History and Development of the MPI Standard

Jesper Larsson Träff
Vienna University of Technology
Faculty of Informatics, Institute of Information Systems
Research Group Parallel Computing
Favoritenstrase 16, 1030 Wien
www.par.tuwien.ac.at

www.mpi-forum.org

21. September, 2012, MPI Forum meeting in Vienna: MPI 3.0 has just been released …

… but MPI has a long history and it is instructive to look at that

„Those who cannot remember the past are condemned to repeat it", George Santyana, The Life of Reason, 1905-1906

"History always repeats itself twice: first time as tragedy, second time as farce", Karl Marx

"History is Written By the Winners", George Orwell, 1944 (but he quotes from someone else)

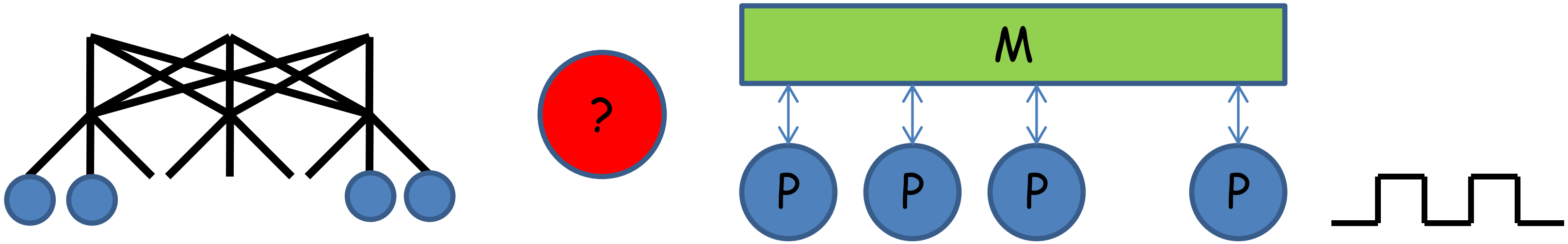
Last quote:
„history“ depends. Who tells it, and why? What informations is available? What‘s at stake?

My stake:

- Convinced of MPI as a well designed and extremely useful standard, that has posed productive research/development problems, with a broader parallel computing relevance
- Critical of current standardization effort, MPI 3.0

- MPI implementer, 2000-2010 with NEC
- MPI Forum member 2008-2010 (with Hubert Ritzdorf, representing NEC)
- Voted „no“ to MPI 2.2

A long debate: shared-memory vs. distributed memory



Question: What shall a parallel machine look like?

causing debate since (at least) the 70ties, 80ties

Answer depends
- What are your concerns?
- What is desirable?
- What is feasible?

Hoare/Dijkstra:
Parallel programs shall be structured as collections of communicating, sequential processes

Their concern: CORRECTNESS

And, of course, PERFORMANCE: many, many practitioneers

Wyllie, Vishkin:
A parallel algorithm is like a collection of synchronized sequential algorithms that access a common shared memory, and the machine is a PRAM

Their concern: (asymptotic) PERFORMANCE

And, of course, CORRECTNESS: Hoare semantics

Hoare/Dijkstra:
Parallel programs shall be structured as collections of communicating, sequential processes

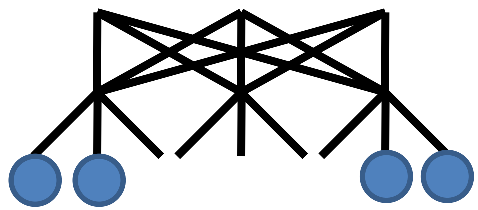
[C. A. R. Hoare: Communicating Sequential Processes. Comm. ACM 21(8): 666-677, 1978]

Wyllie, Vishkin:
A parallel algorithm is like a collection of synchronized sequential algorithms that access a common shared memory, and the machine is a PRAM

[Fortune, Wyllie: Parallelism in Random Access Machines. STOC 1978: 114-118]

[Shiloach, Vishkin: Finding the Maximum, Merging, and Sorting in a Parallel Computation Model. Jour. Algorithms 2(1): 88-102, 1981]

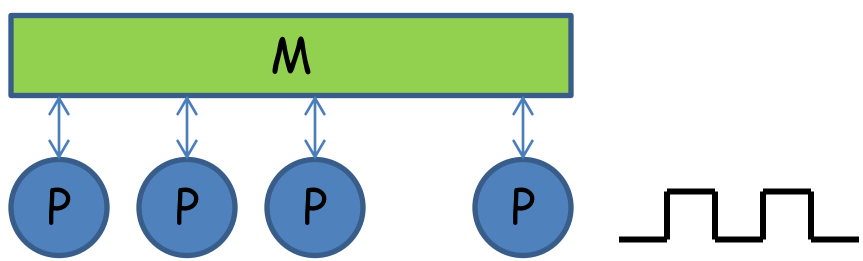
Hoare/Dijkstra:
Parallel programs shall be structured as collections of communicating, sequential processes



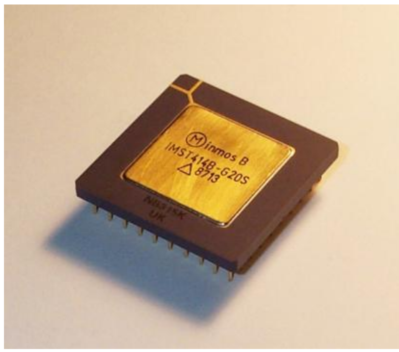
Neither perhaps cared too much about how to build machines…

Wyllie, Vishkin: (many, many practiotioneers, Burton-Smith, …)
A parallel algorithm is like a collection of synchronized sequential algorithms that access a common shared memory, and the machine is a PRAM

Neither perhaps cared too much about how to build machines (in the beginning)

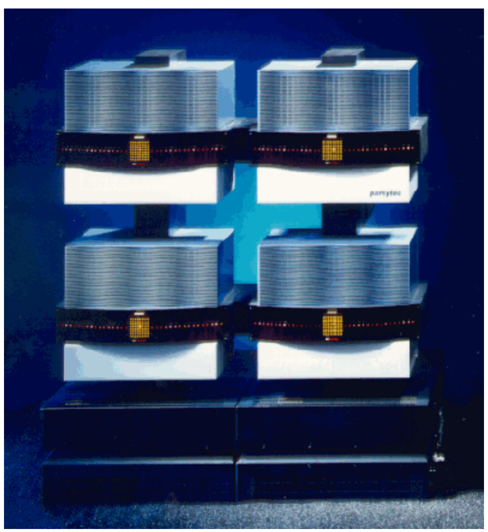


…but others (furtunately) did



The INMOS transputer T400, T800, from ca. 1985

A complete architecture entirely based on the CSP idea. An original programming language, OCCAM (1983, 1987)



Parsytec (ca. 1988-1995)

Intel iPSC/2 ca. 1990

Intel Paragon, ca. 1992

Thinking machines
CM5, ca. 1994

IBM SP/2 ca. 1996

Thinking Machines (1982-94) CM2, CM5

MasPar (1987.1996)
MP2

KSR 2, ca. 1992

HEP Denelcor,
mid 1980ties

Ironically…

Despite algorithmically stronger properties and potential for scaling to much, much larger numbers of processors of shared-memory models (like the PRAM)

than (say) OpenMP

practically, high-performance systems with (quite) substantial parallelism have all been distributed-memory systems

and the corresponding de facto standard – MPI (the Message-Passing Interface) is much stronger

Sources of MPI: the early years

Early 90ties fruitful years for practical parallel computing (funding for „grand challenge“ and „star wars“)

Commercial vendors and national laboratories (including many European) needed practically working programming support for their machines and applications

Vendors and labs proposed and maintained own languages, interfaces, libraries for parallel programming (early 90ties)

- Intel NX, Express, Zipcode, PARMACS, IBM EUI/CCL, PVM, P4, OCCAM, …

- Intel NX, Express, Zipcode, PARMACS, IBM EUI/CCL, PVM, P4, OCCAM, …

intended for distributed memory machines, and centered around similar concepts

Similar enough to warrant an effort towards creating a common standard for message-passing based parallel programming

Portability problem: wasted effort in maintaining own interface for small user group, lack of portability across systems

Message-passing interfaces/languages early 90ties

- Intel NX: send-receive message passing (non-blocking, buffering?), no tags(?), no group concept, no collectives weak encapsulation

- IBM EUI: point-to-point and collectives (more than in MPI), group concept, high performance [Snir et al.]

- Zipcode/Express: point-to-point, emphasis on library building [Skjellum]

- PARMACS/Express: point-to-point, topological mapping [Hempel]

- PVM: point-to-point communication, some collective, virtual machine abstraction, fault-tolerance

Some odd men out

- Linda: tuple space get/put – a first PGAS approach?
- Active messages; seems to presuppose an SPMD model?
- OCCAM: too strict CSP-based, synchronous message passing?
- PVM: heterogeneous systems, fault-tolerance, …

[Hempel, Hey, McBryan, Walker: Special Issue – Message Passing Interfaces. Parallel Computing 29(4), 1994]

Standardization: the MPI Forum and MPI 1.0

A standardization effort was started early 1992; key Dongarra, Hempel, Hey, Walker

Goal: to come out within a few years time frame with a standard for message-passing parallel programming; building on lessons learned from existing interfaces/languages

- Not a research effort (as such)!
- Open to participation from all interested parties

[Hempel, Walker: The emergence of the MPI message passing standard for parallel computing. Computer Standards & Interfaces, 21: 51-62, 1999]

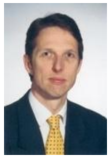
Key technical design points

MPI should encompass and enable

- Basic message-passing and related functionality (collective communication!)
- Enable library building: safe encapsulation of messages (and other things, eg. query functionality)
- High performance, across all available and future systems!
- Scalable design
- Support for C and Fortran

The MPI Forum

Not and ANSI/IEEE Standardization body, nobody „owns" the MPI standard; „free"



Open to participation for all interested parties; protocols open (votes, email discussions)

Regular meetings, 6-8 week intervals

Those who participates at meetings (with a history) can vote, one vote per organization (current discussion: quorum, semantics of abstaining)

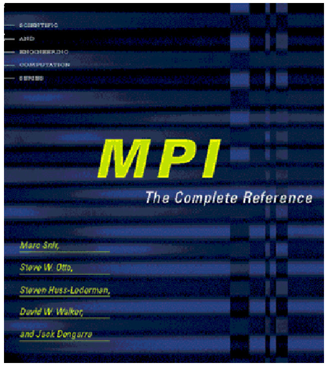
The 1st MPI Forum set out out to work early 1993

After 7 meetings, 1st version of the MPI Standard was ready early 1994. Two finalizing meetings in February 1994

MPI: A Message-Passing Interface standard. May 5th, 1994

The standard is the 226 page pdf-document that can be found at www.mpi-forum.org

as voted by the MPI Forum



Errata, minor adjustments: MPI 1.0, 1.1, 1.2: 1994-1995

Take note:

The MPI 1 standardization process was followed hand-in-hand by a(n amazingly good) prototype implementation: `mpich` from Argonne National Laboratory (Gropp, Lusk, …)

Other parties, vendors could build on this implementation (and did!), so that MPI was quickly supported on many parallel systems

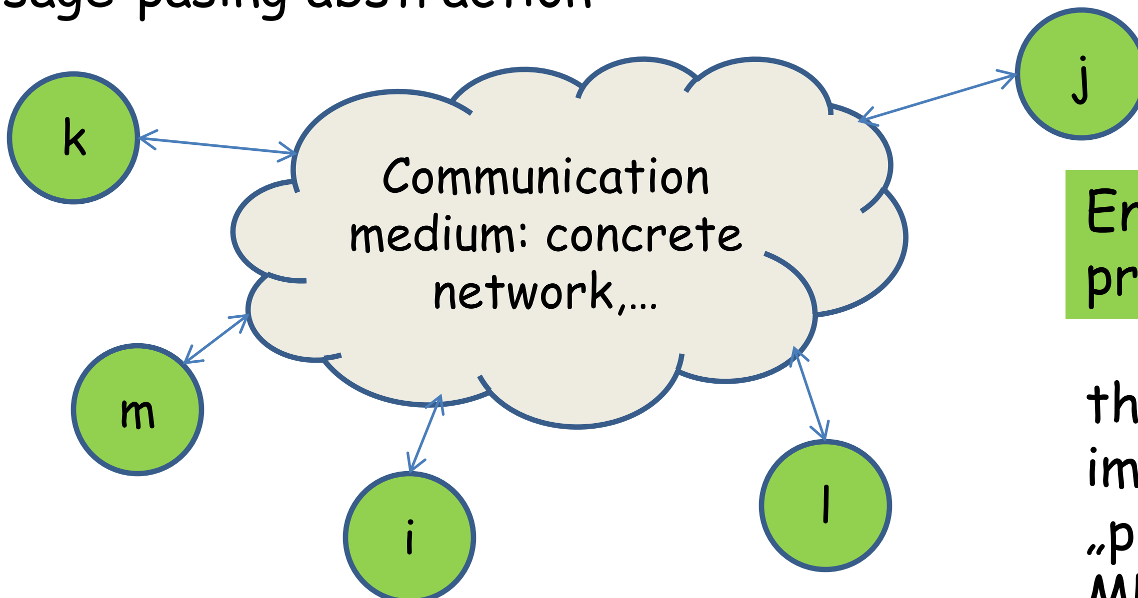
[W. Gropp, E. L. Lusk, N. E. Doss, A. Skjellum: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Parallel Computing 22(6): 789-828, 1996]

Why MPI has been successful: an appreciation

MPI made some fundamental

- abstractions, but is still close enough to common architectures to allow efficient, low overhead implementations („MPI is the assembler of parallel computing…“);
- is formulated with care and precision; but not a formal specification
- is complete (to a high degree), based on few, powerful, largely orthogonal key concepts (few exceptions, few optionals)
- and few mistakes

Message-pasing abstraction



Entities: MPI processes

that can be implemented as „processes" (most MPI implementations), „threads", …

can communicate through a communication medium

nature of which is of no concern to the MPI standard:

- No explicit requirements on network structure or capabilities
- No performance model or requirements

Basic message-pasing: point-to-point communication

MPI_Recv(&data,count,type,i,tag,comm,&status);

MPI_Send(&data,count,type,j,tag,comm);

Only processes in same communicator: ranked set of processes with unique „context" - can communicate

Fundamental library building concept: isolates communication in library routines from application communication

Basic message-pasing: point-to-point communication

MPI_Recv(&data,count,type,i,tag,comm,&status);

Receiving process blocks until data have been transferred

MPI_Send(&data,count,type,j,tag,comm);

MPI implementation must ensure reliable transmission; no time out (see RT-MPI)

Semantics: messages from same sender are delivered in order; possible to write fully deterministic programs

Basic message-pasing: point-to-point communication

MPI_Recv(&data,count,type,i,tag,comm,&status);

Receiving process blocks until data have been transferred

MPI_Send(&data,count,type,j,tag,comm);

Sending process may block or not… this is not synchronous communication (as in CSP; close to this, synchronous MPI_Ssend)

Semantics: upon return, data buffer can safely be reused

Basic message-pasing: point-to-point communication

MPI_Irecv(&data,count,type,i,tag,comm,&status);

Receiving process returns immediately, data buffer must not be touched

MPI_Isend(&data,count,type,j,tag,comm);

Non-blocking communication: MPI_Isend/MPI_Irecv
Explicit completion: MPI_Wait,...

Design principle: MPI specification shall not enforce internal buffering, all communication memory in user space...

Basic message-pasing: point-to-point communication

MPI_Irecv(&data,count,type,i,tag,comm,&status);

MPI_Isend(&data,count,type,j,tag,comm);

Receiving process returns immediately, data buffer must not be touched

Non-blocking communication: MPI_Isend/MPI_Irecv
Explicit completion: MPI_Wait,...

Design choice: No progress rule, communication will/must eventually happen

Basic message-pasing: point-to-point communication

MPI_Irecv(&data,count,type,i,tag,comm,&status);

MPI_Isend(&data,count,type,j,tag,comm);

Receiving process returns immediately, data buffer must not be touched

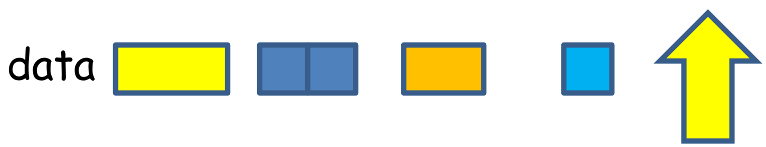
Completeness: MPI_Send, Isend, Issend, …, MPI_Recv, Irecv can be combined, semantics make sense

Basic message-pasing: point-to-point communication

MPI_Recv(&data,count,type,i,tag,comm,&status);

Receiving process blocks until data have been transferred

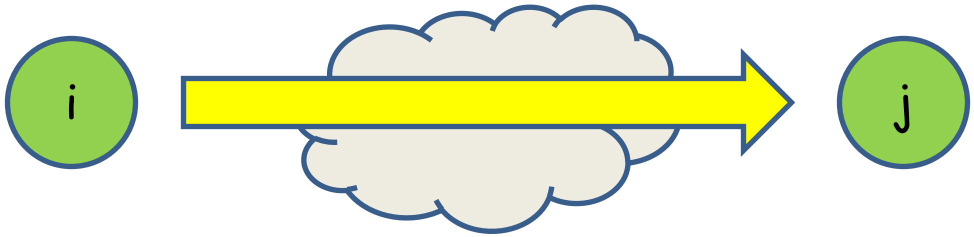
MPI_Send(&data,count,type,j,tag,comm);



MPI_Datatype describes structure of communication data buffer: basetypes MPI_INT, MPI_DOUBLE, …, and recursively applicable type constructors

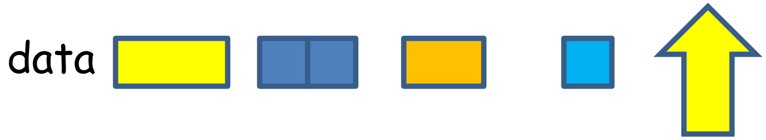
Basic message-pasing: point-to-point communication

MPI_Recv(&data,count,type,i,tag,comm,&status);



Receiving process blocks until data have been transferred

MPI_Send(&data,count,type,j,tag,comm);



Orthogonality: Any MPI_Datatype can be used in any communication operation

Semantics: only signature of data sent and data received must match (performance!)

Other functionality (supporting library building)

- Attributes to describe MPI objects (communicators, datatypes)
- Query functionality for MPI objects (MPI_Status)
- Errorhandlers to influence behavior on errors
- MPI_Group's for manipulating ordered sets of processes

Often heard objections/complaints

„MPI is too large"

„MPI is the assembler of parallel computing"...

and two answers

„MPI is designed not to make easy things easy, but to make difficult things possible"
Gropp, EuroPVM/MPI 2004

Conjecture (tested at EuroPVM/MPI 2002): for any MPI feature there will be at least one (significant) user depending essentially on exactly this feature

Collective communication: patterns of process communication

Fundamental, well-studied, and useful parallel communication patterns captured in MPI 1.0 as socalled collective operations:

- MPI_Barrier(comm);
- MPI_Bcast(…,comm);
- MPI_Gather(…,comm); MPI_Scatter(…,comm);
- MPI_Allgather(…,comm);
- MPI_Alltoall(…,comm);
- MPI_Reduce(…,comm); MPI_Allreduce(…,comm);
- MPI_Reduce_scatter(…,comm);
- MPI_Scan(…,comm);

Semantics: all processes in comm participates; blocking; no tags

Collective communication: patterns of process communication

Fundamental, well-studied, and useful parallel communication patterns captured in MPI 1.0 as socalled collective operations:

- MPI_Barrier(comm);
- MPI_Bcast(…,comm);
- MPI_Gather(…,comm); MPI_Scatter(…,comm);
- MPI_Allgather(…,comm);
- MPI_Alltoall(…,comm);
- MPI_Reduce(…,comm); MPI_Allreduce(…,comm);
- MPI_Reduce_scatter(…,comm);
- MPI_Scan(…,comm);

Completeness: MPI_Bcast dual of MPI_Reduce; MPI_Gather dual of MPI_Scatter. Regular and irregular (vector) variants

Collective communication: patterns of process communication

Fundamental, well-studied, and useful parallel communication patterns captured in MPI 1.0 as socalled collective operations:

- MPI_Gatherv(…,comm); MPI_Scatterv(…,comm);
- MPI_Allgatherv(…,comm);
- MPI_Alltoallv(…,comm);

- MPI_Reduce_scatter(…,comm);

Completeness: MPI_Bcast dual of MPI_Reduce; MPI_Gather dual of MPI_Scatter. Regular and irregular (vector) variants

Collective communication: patterns of process communication

Fundamental, well-studied, and useful parallel communication patterns captured in MPI 1.0 as socalled collective operations

Collectives capture complex patterns, often with non-trivial algorithms and implementations: delegate work to library implementer, save work for the application programmer

Obligation: MPI implementation must be of sufficiently high quality – otherwise application programmer will not use or implement own collectives

This did happen! For datatypes: unused for a long time

Collective communication: patterns of process communication

Fundamental, well-studied, and useful parallel communication patterns captured in MPI 1.0 as socalled collective operations

Collectives capture complex patterns, often with non-trivial algorithms and implementations: delegate to library implementer, save work for the application programmer

Completeness: MPI makes it possible to (almost) implement MPI collectives „on top of" MPI point-to-point communication

Some exceptions for reductions, MPI_Op; datatypes

Collective communication: patterns of process communication

Fundamental, well-studied, and useful parallel communication patterns captured in MPI 1.0 as socalled collective operations

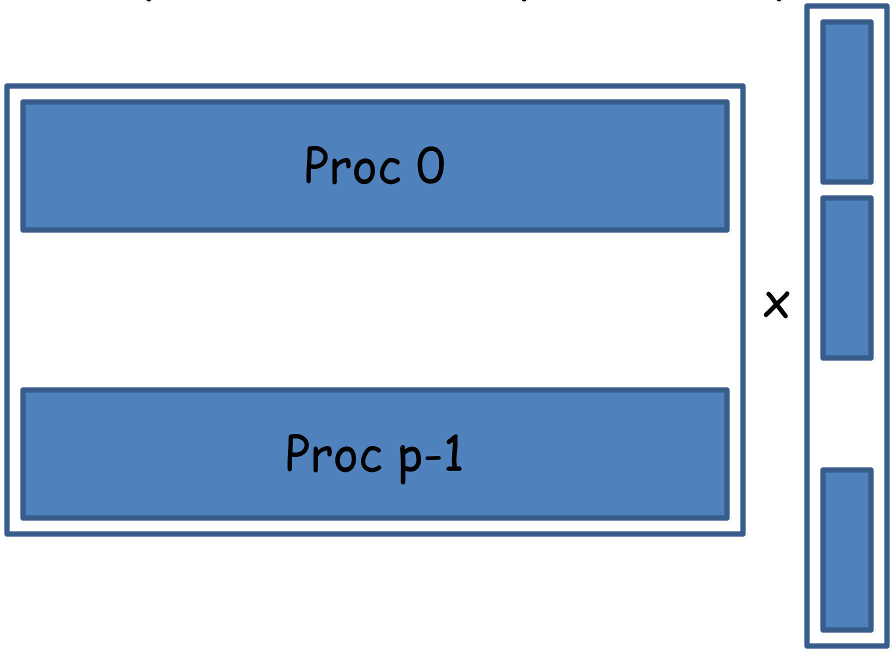
Collectives capture complex patterns, often with non-trivial algorithms and implementations: delegate to library implementer, save work for the application programmer

Conjecture: well-implemented collective operations contributes significantly towards application „performance portability“

[Träff, Gropp, Thakur: Self-Consistent MPI Performance Guidelines. IEEE TPDS 21(5): 698-709, 2010]

Three algorithms for matrix-vector multiplication

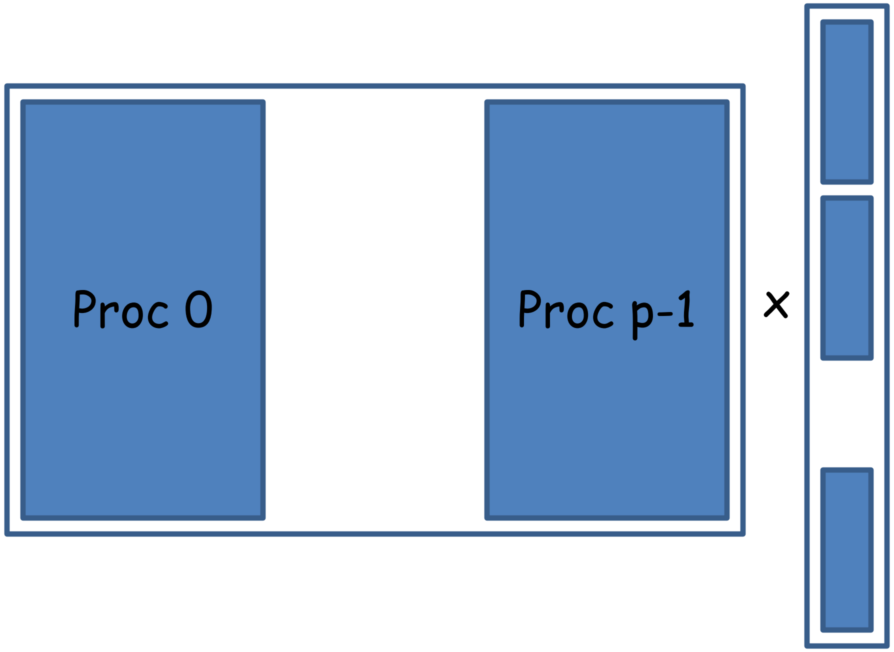
nxm matrix A and m-element vector y distributed evenly across p MPI processes: compute z = Ay



Algorithm 1:

- Row-wise matrix distribution
- Each process needs full vector: MPI_Allgather(v)
- Compute blocks of result vector locally

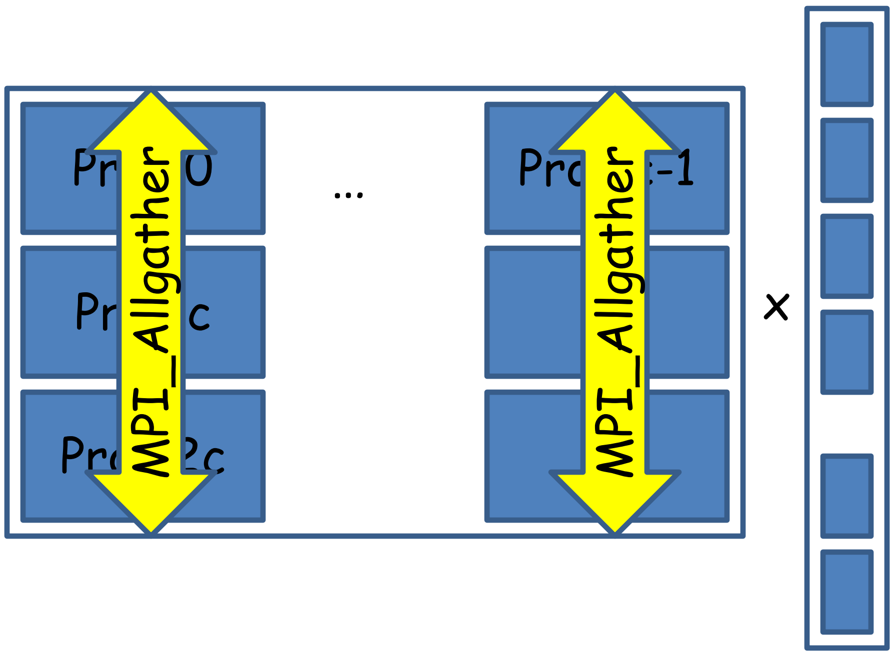
Three algorithms for matrix-vector multiplication



Algorithm 2:

- Column-wise matrix distribution
- Compute local partial result vector
- MPI_Reduce_scatter to sum and distribute partial results

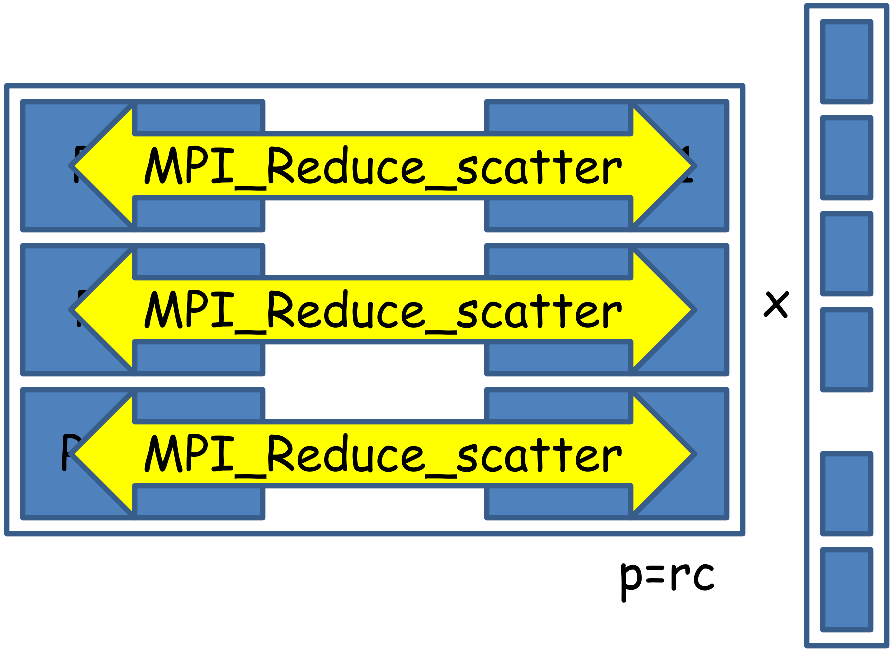
Three algorithms for matrix-vector multiplication



- <u>Algorithm 3</u>:
- Matrix distribution into blocks of n/r x m/c elements
- Algorithm 1 on columns
- Algorithm 2 on rows

Three algorithms for matrix-vector multiplication

Algorithm 3 is more scalable. Partitioning the set processes (new communicators) is essential!



- Algorithm 3:
- Matrix distribution into blocks of n/r x m/c elements
- Algorithm 1 on columns
- Algorithm 2 on rows

Interfaces that do support collectives on subsets of processes are not able to express Algorithm 3: case in point UPC

Three algorithms for matrix-vector multiplication

For the „regular" case where p divides n (and p=rc)
•Regular collectives: MPI_Allgather, MPI_Reduce_scatter

For the „irregular" case
•Irregular collectives: MPI_Allgatherv, MPI_Reduce_scatter

MPI 1.0 defined regular/irregular versions – completeness – for all the considered collective patterns; except for MPI_Reduce_scatter

Performance: irregular subsume regular counterparts; but much better algorithms are known for the regular ones

A lesson: Dense Linear Algebra and (regular) collective communication as offered by MPI go hand in hand

[F. G. van Zee, E. Chan, R. A. van de Geijn, E. S. Quintana-Ortí, G. Quintana-Ortí: The libflame Library for Dense Matrix Computations. Computing in Science and Engineering 11(6): 56-63 (2009)]

[R. A. van de Geijn, J. Watts: SUMMA: scalable universal matrix multiplication algorithm. Concurrency - Practice and Experience 9(4): 255-274 (1997)]

[Ernie Chan, Marcel Heimlich, Avi Purkayastha, Robert A. van de Geijn: Collective communication: theory, practice, and experience. Concurrency and Computation: Practice and Experience 19(13): 1749-1783 (2007)]

Note: Most of these collective communication algorithms are a factor 2 off from best possible

Another example: Integer (bucket) sort

n integers in a given range [0,R-1], distributed evenly across p MPI processes: m= n/p integers per process

A = | 0 1 3 0 0 2 0 1 … | |

$$B = \begin{bmatrix} 4 \\ 2 \\ 1 \\ 3 \end{bmatrix}$$

Step 1: bucket sort locally, let B[i] number of elements with key i

Step 2: MPI_Allreduce(B,AllB,R,MPI_INT,MPI_SUM,comm);

Step 3: MPI_Exscan(B,RelB,R,MPI_INT,MPI_SUM,comm);)

Now: Element A[j] needs to go to position AllB[A[j]-1]+RelB[A[j]]+j

Another example: Integer (bucket) sort

n integers in a given range [0,R-1], distributed evenly across p MPI processes: m= n/p integers per process

A = | 0 1 3 0 0 2 0 1 … | | B = (4, 2, 1, 3)

Step 4: compute number of elements to be sent to each other process, sendelts[i], i=0,…,p-1

Step 5:
MPI_Alltoall(sendelts,p,MPI_INT,recvelts,p,MPI_INT,comm);

Step6: redistribute elements
MPI_Alltoallv(A,sendelts,sdispls,…,comm);

Another example: Integer (bucket) sort

The algorithm is stable → Radixsort

Choice of radix R depends on properties of network (fully connected, fat tree, mesh/torus, …) and quality of reduction/scan-algorithms

The algorithm is portable (by virtue of the MPI collectives), but tuning depends on systems – concrete performance model needed, but this is outside scope of MPI

Note: on strong network T(MPI_Allreduce(m)) = O(m+log p)

NOT: O(mlog p)

A last feature

Process topologies:

Specify application communication pattern (as either a directed graph, or Cartesian grid) to MPI library, let library assign processes to processors so as to improve communication follwing specified pattern

MPI version: collective communicator construction functions, process ranks in new communicator represent new (improved) mapping

And a very last: (simple) tool building support – the MPI profiling interface

The mistakes

- MPI_Cancel(), semantically ill-defined, difficult to implement; a concession to RT?
- MPI_Rsend(); vendors got too much leverage?
- Pack/unpack; was added as an afterthough in last 1994 meetings (functionality is useful/needed, limitations in specification)
- Some functions enforce full copy of argument (list)s into library

Missing functionality

- Datatype query functions – not possible to query/reconstruct structure specified by given datatype

- Some MPI objects are not first class citizens (MPI_Aint, MPI_Op, MPI_Datatype); makes it difficult to build certain types of libraries

- Reductions cannot be performed locally

Is MPI scalable?

Question must distinguish between specification and implementation

Definition:

An MPI construct is non-scalable, if memory or time overhead(*) is $\Omega(p)$, p number of processes

(*)cannot be accounted for in application

Questions:

- Are there aspects of the MPI specification that are non-scalable (forces $\Omega(p)$ memory or time)?
- Are there aspects of (typical) MPI implementations that are non-scalable

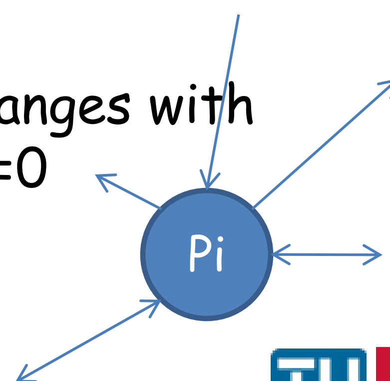
Answer is "yes" to both questions

Example:
Irregular collective alltoall communication (each process exchange some data with each other process)

```
MPI_Alltoallw(sendbuf,sendcounts[],senddispl[],sendtypes[],
              recvbuf,recvcounts[],recvdispls[],recvtypes[],…)
```

takes 6 p-sized arrays (4- or 8-byte integers) ~ 5MBytes, 10% of memory on BlueGene/L

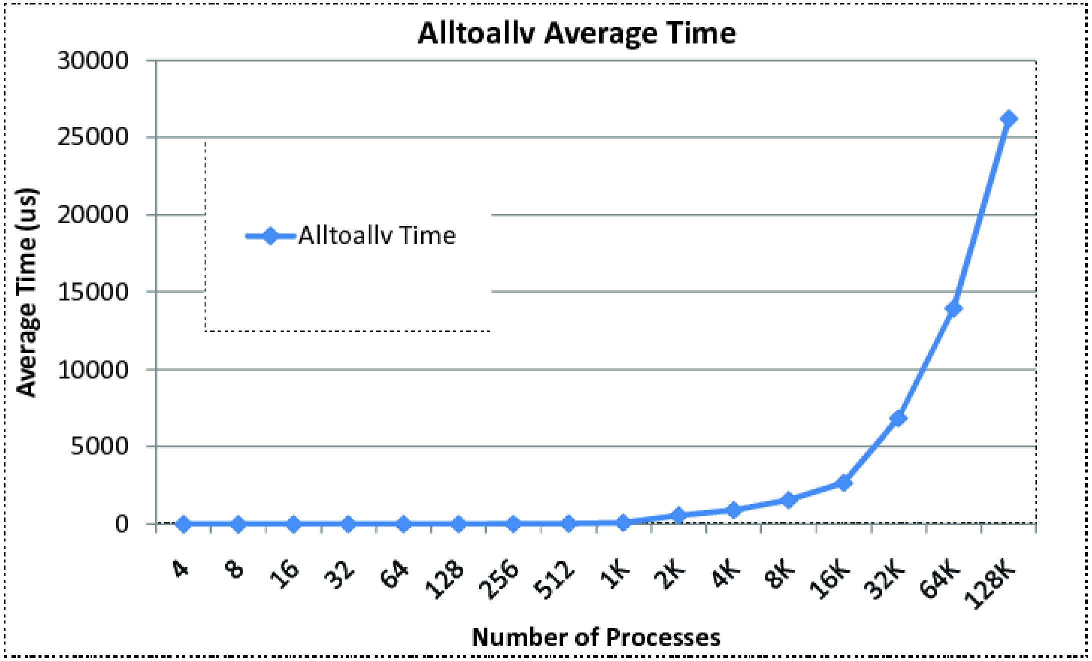
Sparse usage pattern: often each process exchanges with only few neighbors, so most send/recvcounts[i]=0



→ MPI_Alltoallw is non-scalable

Experiment:
sendcounts[i]=0, recvcounts[i] =0 for all processes and all i

Entails no communication



Argonne Natl. Lab
BlueGene/L

[Balaji, …, Träff : MPI on millions of cores. Parallel Processing Letters 21(1): 45-60, 2011]

Definitely non-scalable features in MPI 1.0

- Irregular collectives: p-sized lists of counts, displacements, types
- Topology interface: requires specification of full process topology by all processes

MPI 2: what (almost) went wrong

A number of issues/desired functionality were left open by MPI 1.0, either because of

- no agreement
- deadline, desire to get a consolidated standard out in time

Major open issues

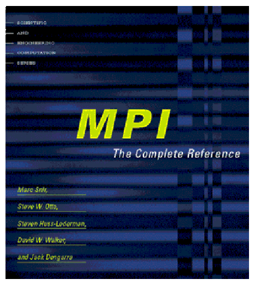
- Parallel IO
- One-sided communication
- Dynamic process management

were partly described in the socalled JOD: „Journal of Development“ (see www.mpi-forum.org)

MPI Forum started to reconvene already in 1995

Between 1995-1997 there were 16 meetings which lead to MPI 2.0

MPI 1.0: 226 pages



MPI 2.0: 356 additional pages

Major new features, with new concepts: extended message passing models

1. Dynamic process management
2. One-sided communication
3. MPI-IO

1. Dynamic process management:

MPI 1.0 was completely static: a communicator cannot change (design principle: no MPI object can change; new objects can be created and old ones destroyed), so the number of processes in MPI_COMM_WORLD cannot change: therefore not possible to add or remove processes from a running application

MPI 2.0 process management relies on inter-communicators (from MPI 1.0) to establish communication with newly started processes or already running applications

- MPI_Comm_spawn
- MPI_Comm_connect/MPI_Comm_accept
- MPI_Intercomm_merge

1. Dynamic process management:

MPI 1.0 was completely static: a communicator cannot change (design principle: no MPI object can change; new objects can be created and old ones destroyed), so the number of processes in MPI_COMM_WORLD cannot change: therefore not possible to add or remove processes from a running application

What if a process (in a communicator) dies? The fault-tolerance problem

Most (all) MPI implementations also die – but this may be an implementation issue

1. Dynamic process management:

MPI 1.0 was completely static: a communicator cannot change (design principle: no MPI object can change; new objects can be created and old ones destroyed), so the number of processes in MPI_COMM_WORLD cannot change: therefore not possible to add or remove processes from a running application

What if a process (in a communicator) dies? The fault-tolerance problem

If implementation does not die, it might be possible to program around/isolate faults using MPI 1.0 error handlers and inter-communicators

[W. Gropp, E. Lusk: Fault Tolerance in Message Passing Interface Programs. IJHPCA 18(3): 363-372, 2004]

1. Dynamic process management:

MPI 1.0 was completely static: a communicator cannot change (design principle: no MPI object can change; new objects can be created and old ones destroyed), so the number of processes in MPI_COMM_WORLD cannot change: tehrefore not possible to add or remove processes from a running application

What if a process (in a communicator) dies? The fault-tolerance problem

The issue is contentious&contagious…

2. One-sided communication

Motivations/arguments:

- Expressivity/convenience: For applications where only one process may readily know with which process to communicate data, the point-to-point message-passing communication model may be inconvenient

- Performance: On some architectures point-to-point communication could be inefficient; e.g. if shared-memory is available

Challenge: define a model that captures the essence of one-sided communication, but can be implemented without requiring specific hardware support

2. One-sided communication

Challenge: define a model that captures the essence of one-sided communication, but can be implemented without requiring specific hardware support

New MPI 2.0 concepts: communication window, communication epoch

MPI one-sided model cleanly separates communication from synchronization; three specific synchronization mechanisms
- MPI_Win_fence
- MPI_Win_Start/Complete/Post/Wait
- MPI_Win_lock/unlock

with cleverly thought out semantics and memory model

2. One-sided communication

MPI one-sided model cleanly separates communication from synchronization; three specific synchronization mechanisms

- MPI_Win_fence
- MPI_Win_Start/Complete/Post/Wait
- MPI_Win_lock/unlock

with cleverly thought out semantics and memory model

Unfortunately, application programmers did not seem to like it

- „too complicated“
- „not efficient“
- …

3. MPI-IO

Communication with external (disk/file) memory. Could leverage MPI concepts and implementations:

- Datatypes to describe file structure
- Collective communication for utilizing local file systems
- Fast communication

MPI datatype mechanism is essential, and the power of this concept starts to become clear

MPI 2.0 introduces (inelegant!) functionality to decode a datatype = discover the structure described by datatype. Needed for MPI-IO implementation (on top of MPI) and supports library building

Take note:

Apart from MPI-IO (ROMIO), the MPI 2.0 standardization process was not followed by prototype implementations

New concept (IO only): split collectives

Not discussed:

Thread-support/compliance, the ablity of MPI to work in a threaded environment

•MPI 1.0: a recommendation that MPI implementations be thread safe

•MPI 2.0: level of thread support can be requested and queried; an MPI library is not required to support the requested level, but returns information on the highest smaller level supported

MPI_THREAD_SINGLE
MPI_THREAD_FUNNELED
MPI_THREAD_SERIALIZED
MPI_THREAD _MULTIPLE

Quit years: 1997-2006

No standardization activity from 1997

Implementations evolved and improved; MPI was an interesting topic to work on, good MPI work was/is acceptable to all parallel computing conferences (SC, IPDPS, ICPP, Euro-Par, PPoPP, SPAA)

MPI 2.0 implementations

- Fujitsu (claim) 1999
- NEC 2000
- mpich 2004
- OpenMPI 2006(?)
- …

[J. L.Träff, H. Ritzdorf, R. Hempel: The Implementation of MPI-2 One-Sided Communication for the NEC SX-5. SC 2000]

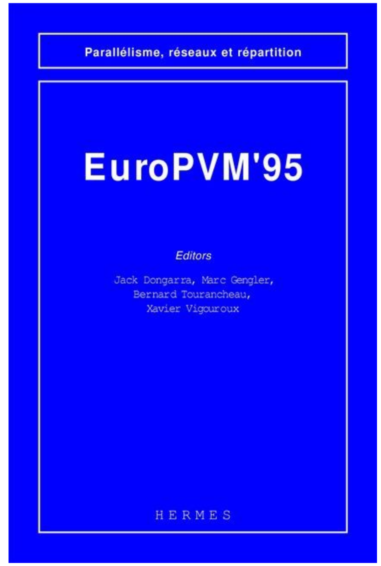
Ca. 2006 most/many implementations support mostly full MPI 2.0

Euro(PVM/)MPI conference series: dedicated to MPI

2012	Wien, Austria	
2011	Santorini, Greece	
2010	Stuttgart, Germany	EuroMPI (no longer PVM)
2009	Helsinki, Finland	EuroPVM/MPI
2008	Dublin, Ireland	
2007	Paris, France	
2006	Bonn, Germany	
2005	Sorrento, Italy	
2004	Budapest, Hungary	
2003	Venice, Italy	
2002	Linz, Austria	
2001	Santorini, Greece (9.11 – did not actually take place)	
2000	Balatonfüred, Hungary	
1999	Barcelona, Spain	
1998	Liverpool, UK	
1997	Cracow, Poland	Now EuroPVM/MPI
1996	Munich, Germany	
1995	Lyon, France	
1994	Rome, Italy	EuroPVM

MPI Forum meetings (2008–2012)

…biased towards MPI implementation



[Thanks to Xavier Vigouroux, Vienna 2012]

Bonn 2006: discussions („Open Forum“) on restarting MPI Forum starting

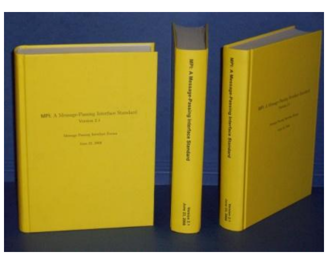
The MPI 2.2 – MPI 3.0 process

Late 2007: MPI Forum reconvenes

Consolidate standard: MPI 1.2 and MPI 2.0 into single standard document: MPI 2.1 (Sept. 4th, 2008)

MPI 2.2 intermediate step towards 3.0

- Address scalability problems
- Missing functionality
- BUT preserve backwards compatibility



586 pages

623 pages

Some MPI 2.2 features

- Addressing scalability problems: new topology interface, application communication graph is specified in a distributed fashion

  [T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, J. L. Träff: The scalable process topology interface of MPI 2.2. Concurrency and Computation: Practice and Experience 23(4): 293-310, 2011]

- Library building: MPI_Reduce_local

- Missing function: regular MPI_Reduce_scatter_block

- More flexible MPI_Comm_create (MPI 3.0: MPI_Comm_create_group)

- New datatypes, e.g. MPI_AINT

Some MPI 2.2 features

Fortran bindings modernized and corrected

C++ bindings (since MPI 2.0) deprecated! With the intention that they will be removed

MPI_Op, MPI_Datatype still not first class citizens (datatype support is weak and cumbersome)

MPI 2.1, 2.2, and 3.0 process

- 6 meetings 2012
- 6 meetings 2011
- 7 metings 2010

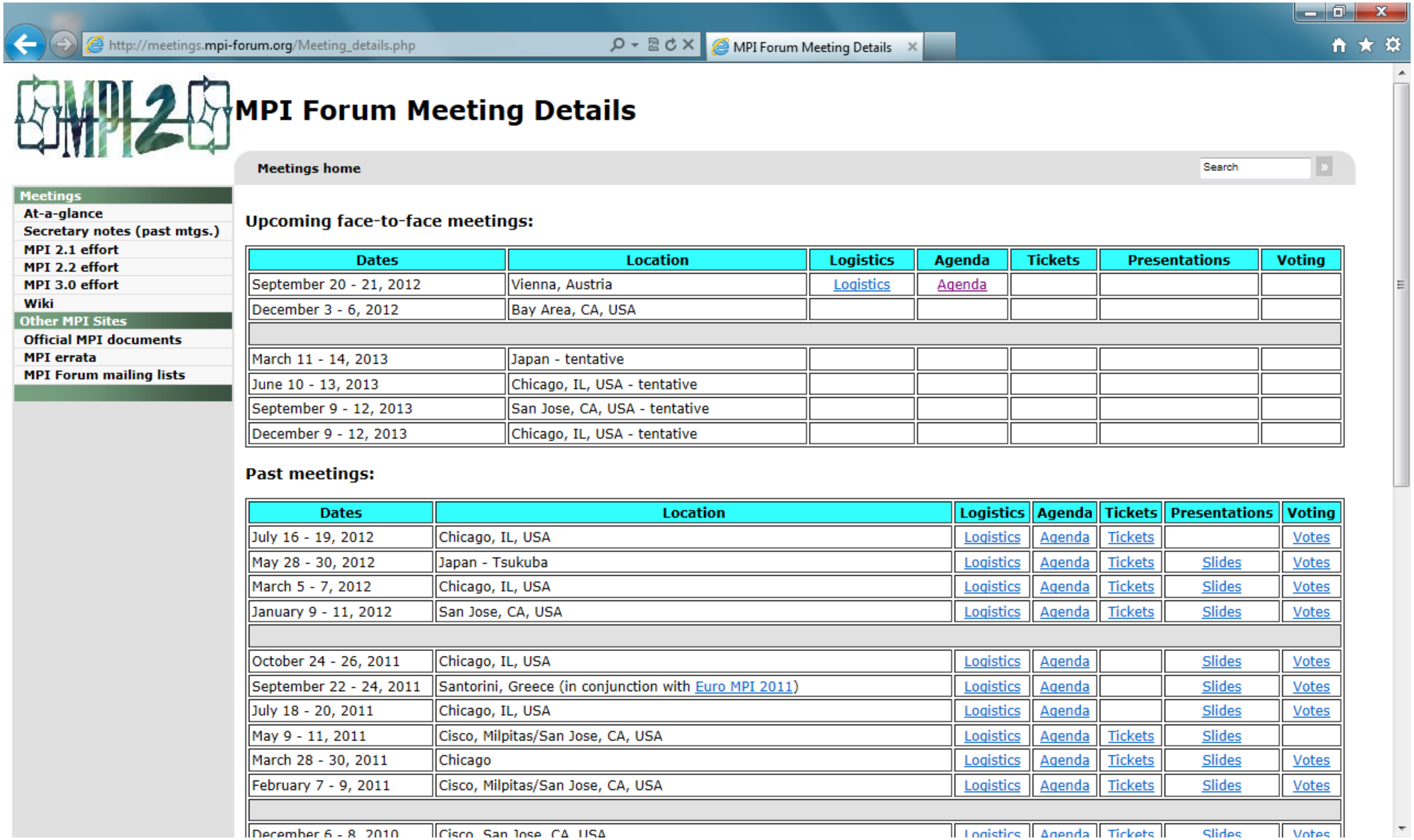
- 6 meetings 2009
- 7 meetings 2008

Recall:
- MPI 1: 7 meetings
- MPI 2.0: 16 meetings

Total: 32 meetings (and counting…)

MPI Forum rules: presence at physical meetings with a history (presence at past two meetings) required to vote

Requirement: new functionality must be supported by use-case and prototype implementation; backwards compatibility not strict


http://meetings.mpi-forum.org/Meeting_details.php

MPI Forum Meeting Details

MPI Forum Meeting Details

Meetings home

Meetings
- At-a-glance
- Secretary notes (past mtgs.)
- MPI 2.1 effort
- MPI 2.2 effort
- MPI 3.0 effort
- Wiki

Other MPI Sites
- Official MPI documents
- MPI errata
- MPI Forum mailing lists

Upcoming face-to-face meetings:

Dates	Location	Logistics	Agenda	Tickets	Presentations	Voting
September 20 - 21, 2012	Vienna, Austria	Logistics	Agenda			
December 3 - 6, 2012	Bay Area, CA, USA					
March 11 - 14, 2013	Japan - tentative					
June 10 - 13, 2013	Chicago, IL, USA - tentative					
September 9 - 12, 2013	San Jose, CA, USA - tentative					
December 9 - 12, 2013	Chicago, IL, USA - tentative					

Past meetings:

Dates	Location	Logistics	Agenda	Tickets	Presentations	Voting
July 16 - 19, 2012	Chicago, IL, USA	Logistics	Agenda	Tickets		Votes
May 28 - 30, 2012	Japan - Tsukuba	Logistics	Agenda	Tickets	Slides	Votes
March 5 - 7, 2012	Chicago, IL, USA	Logistics	Agenda	Tickets	Slides	Votes
January 9 - 11, 2012	San Jose, CA, USA	Logistics	Agenda	Tickets	Slides	Votes
October 24 - 26, 2011	Chicago, IL, USA	Logistics	Agenda		Slides	Votes
September 22 - 24, 2011	Santorini, Greece (in conjunction with Euro MPI 2011)	Logistics	Agenda		Slides	Votes
July 18 - 20, 2011	Chicago, IL, USA	Logistics	Agenda		Slides	Votes
May 9 - 11, 2011	Cisco, Milpitas/San Jose, CA, USA	Logistics	Agenda	Tickets	Slides	
March 28 - 30, 2011	Chicago	Logistics	Agenda	Tickets	Slides	Votes
February 7 - 9, 2011	Cisco, Milpitas/San Jose, CA, USA	Logistics	Agenda	Tickets	Slides	Votes
December 6 - 8, 2010	Cisco, San Jose, CA, USA	Logistics	Agenda	Tickets	Slides	Votes

MPI 2.2 – MPI 3.0 process had working groups on

- Collectives Operations
- Fault Tolerance
- Fortran bindings
- Generalized requests ("on hold")
- Hybrid Programming
- Point to point (this working group is "on hold")
- Remote Memory Access
- Tools
- MPI subsetting ("on hold")
- Backward Compatibility
- Miscellaneous Items
- Persistence

MPI 3.0: new features, new themes, new opportunities

MPI 3.0, 21. September 2012: 822 pages

Major new functionalities:

1. Non-blocking collectives
2. Sparse collectives
3. New one-sided communication

4. Performance tool support

Deprecated functions removed: C++ interface has gone

Implementation status: mpich should cover MPI 3.0

Performance/quality?

1. Non-blocking collectives

Introduced for performance (overlap) and convenience reasons

Similarly to non-blocking point-to-point routines; MPI_Request object to check and enforce progress

Sound semantics based on ordering, no tags

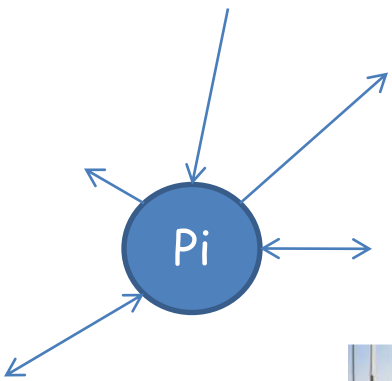
Different from point-to-point (with good reason): blocking and non-blocking collectives do not mix and match: MPI_Ibcast() is incorrect with MPI_Bcast()

Incomplete: non-blocking versions for some other collectives (MPI_Icomm_dup)
Non-orthognal: split and non-blocking collectives

2. Sparse collectives

Addresses scalability problem of irregular collectives.
Neighborhood specified with topology functionality

```
MPI_Neighbor_allgather(…,comm);
MPI_Neighbor_allgatherv(…,comm);
MPI_Neighbor_alltoall(…,comm);
MPI_Neighbor_alltoallv(…,comm);
MPI_Neighbor_alltoallw(…,comm);
```



and corresponding non-blocking versions

Will users take up? Optimization potential?

[T. Hoefler, J. L. Träff: Sparse collective operations for MPI. IPDPS 2009]

3. One-sided communication

Model extension for better performance on hybrid/shared memory systems

Atomic operations (lacking in MPI 2.0 model)

Per operation local completion, MPI_Rget, MPI_Rput, … (but only for passive synchronization)

[Hoefler, Dinan, Buntinas, Balaji, Barrett, Brightwell, Gropp, Kale, Thakur: Leveraging MPI's one-sided communication for shared-memory programming. EuroMPI 2012, LNCS 7490, 133-141, 2012]

4. Performance tool support

Problem of MPI 1.0 allowing only one profiling interface at a time (linker interception of MPI calls) NOT solved

Functionality added to query certain internals of the MPI library

Will tool writers take up?

MPI at a turning point

Extremely large-scale systems now appearing stretch the scalability of MPI

Is MPI for exascale?

•Heterogeneous?
•memory constrained?
•low bisection width?
•unreliable?
systems

MPI Forum at a turning point

Attendance large enough?
Attendance broad enough?

More meetings, smaller attendance

MPI 2.1-MPI 3.0 process has been long and exhausting, attendance driven by implementors, relatively little input form users and applications, non-technical goals have played a role; research has been conducted but not lead to useful outcome for the standard (fault tolerance, thread/hybrid support, persistence, …)

Perhaps time to take a break?

Summary:

Study history and learn from it: how to do better than MPI

Standardization is a major effort, has taken a lot of dedication and effort from a relatively large (but declining?) group of people and institutions/companies

MPI 3.0 will raise many new implementation challenges

MPI 3.0 is not the end of the (hi)story