

MPI Evolution

Advanced Message-Passing Programming



Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Acknowledge EPCC as follows: “© EPCC, The University of Edinburgh, www.epcc.ed.ac.uk”

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

Overview

- History of the MPI Standard
 - Before MPI
 - MPI 1
 - MPI 2
- Present of the MPI Standard
 - MPI 3.0
- Future of the MPI Standard
 - MPI 3.1/4.0/Next
- MPI Implementations

Before MPI

- Before MPI, many competing message passing libraries.
 - Most computer vendors developed their own proprietary libraries.
 - There were also various portable libraries:
 - These targeted a variety of systems/interconnects.
 - Mostly developed by academic groups.
 - Usually only optimised for a small subset of the supported platforms.
- Different libraries used different models of communication
- This made application development very hard
 - Applications often needed their own communication module to encapsulate the different message passing systems.
- MPI was an attempt to define a standard set of communication calls.

MPI Forum

- Main web site at <http://mpi-forum.org/meetings/>
- The MPI Forum contains representatives from many of the vendors and academic library developers.
- This is one reason the specification is so large:
 - MPI supports many different models of communication, corresponding to various communication models supported by its predecessors.
- Much of the specification driven by the library developers.
 - leaves a lot of scope for optimised versions on different hardware.
 - many aspects of the MPI specification deliberately allow different implementations the freedom to work in different ways.
 - this makes it easy to port/optimize MPI for new hardware.
 - application developers need to be aware of this when writing code.
 - erroneous applications may work fine on one MPI implementation but fail using a different one.

Early History of MPI

- MPI is an “Application Programming Interface” (API) specification.
 - it is a specification not a piece of code.
 - there are many different implementations of the MPI specification.
- The MPI Standard is defined by the MPI Forum
 - Work started 1992
 - V 1.0 in 1994 – basic point-to-point, collectives, data-types, etc
 - V 1.1 in 1995 – fixes and clarifications to MPI 1.0
 - V 1.2 in 1996 – fixes and clarifications to MPI 1.1
 - V 1.3 in 1997 – refers to MPI 1.2 after combination with MPI-2.0
 - V 2.0 in 1997 – parallel I/O, RMA, dynamic processes, C++, etc

More recent History of MPI

- ... V 2.0 in 1997 – parallel I/O, RMA, dynamic processes, C++, etc
- Stable for 10 years
 - Version 2.1 in 2008 – fixes and clarifications to MPI 2.0
 - Version 2.2 in 2009 – small updates and additions to MPI 2.1
 - Version 3.0 in 2012 – neighbour collectives, unified RMA model, etc
 - Version 3.1 in 2015 – fixes, clarifications and additions to MPI 3.0

MPI-2 One-sided communication

- Separates data transmission from process synchronisation
- All comms parameters specified by a single process
- Definitions: “origin” calls MPI, memory accessed at “target”
- Initialise by creating a “window”
 - A chunk of local memory that will be accessed by remote processes
- Open origin “access epoch” (and target “exposure epoch”)
 - Communicate: MPI_Put, MPI_Get, MPI_Accumulate
 - Synchronise: passive target (or active target)
 - Use data that has been communicated
- Tidy up by destroying the window – MPI_Win_free

MPI 3.0

- Major new features
 - Non-blocking collectives, neighbourhood collectives
 - Improvements to one-sided communication
 - Added a new tools interface and new bindings for Fortran 2008
- Other new features
 - Matching Probe and Recv for thread-safe probe and receive
 - Non-collective communicator creation function
 - Non-blocking communication duplication function
 - “const” correct C language bindings
 - New MPI_Comm_split_type function
 - New MPI_Type_create_hindexed_block function
- C++ language bindings removed
 - previously deprecated functions removed

MPI 3.0 – Changes to collectives

- Non-blocking versions of all collective communication functions added
 - MPI_Ibcast, MPI_Ireduce, MPI_Iallreduce, etc
 - There is even a non-blocking barrier, MPI_Ibarrier
 - They return MPI_Request like other non-blocking functions
 - The user code must complete the operation with (one of the variants of) MPI_Test or MPI_Wait
 - Multiple non-blocking collectives can be outstanding but they must be called in the same order by all MPI processes
- New neighbourhood collective functions added
 - MPI_Neighbor_allgather and MPI_Neighbor_alltoall (plus variants)
 - Neighbours defined using a virtual topology, i.e. cartesian or graph
 - Extremely useful for nearest-neighbour stencil-based computations
 - Allow a scalable representation for common usage of MPI_Alltoally

MPI 3.0 – Changes to One-sided

- New window creation functions
 - New options for where, when and how window memory is allocated
- New atomic read-modify-write operations
 - MPI_Fetch_and_op and MPI_Compare_and_swap
- New “unified” memory model
 - Old one still supported, now called “separate” memory model
 - Simplifies memory consistency rules on cache-coherent machines
- New local completion semantics for one-sided operations
 - MPI_Rput, MPI_Rget and MPI_Raccumulate return MPI_Request
 - User can use MPI_Test or MPI_Wait to check for local completion

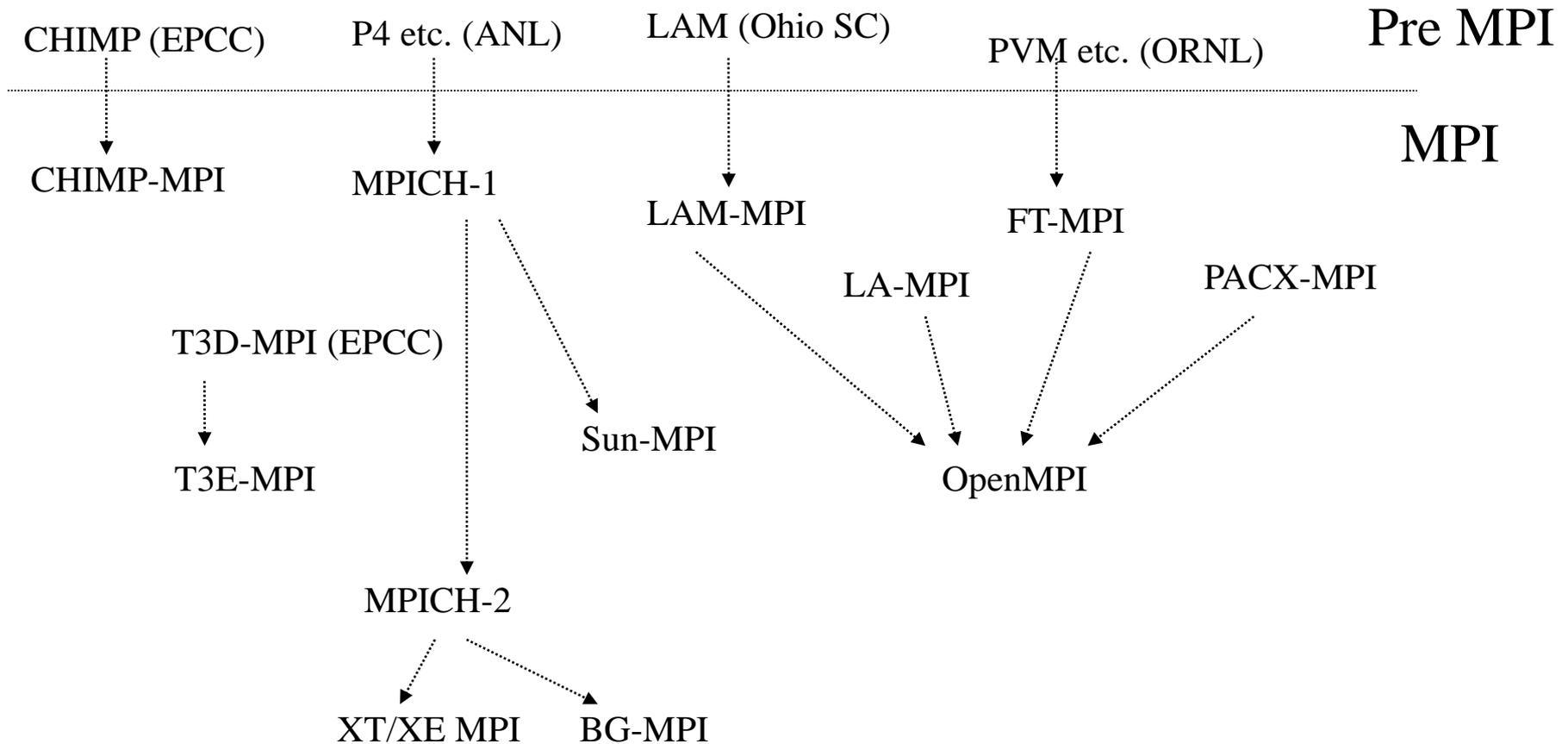
MPI Next: End-points and Fault-tolerance

- End-points proposal – improved support for hybrid programs
 - Allow threads to act like MPI processes
 - Allow multiple MPI ranks for a communicator in a single OS process
 - Example use-case: easier to map UPC thread id to MPI rank
- Fault-tolerance proposal – improved error-handling
 - Allow an MPI program to survive various types of failure
 - Node failure, communication link failure, etc
 - Notification: local process told particular operation will not succeed
 - Propagation: local knowledge of faults disseminated to global state
 - Consensus: vote for and agree on a common value despite failures
 - Low-level minimum functionality to support fault-tolerance libraries

MPI implementations

- There are many different implementations of the MPI specification.
- Many of the early ones were based on pre-existing portable libraries.
- Currently there are 2 main open source MPI implementations
 - MPICH
 - OpenMPI
- Many vendor MPI implementations are now based on these open source versions.

MPI family tree (partial)



MPICH

- Virtually the default MPI implementation
 - Mature implementation.
 - Good support for generic clusters (TCP/IP & shared memory).
 - Many vendor MPIs now based on MPICH.
- Original called MPICH (MPI-1 functionality only)
- Re-written from scratch to produce MPICH-2 (MPI-2)
- Incorporated MPI-3 and renamed back to MPICH again
- Ported to new hardware by implementing a small core ADI
 - ADI = Abstract Device Interface.
 - Full API has default implementation using the core ADI functions.
 - Any part can be overridden to allow for optimisation.

OpenMPI

- New MPI implementation
 - Joint project between developers of
 - FT-MPI
 - LA-MPI
 - LAM/MPI
 - PACX/MPI
- Very active project
 - Special emphasis on support for infiniband hardware
 - Special emphasis on Grid MPI
 - Fault tolerant communication
 - Heterogeneous communication
 - Current version supports MPI-3
 - Open Source project with large and varied community effort

Summary

- Most MPI implementations use a common “superstructure”
 - lots of lines of code dealing with a whole range of MPI issues: datatypes, communicators, argument checking, ...
 - will implement a number of different ways (protocols) of sending data
 - all hardware-specific code kept separate from the rest of the code, e.g. hidden behind an Abstract Device Interface
- To optimise for a particular architecture
 - rewrite low-level communication functions in the ADI
 - optimise the collectives especially for offload hardware
 - use machine-specific capabilities when advantageous
- Multi-core nodes
 - modern MPI libraries are aware of shared-memory nodes
 - already include optimisations to speed up node-local operations
 - uses multiple implementations of the same ADI in a single library