

From MPI-1.1 to MPI-3.1, publishing and teaching, with a special focus on MPI-3 shared memory and the Fortran nightmare

Rolf Rabenseifner, HLRS, University of Stuttgart, www.hlrs.de
rabenseifner@hlrs.de



Outline

My background

Printing the MPI standards

Fortran, a nightmare ?!?

Complete MPI-3.1 Courses / Tutorials

The MPI shared memory interface

My Background

- Sent by HLRS to the MPI-Forum since MPI-2
 - Impressed from the very democratic process
 - Rusty always tried to break it down to binary decisions
- It was my way to learn what MPI is
 - the whole so far existing MPI-1.1
 - and of course all MPI-2
- My apologies - I was helping in the work for MPI 1-sided
 - but the result was not really as good as we hoped.
 - I looked at consistency, but no idea about performance.
- 10 years of pause (between MPI-2.0 and the start for MPI-3)



Bill Saphir (left) and
Ewing (Rusty) Lusk (ANL, MPI1.2 & 2.0 convener and meeting chair)

MPI 1.2 and MPI-2.0 Forum 1995 - 1997



Marc Snir, Bill Gropp, Bill Saphir (from left)

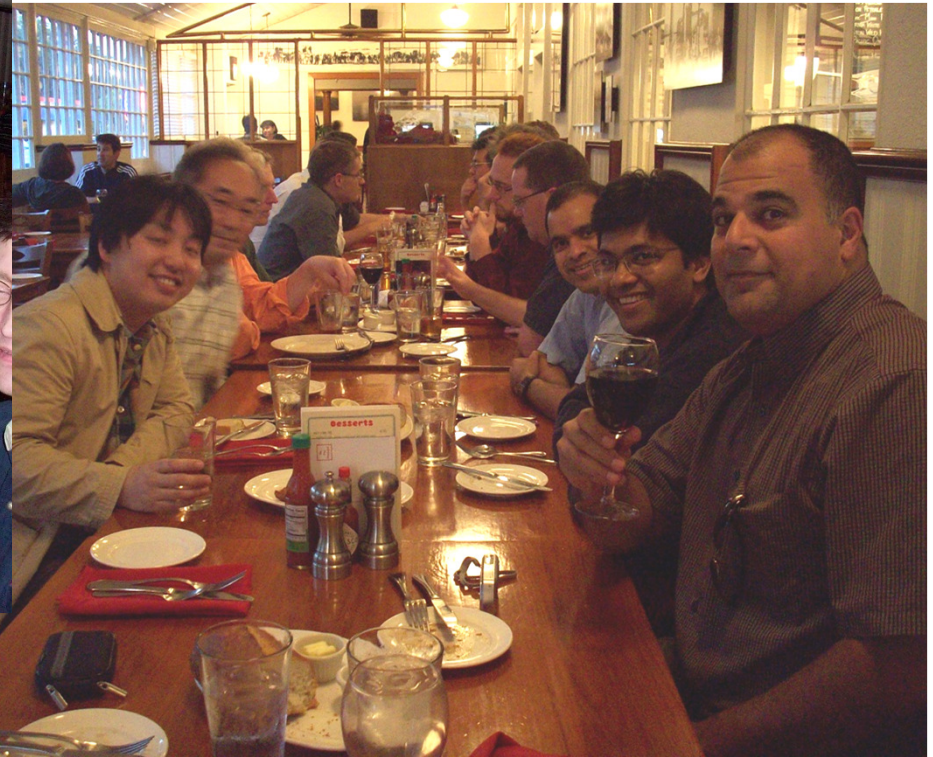
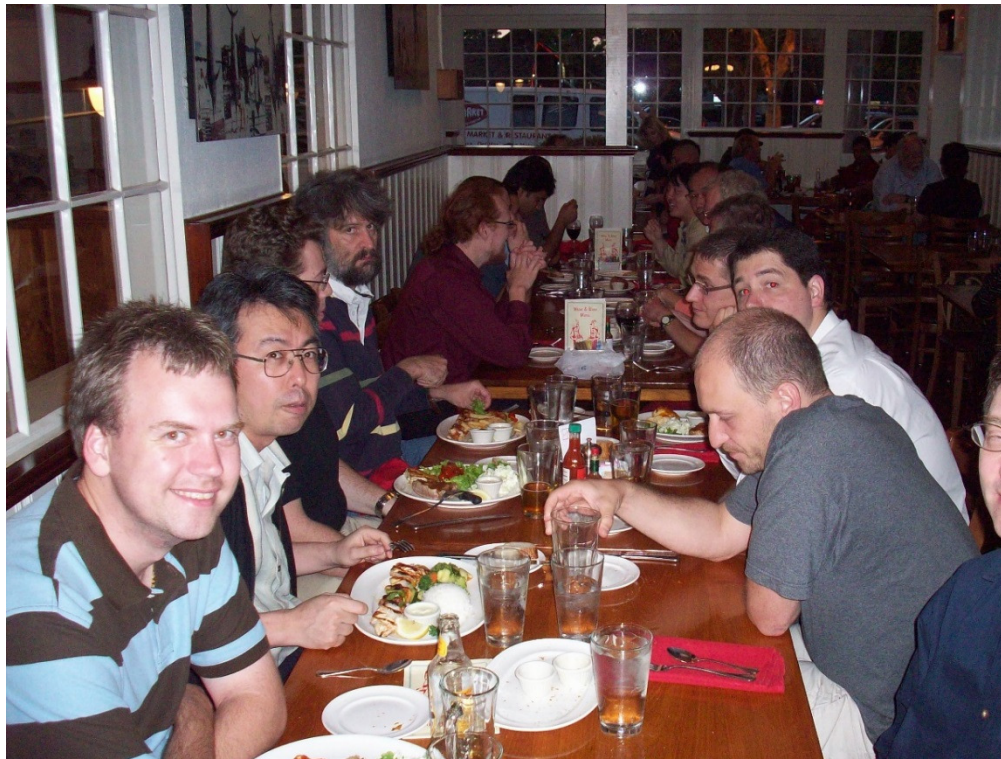
My Background (continued)

- Rich Graham asked me to get member of the MPI steering committee and invited me to a telcon in December 2007.
 - I prepared a plan to securely combine MPI-1.1 + MPI-2.0
 - with full control about all lines & without losing portions by bad luck.
 - At the telcon there was a long discussion on the strange situation of two documents
 - until Rusty Lusk said something like
“stop, Rolf said he will do it - we need not to discuss!”
- I detected many years later that all the others were sitting around a table and I was the only one on a pure phone - no webex!
- MPI-2.1 then started in the meeting Jan. 14-16, 2008 in Chicago
- After I managed MPI-2.1, I was really knowing the whole MPI 😊



MPI-2.1 Forum Meeting June 30 – July 2, 2008, Menlo Park, CA, USA
First vote for MPI-1.3 and **MPI-2.1**

Pictures from Rolf Rabenseifner



MPI-2.1 Forum Meeting
June 30 – July 2, 2008,
Menlo Park, CA, USA
Forum Dinner



After final vote for **MPI-3.0**, Sep. 21, 2012, at MPI Forum meeting in Vienna, Austria, Sep. 20-21, 2012

(Photos by Jesper Traeff and Rolf Rabenseifner)

(Combined by Jutta Sauer)

1st row sitting (from left to right): [1] Alexander Supalov (Intel), [2] William (Bill) Gropp (NCSA/UIUC),

2nd row sitting: [3] Rolf Rabenseifner (HLRS), [4] David Goodell (ANL), [5] Jeff Squyres (Cisco), [6] Brian Barrett (Sandia), [7]] Brian Smith (ORNL),

3rd + 4th row sitting: [8] Jesper Traeff (TU Vienna), [9] George Bosilca (INRIA), [10] Aurelien Bouteiller (U. Tennessee), [11] Atsushi Hori (Riken AICS),

Standing: [12] Rich Graham (Mellanox, MPI-3.0 chair), [13] Manjunath Gorentla Venkata (ORNL), [14] Shinji Sumumoto (Fujitsu), [15] Puri Bangalore (UAB),

[16] Hideyuki Jitsumoto (U.Tokyo), [17] Takeshi Nanri (Kyushu U.), [18] Christian Siebert (GRS-Sim),

[19] Devendar Bureddy(OSU), [20] Paddy Gillies (AWE Plc), [21] Tomotake Nakamura (Riken AICS)

Probably not on the picture, but at the meeting: Nathan Heljm (LANL)

(Thanks to Atsushi Hori for assisting)



After final vote for **MPI-3.1**, June 4, 2015, at MPI Forum meeting in Chicago, June 1-4, 2015

(Photo by David Eder with smart phone of Jeff Squyres)

1st row sitting: [1] William (Bill) Gropp (NCSA/UIUC), [2] Martin Schulz (LLNL, MPI-3.1 chair), [3] Jeff Squyres (Cisco), [4] Rolf Rabenseifner (HLRS), [5] Rich Graham (Mellanox)
 2nd row sitting: [6] Anh Vo (Microsoft), [7] Pavan Balaji (ANL), [8] Xiaoyi Lu (OSU), [9] Krishna Kandalla (Cray),
 3rd + 4th row sitting: [10] Takafumi Nose (Fujitsu), [11] Aurelien Bouteiller (U Tennessee), [12] Atsushi Hori (Riken), [13] Wesley Bland (Intel), [14] Sangmin Seo (ANL),
 Standing: [15] Sameh Sharkawi (IBM), [16] Alice Koniges (LBNL), [17] Chulho Kim (Lenovo), [18] Kathryn Mohror (LLNL), [19] Ryan Grant (Sandia),
 [20] Puri Bangalore (UAB), [21] Jeff Hammond (Intel), [22] Daniel Holmes (EPCC), [23] Lena Oden (ANL), [24] Howard Pritchard (LANL), [25] Takeshi Nanri (Kyushu U)
 [26] Sayantan Sur (Intel), [27] Ignacio Laguna Peralta (LLNL), [28] Nathan Hjelm (LANL), [29] Manjunath Gorentla Venkata (ORNL), [30] Sreeram Potluri (Nvidia)
 At the meeting, but not on the picture: Rajeev Thakur (ANL), Anthony Skjellum (Auburn U), Ken Raffanetti (ANL), Junchao Zhang (ANL) *(Thanks to Jeff Squyres for assisting)*

.....



Printing the MPI standards

HLRS as MPI book publisher

- In our many training courses,
always people like to have MPI as a book!
 - MPI-2.1 (608 pages, 821g=29oz, June 23, 2008)
 - 916 printed / **738 sold** / 178 unsold
 - MPI-2.2 (647 pages, 840g=29.6oz, Sep 4, 2009)
 - 921 printed / **900 sold** / 21 unsold 😊 😊
 - MPI-3.0 (852 pages, 1031g=36oz!!, Sep 21, 2012)
 - 1055 printed / **969 sold** / 86 unsold 😊
 - MPI-3.1 (868 pages, 1066g=38oz!!, June 4, 2015)
 - 1040 printed / **487 sold** (by Sep 20, 2017)
/ 170 expected until end of 2018
/ 383 unsold (still enough if MPI-4 is coming 2019/2020)



For whom are the MPI standards

- MPI implementers
- MPI users
 - It is still not a tutorial
 - But well readable & with many examples and “advices to users”
 - And we added several index sections
 - Latest the “Global Index” (since MPI-3.1, see page 816ff)
- My recommendation, use together
 - The current MPI standard
 - And the books “Using MPI” and “Using advanced MPI”

Also helpful for the implementers,
because they are also human beings

MPI pdf and book from
<http://mpi-forum.org/docs/>
19.50 € / \$ 23 + shipping



+



Images & further information from
<http://wgropp.cs.illinois.edu/usingmpiweb/> ■

.....

Fortran, a nightmare ?!?

Fortran, a nightmare ?!?

- Only a few MPI Forum members speak Fortran
 - The few ones had a hard job to get MPI and Fortran consistent
- Major problems: Compiler optimizations may lead to wrong MPI execution
 - with all MPI_Wait/Test routines
 - with using MPI_BOTTOM together with derived datatypes
 - with absolute addresses
 - calling nonblocking routines with strided data arrays that are not simple contiguous
- Already in MPI-2.0 (1997!) the inconsistency-problem was known
 - but more than some text about a user-written "dd" dummy routine as a work-around was not going through the Forum!

Fortran, a nightmare – solved in MPI-3.0 (15 years later) ?!?

- For MPI-3.0 we received full service from the Fortran standardization body by “Fortran Technical Specification TS 29113”
 - Enabling the new Fortran module `mpi_f08`
 - which is the first time full consistent with the Fortran standard
 - Major solution:
Fortran extended the `ASYNCHRONOUS` keyword for any asynchronous use-case, including MPI nonblockings and `MPI_BOTTOM`
- In MPI-3.0 we did the backend wrong ☹ – my apologies
 - A whole section in an errata → MPI-3.1
 - Did really slowed down the implementation
 - Still some MPI implementations claim to be MPI-3.1 compliant
 - although they do **not** provide **compile-time argument checking**
 - nor **name based argument list** with the `mpi` module

.....

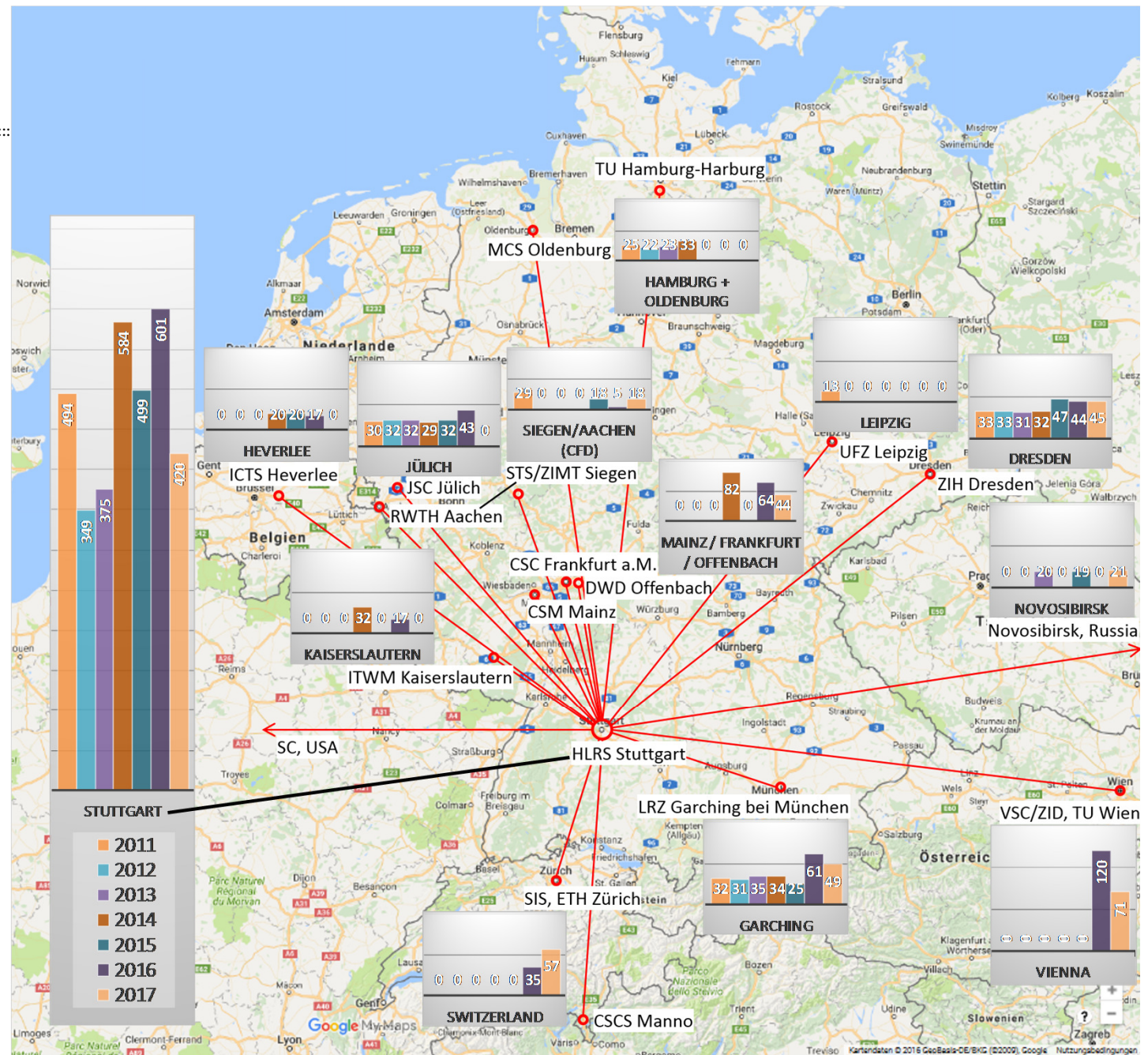
Complete MPI-3.1 Courses / Tutorials

Teaching complete advanced MPI-3.1

- **Important** for users can take **advantages**
 - from all the work in the MPI Forum, and
 - from the implementations of all the new MPI features in many MPI libraries
 - My MPI-3.1 course is based on the MPI-1.1 course from EPCC
 - They did a great job!
- **Nonblocking collectives**
 - **The New Fortran Module mpi_f08**
 - Groups & Communicators, Environment Management
 - MPI_Comm_split, intra- & inter-communicators
 - **Re-numbering on a cluster**, collective communication on inter-communicators, info object, naming & attribute caching, implementation information
 - Virtual topologies
 - **including neighborhood communication** +MPI_BOTTOM
 - One-sided Communication
 - **Shared Memory One-sided Communication**
 - **including hybrid MPI and MPI-3 shared memory programming**
 - **MPI memory models and synchronization rules**
 - Derived datatypes
 - including advanced features, **alignment, resizing**
 - Parallel File I/O
 - MPI and Threads, e.g., **hybrid MPI and OpenMP**
 - Probe, Persistent Requests, Cancel
 - Process Creation and Management

The network of HLRS courses

- Cooperation with several centers in Germany and EU
- 1007 participants in 39 courses in 2016



.....

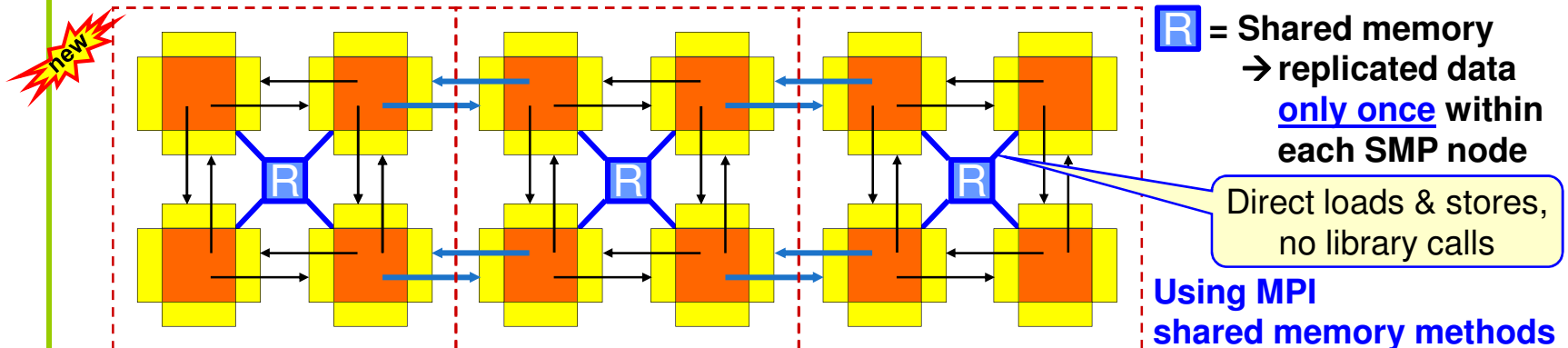
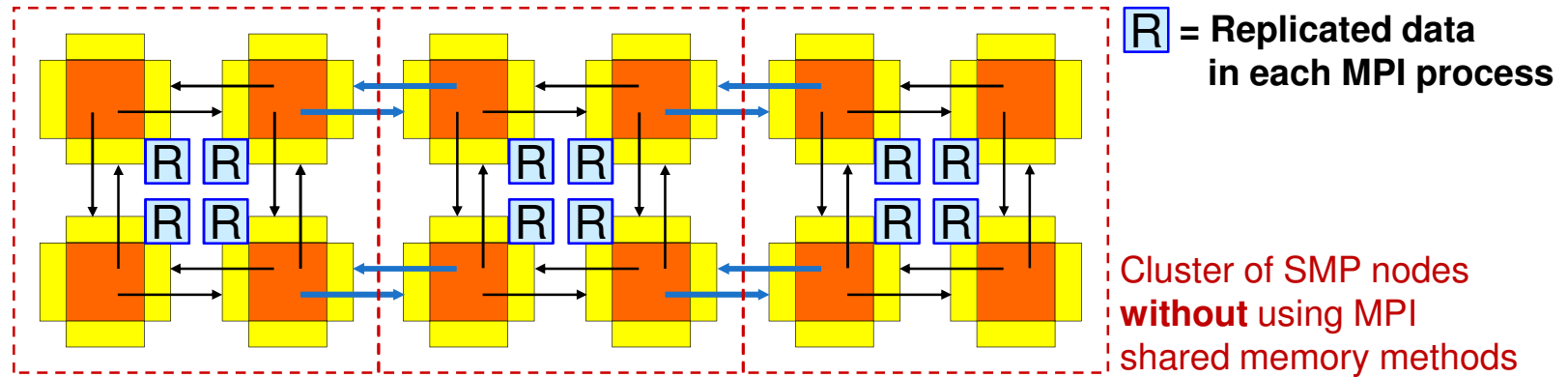
The MPI shared memory interface

MPI-3 shared memory interface

- Help users to understand the MPI-3 shared memory interface
 - mainly for minimizing memory needs for replicated data (only once per shared memory node)
 - advanced synchronization rules for minimizing latencies when synchronizing MPI shared memory accesses

Programming opportunities with MPI shared memory:

1) Reducing memory space for replicated data

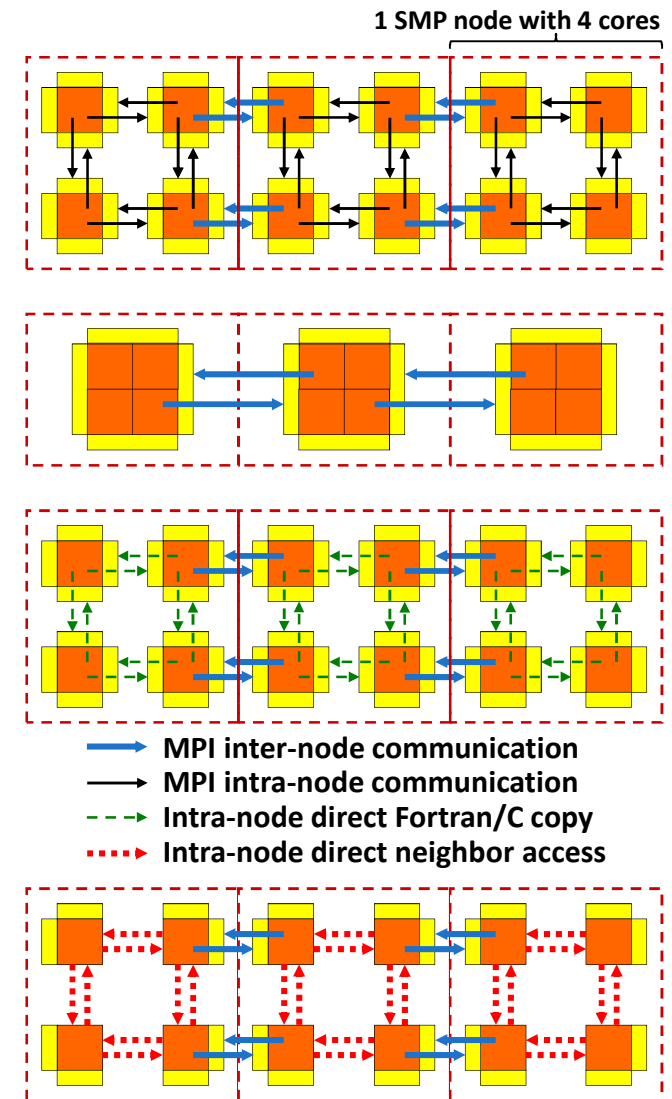


MPI-3.0 shared memory can be used
to significantly reduce the memory needs for replicated data.

— skipped —

Programming opportunities with MPI shared memory: 2) Hybrid shared/cluster programming models

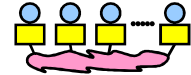
- MPI on each core (not hybrid)
 - Halos between all cores
 - MPI uses internally shared memory and cluster communication protocols
- MPI+OpenMP
 - Multi-threaded MPI processes
 - Halos communica. only between MPI processes
- new** • MPI cluster communication + MPI shared memory communication
 - Same as “MPI on each core”, but
 - within the shared memory nodes, halo communication through direct copying with C or Fortran statements
- new** • 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



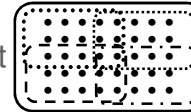
skipped

Chap.11 Shared Memory One-sided Communication

1. MPI Overview
2. Process model and language bindings
3. Messages and point-to-point communication
4. Nonblocking communication
5. The New Fortran Module mpi_f08
6. Collective communication
7. Error Handling
8. Groups & communicators, environment management
9. Virtual topologies
10. One-sided communication

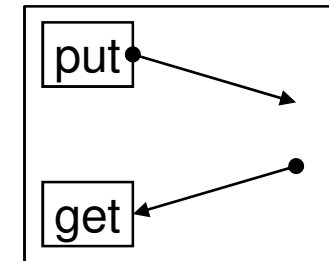


MPI_Init()
MPI_Comm_rank()

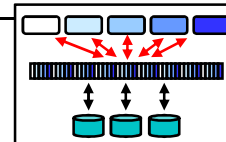
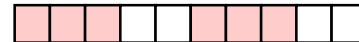


11. Shared memory one-sided communication

- (1) MPI_Comm_split_type & MPI_Win_allocate_shared
Hybrid MPI and MPI-3 shared memory programming
- (2) MPI memory models and synchronization rules



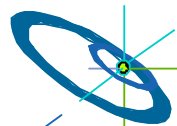
12. Derived datatypes
13. Parallel file I/O
14. MPI and threads
15. Probe, Persistent Requests, Cancel
16. Process creation and management
17. Other MPI features
18. Best Practice



— skipped —

Lowest latencies

- Usage of MPI shared memory without one-sided synchronization methods
- MPI provides the shared memory, but used
 - only with compiler generated loads & stores
 - together with C++11 memory fences



skipped

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)

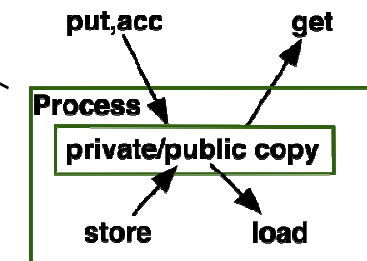
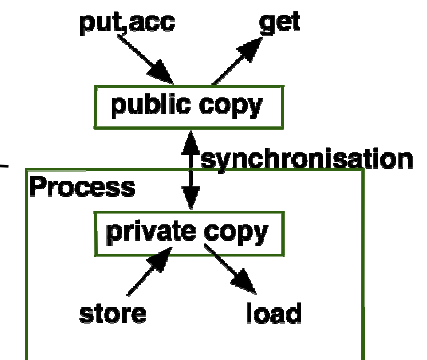
- For synchronization **without delay**: **MPI_WIN_SYNC()**

(MPI-3.1 Section 11.7: "Advice to users. In the unified memory model..." on page 456, and 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)



skipped

“eventually synchronized” – the Problem

- The problem with shared memory programming using libraries is:

X is a variable in a shared window initialized with 0.

Or any other process-to-process synchronization, e.g., using also shared memory stores and loads

Process
Rank 0

X = 1

MPI_Send(empty msg to rank 1) → MPI_Recv(from rank 0)

Process
Rank 1

printf ... X

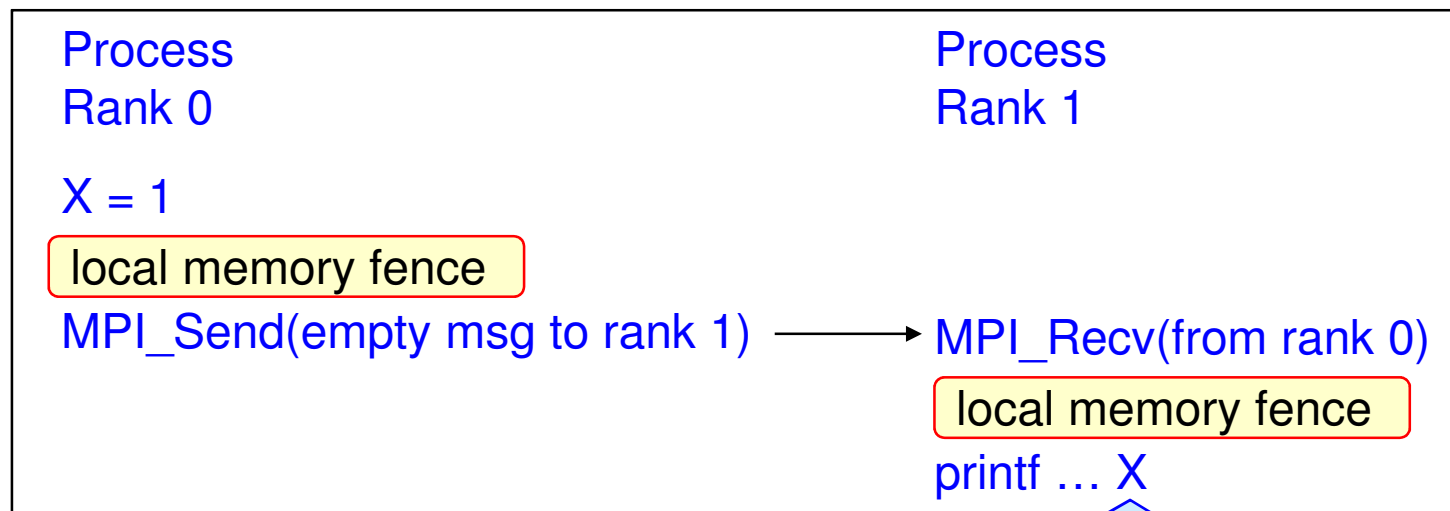
X can be still 0,
because the “1” will be eventually visible to the other process,
i.e., the “1” will be visible but maybe too late ☹ ☹ ☹

— skipped —

“eventually synchronized” – the Solution

- A pair of local memory fences is needed:

X is a variable in a shared window initialized with 0.



Now, it is guaranteed that the “1” in X is visible in this process



— skipped —

“eventually synchronized” – Last Question

Several options & heavy discussions in the MPI Forum

$X = 1$

X is a variable in a shared memory window initialized with 0.

local memory fence

$\text{MPI_Send}(\text{empty msg to rank 1}) \longrightarrow \text{MPI_Recv}(\text{from rank 0})$

local memory fence

$\text{printf} \dots X$

How to make the local memory fence ?

- C11 `atomic_thread_fence(order)`
 - **Advantage:** one can choose appropriate order = `memory_order_acquire`, or `..._release` to achieve minimal latencies
- `MPI_Win_sync`
 - **Advantage:** works also for Fortran
 - **Disadvantage:** may be slower than C11 `atomic_thread_fence` with appro. order
- Using RMA synchronization with integrated local memory fence instead of `MPI_Send` \rightarrow `MPI_Recv`
 - **Advantage:** May prevent double fences
 - **Disadvantage:** The synchronization itself may be slower

5 sync methods,
see next slide

$X = 1$

X is a variable in a shared memory window initialized with 0.

`MPI_Win_fence`

Include needed
memory fence

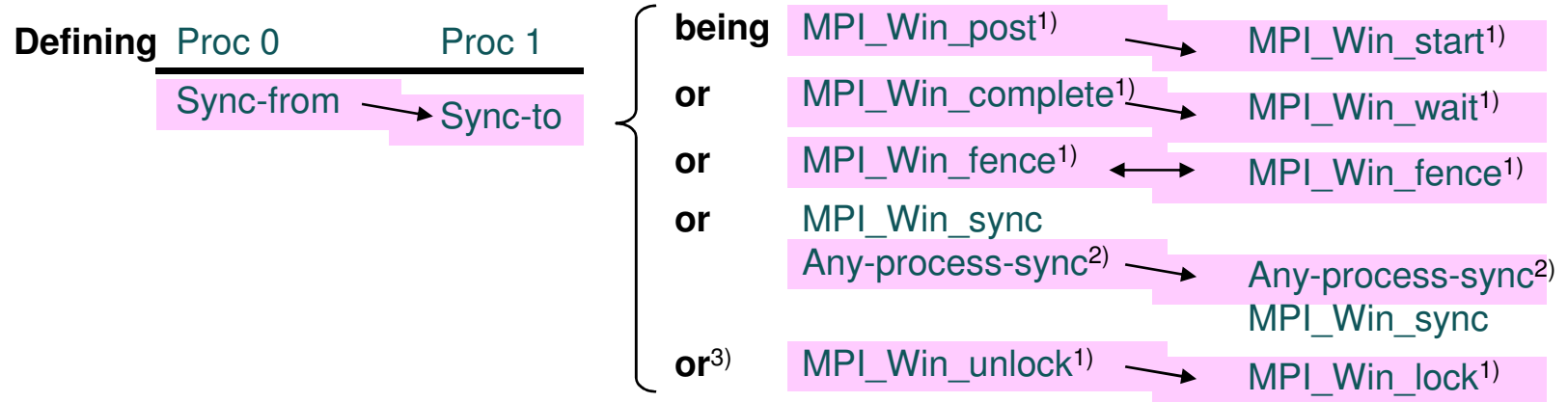
\longrightarrow `MPI_Win_fence`

Includes needed
memory fence

$\text{printf} \dots X$

General MPI-3 shared memory synchronization rules

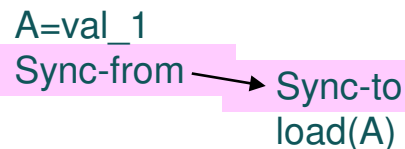
(based on MPI-3.0/3.1, MPI_Win_allocate_shared, page 410/408, lines 16-20/43-47: “**A consistent view ...**”)



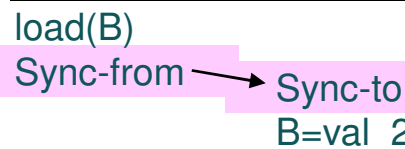
and the lock on process 0 is granted first

and having ...

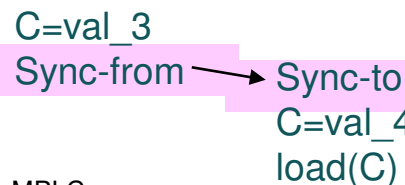
then it is **guaranteed** that ...



⇒ ... the load(A) in P1 loads val_1
 (this is the write-read-rule)



⇒ ... the load(B) in P0 is not affected by the store of val_2 in P1
 (read-write-rule)



⇒ ... that the load(C) in P1 loads val_4
 (write-write-rule)

See next slide

¹⁾ Must be paired according to the general on-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 11.21, but also with some synchronization through MPI shared memory loads and stores, e.g. with C++11 atomic loads and stores).

³⁾ No rule for MPI_Win_flush (according current forum discuss.)

skipped

“Any-process-sync” & MPI_Win_sync 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
<code>MPI_WIN_LOCK_ALL(MPI_MODE_NOCHECK,win)</code>	<code>MPI_WIN_LOCK_ALL(MPI_MODE_NOCHECK,win)</code>
<code>DO ...</code>	<code>DO ...</code>
<code>X=...</code>	
<code>MPI_F_SYNC_REG(X)¹⁾</code>	
<code>MPI_WIN_SYNC(win)</code>	
<code>MPI_Send</code>	
	<code>MPI_Recv</code>
	<code>MPI_WIN_SYNC(win)</code>
	<code>MPI_F_SYNC_REG(X)¹⁾</code>
	<code>local_tmp = X</code>
	<code>MPI_F_SYNC_REG(X)¹⁾</code>
	<code>MPI_WIN_SYNC(win)</code>
	<code>MPI_Send</code>
	<code>print local_tmp</code>
<code>MPI_Recv</code>	
<code>MPI_WIN_SYNC(win)</code>	
<code>MPI_F_SYNC_REG(X)¹⁾</code>	
<code>END DO</code>	<code>END DO</code>
<code>MPI_WIN_UNLOCK_ALL(win)</code>	<code>MPI_WIN_UNLOCK_ALL(win)</code>

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

- 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**.

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

- See also MPI-3.1, Section 11.8, Example 11.21 on pages 468-469.

¹⁾ Fortran only.

→ See Exercise 3

.....



Thank you for your interest –
any questions?

Appendix

Abstract

As a long-standing member of the MPI forum, I try to sketch my special way through the times of this standardization body, which also lead to become the publisher of the MPI books. From the very first, I was involved in the MPI-Fortran nightmare. At the end, we significantly enhanced the existing MPI module and added the new `mpi_f08` module, which is the first one that is fully consistent with the Fortran standard. Having the MPI standard is nothing without good libraries, but having such libraries is nothing if the users do not use them. For that, I tried to develop a complete MPI course that includes all the new MPI-3.0 and MPI-3.1 methods, which were developed to better serve the needs of the parallel computing user community, including better platform and application support. My own special interest here is the new MPI-3 shared memory interface.