectives	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	
RMA	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	
Shared memory	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	✓
Tools Interface	✓	✓	✓	✓	✓	✓	✓ [2]	✓	Q3 '15	✓	Q2 '15	*
Non-collective comm. create	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	
F08 Bindings	✓	✓	✓	✓	✓	Q2 '15	✓	✓	Q3 '15	✓	Q2 '15	
New Datatypes	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	*
Large Counts	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	Q2 '15	*
Matched Probe	✓	✓	✓	✓	✓	✓	✓	✓	Q3 '15	✓	✓	*

Release dates are estimates and are subject to change at any time.

March 2015

Empty cells indicate no *publicly announced* plan to implement/support that feature.

Platform-specific restrictions might apply for all supported features.

Courtesy of Pavan Balaji (ANL)

[1] Open source, but unsupported   [2] No MPI_T variables exposed   * Under development

Further information

- www.mpi-forum.org → MPI documents → the official standard & link to printed books
- 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 → Tickets for MPI-3.0 document
      → Version = MPI-3.1 **or** MPI-3.0-errata → Tickets for MPI-3.1 document
      → Version = MPI-4.0 **or** MPI<next> → Tickets for future MPI document
- http://meetings.mpi-forum.org/
  - At a glance → All meeting information
    - http://meetings.mpi-forum.org/Meeting_details.php
  - MPI-3.1 Wiki and chapter committees
    - http://meetings.mpi-forum.org/MPI_3.1_main_page.php
  - MPI-3.1/4.0 Working groups:
    - http://meetings.mpi-forum.org/MPI_4.0_main_page.php

Thank you for your interest

APPENDIX

Additional slides on

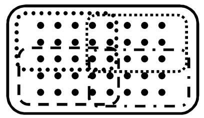
- Nonblocking collective communication (slide 53)
- MPI shared memory with one-side communication
  - Other synchronization on shared memory – with MPI_WIN_SYNC (slide 54)
  - General MPI-3 shared memory synchronization rules (slide 55)
    (write-read-rule, read-write-rule, write-write-rule)
  - Benchmark results (slide 56)
    (Low latency and high bandwith by combining pt-to-pt synchronization & direct shared memory store)
- The MPI Forum: After final vote for MPI-3.1, June 4, 2015 (slide 58)

Nonblocking Collective Communication Routines

New in MPI-3.0

MPI_I.......... **Nonblocking** variants of all collective communication:
MPI_Ibarrier, MPI_Ibcast, …

- **Nonblocking** collective operations do **not match** with **blocking** collective operations
- Collective initiation and completion are separated
- May have multiple outstanding collective communications on same communicator
- Ordered initialization on each communicator
- Offers opportunity to overlap
  - several collective communications, e.g., on several overlapping communicators
    - **Without deadlocks or serializations!**
  - computation and communication



With point-to-point message passing, such matching is allowed

→ **Often a background MPI progress engine is missing or not efficient**

→ **Alternative:**

- Several calls to MPI_Test(), which enables progress
- Use non-standard extensions to switch on asynchronous progress
  - export MPICH_ASYNC_PROGRESS=1

Implies a helper thread and MPI_THREAD_MULTIPLE, see Chapter 13. MPI and Threads

Other synchronization on shared memory

- If the shared memory data transfer is done without RMA operation, then the synchronization can be done by other methods.
- This example demonstrates the rules for the unified memory model if the data transfer is implemented only with load and store (instead of MPI_PUT or MPI_GET) and the synchronization between the processes is done with MPI communication (instead of RMA synchronization routines).

```
Process A                          Process B

MPI_WIN_LOCK_ALL(                  MPI_WIN_LOCK_ALL(
MPI_MODE_NOCHECK,win)              MPI_MODE_NOCHECK,win)

DO ...                             DO ...
  X=...
  MPI_F_SYNC_REG(X) 1)
  MPI_WIN_SYNC(win)
  MPI_Send  ---------------------->  MPI_Recv
                                     MPI_WIN_SYNC(win)
                                     MPI_F_SYNC_REG(X) 1)
                                     local_tmp = X
                                     MPI_F_SYNC_REG(X) 1)
                                     MPI_WIN_SYNC(win)
  MPI_Recv  <----------------------  MPI_Send
                                     print local_tmp
  MPI_WIN_SYNC(win)
  MPI_F_SYNC_REG(X) 1)
END DO                             END DO

MPI_WIN_UNLOCK_ALL(win)            MPI_WIN_UNLOCK_ALL(win)
```

Data exchange in this direction, therefore **MPI_WIN_SYNC** is needed in both processes: **Write-read-rule**

Also needed due to **read-write-rule**

1) Fortran only.

- The used synchronization must be supplemented with MPI_WIN_SYNC, which acts only locally as a processor-memory-fence.
  For MPI_WIN_SYNC, a passive target epoch is established with MPI_WIN_LOCK_ALL.
- **X** is part of a shared memory window and should be **the same** memory location **in both processes.**

- See also tickets #413 and #456: https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/413 and https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/456

General MPI-3 shared memory synchroniz. rules

Defining Proc 0 — Proc 1

Sync-from → Sync-to

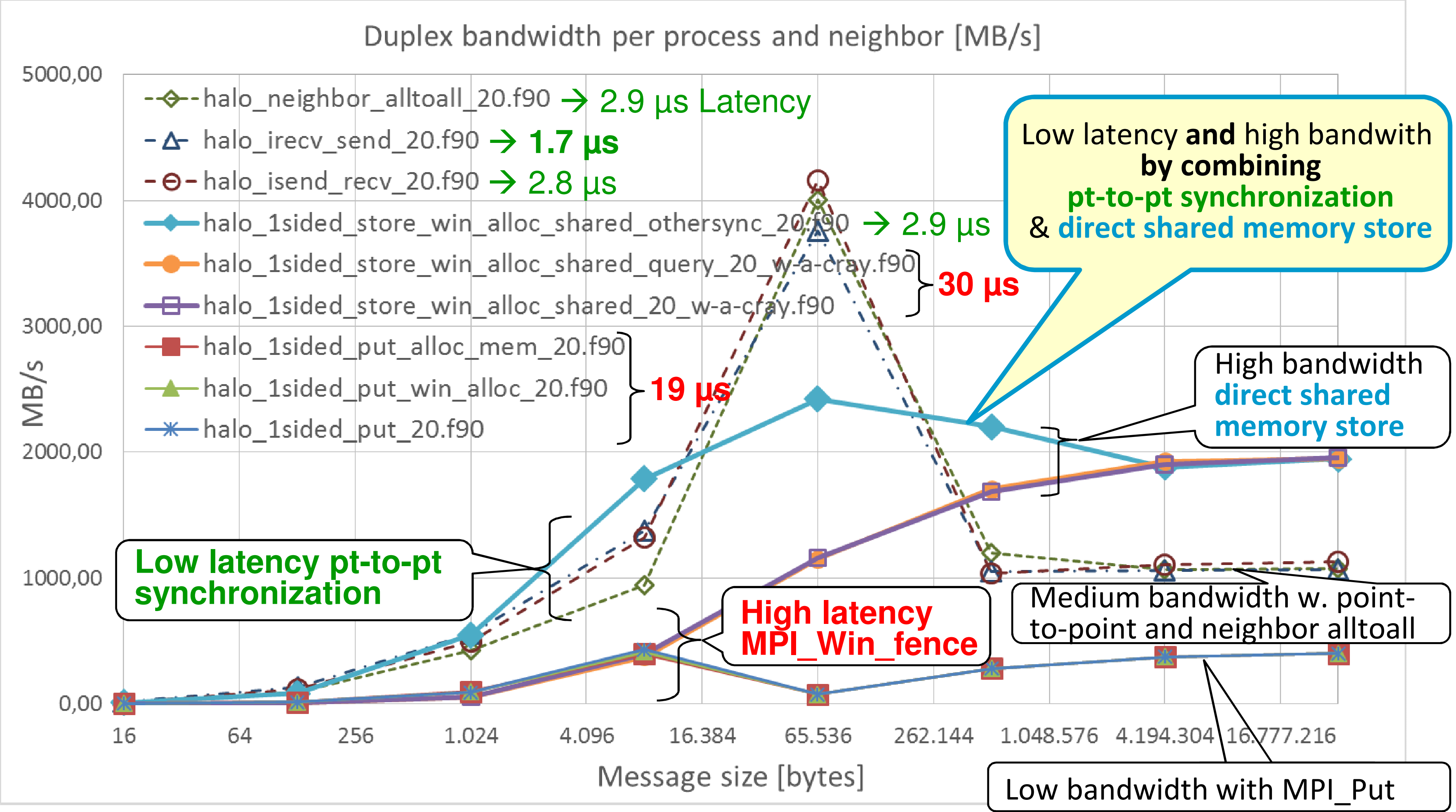
being MPI_Win_post[1)] → MPI_Win_start[1)]
or MPI_Win_complete[1)] → MPI_Win_wait[1)]
or MPI_Win_fence[1)] ↔ MPI_Win_fence[1)]
or MPI_Win_sync, Any-process-sync[2)] → Any-process-sync[2)], MPI_Win_sync (Candidate for **lowest latency**)
or MPI_Win_unlock[1)] → MPI_Win_lock[1)]
and the lock on process 0 is granted first

and having …	**then it is guaranteed that …**
A=val_1 Sync-from → Sync-to load(A)	⇒ **… the load(A) in P1 loads val_1** (this is the write-read-rule)
load(B) Sync-from → Sync-to B=val_2	⇒ **… the load(B) in P0 is not affected by the store of val_2 in P1** (read-write-rule)
C=val_3 Sync-from → Sync-to C=val_4 load(C)	⇒ **… that the load(C) in P1 loads val_4** (write-write-rule)

1) Must be paired according to the general one-sided synchronization rules.

2) "Any-process-sync" may be done with methods from MPI (e.g. with send-->recv as in MPI-3.1 Example 11.13, but also with some synchronization through MPI shared memory loads and stores, e.g. with C++11 atomic loads and stores).

Benchmark results on a Cray XE6 –
1-dim ring communication on 1 node w. 32 cores



Benchmark on Cray XE6 Hermit at HLRS
with aprun –n 32 –d 1 –ss, best values out of 6 repetitions, modules PrgEnv-cray/4.1.40 and cray-mpich2/6.2.1

The MPI Forum: After final vote for MPI-3.1, June 4, 2015

Photo by D. Eder

Attendance of the meeting June 1-4, 2015, in Chicago: **34 participants** from **24 organisations.**