**MPI**

The Message Passing Interface (MPI) is an open library standard for distributed memory parallelization. The library API (Application Programmer Interface) specification is available for C and Fortran. There exist unofficial language bindings for many other programming languages, e.g. Python3, or JAVA 1, 2, 3. The first standard document was released in 1994. MPI has become the de-facto standard to program HPC cluster systems and is often the only way available. There exist many implementations, Open source and proprietary. The latest version of the standard is [**MPI** 3.1](https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf)(released in 2015).

MPI allows to write portable parallel programs for all kinds of parallel systems, from small shared memory nodes to petascale cluster systems. While many criticize its bloated API and complicated function interface no alternative proposal could win a significant share in the HPC application domain so far. There exist optimized implementations, open source and proprietary, for any HPC platform and architecture and a wealth of tools and libraries. Common implementations are [OpenMPI](https://www.open-mpi.org/" \t "_blank), [mpich](https://www.mpich.org/" \t "_blank) and [Intel MPI](https://software.intel.com/en-us/mpi-library). Because MPI is available for such a long time and almost any HPC application is implemented using MPI it is the safest bet for a solution that will be supported and stable on mid- to long-term future systems.

Information on how to run an existing MPI program can be found in the [How to Use MPI](https://hpc-wiki.info/hpc/How_to_Use_MPI) Section.

**API Overview**

The standard specifies interfaces to the following functionality (incomplete list):

* Point-to-point communication,
* Datatypes,
* Collective operations,
* Process groups,
* Process topologies,
* One-sided communication,
* Parallel file I/O.

**Point-to-Point communication**

Sending and receiving of messages by process is the basics MPI communication mechanism. P2P communication operation are send and receive.

**Syntax:**

MPI\_Send(**const** void\* buf, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm)

MPI\_Recv (void\* buf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Status\* status)

**Collective Operations**

MPI supports collective communication primitives for cases where all ranks are involved in communication.

There exist three types of collective communication:

* Synchronization (barrier)
* Data movement (e.g. gather, scatter, broadcast)
* Collective computation (reduction)

**Syntax:**

MPI\_Bcast(void\* buffer, int count, MPI\_Datatype datatype, int root, MPI\_Comm comm)

MPI\_BCAST broadcasts a message from the process with rank root to all processes of the communicator, itself included

MPI\_Reduce(**const** void\* sendbuf, void\* recvbuf, int count, MPI\_Datatype datatype, MPI\_Op op, int root, MPI\_Comm comm)

MPI\_REDUCE combines the elements provided in the input buffer of each process in the communicator, using the operation op, and returns the combined value in the output buffer of the process with rank root.