MPI - Message Passing Interface

MPI provides an API for performing distributed memory message passing in parallel programs. Supported programming languages are Fortran, C and C++, although others, like Java, Perl, R and Python can also be integrated. This document describes the common supported features of all MPI implementations used on the LRZ HPC systems, and provides documentation on the MPI standard.

- Introduction
- Standardization
- Parallel environments on LRZ systems
- Compiling and linking
- Special cases
- Executing MPI binaries
- How to start up MPI programs
- Startup mechanisms
  - SPMD mode
  - MPMD execution
  - Hybrid parallel programs
  - Environment variables
- MPI-2 and other special topics
- Troubleshooting MPI
- General MPI Documentation
  - Standard document
  - Off-site MPI information
  - Tutorials

Introduction

The message passing interface is at present the most heavily used paradigm for large-scale parallel programming. In particular, to fully exploit the capabilities for the specialized interconnects used in supercomputers, a number of proprietary MPI implementations are deployed at LRZ. A full list of available MPI environments is provided below.

Standardization

In order to guarantee portability as well as to allow vendors to produce well-optimized implementations, the interface is standardized. The most up-to-date release of the standard is Version 3.1. Basic functionality covered by MPI is

- point-to-point communication in blocking, nonblocking, buffered and unbuffered modes
- collective communication (e.g., all-to-all, scatter/gather, reduction operations)
- using basic and derived MPI data types
- and running on a static processor configuration

More advanced functionality (which may not be fully implemented by a given real-world implementation) is

- parallel I/O operations (MPI-IO)
- dynamic process generation
- one-sided communication routines
- extended and non-blocking collective operations
- an updated tools interface
- external interfaces, and improved language bindings (especially the new mpi_f08 module that finally allows implementing MPI programs that fully conform with the Fortran standard).

Parallel environments on LRZ systems

A parallel environment is automatically set up at login via an appropriate environment module. Alternative MPI environments are normally also available; these can be accessed by switching to a different module.

A given parallel environment may not be usable on all systems; in such a case loading the environment module will usually fail with an error message.

All environments are listed in the following table; links to individual subdocuments which contain specifics on the implementation are also provided.

<table>
<thead>
<tr>
<th>Fully supported MPI environments</th>
<th>Hardware Interface</th>
<th>supported Compiler(s)</th>
<th>MPI flavour</th>
<th>Environment Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td></td>
<td>Intel and GNU compilers</td>
<td>Intel MPI</td>
<td>mpi.intel</td>
</tr>
</tbody>
</table>

Usable on all Intel and AMD based systems; certain tuning settings will only work on Intel based systems.

default environment on SuperMUC-NG, Linux-Cluster systems, also used on the visualization systems
### Experimental MPI environments

<table>
<thead>
<tr>
<th>Hardware Interface</th>
<th>supported Compiler(s)</th>
<th>MPI flavour</th>
<th>Environment Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any, but may only partially work or have reduced performance</td>
<td>Intel compilers, GNU compilers, PGI compilers, NAG compiler</td>
<td>Open MPI</td>
<td>mpi.ompi</td>
</tr>
<tr>
<td>Any, but may have reduced performance for distributed systems (currently not installed)</td>
<td>Intel compilers, GNU compilers (Others are possible)</td>
<td>MPICH2</td>
<td></td>
</tr>
</tbody>
</table>

If multiple compilers are supported, this is typically encoded into the module name. For example, the Intel MPI 2017 version that supports the compiler wrappers for GCC might be called `mpi.intel/2017_gcc`.

Finally, it should be remarked that different parallel environments are normally **not binary compatible**, so switching over to an alternative MPI usually requires:

- complete recompilation
- relinking, and
- in most cases also using the same environment for execution.

In particular, builds directed to a particular processor may not run on another processor. For example, a binary built for a Sandy Bridge processor that uses AVX instructions will not execute on an earlier Intel processor, or an AMD processor.

### Compiling and linking

For compilation and linkage, compiler wrappers are made available which should be used since they automatically attend to adding the correct include paths and required libraries. The following table illustrates how compilation and linkage might be performed for all supported languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler invocation</th>
<th>Linkage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>mpiif90 -c -o foo.o foo.f90</td>
<td>mpiif90 -o myprog.exe myprog.o foo.o</td>
</tr>
<tr>
<td>Fortran 2008 and later (use mpi_f08)</td>
<td>mpiifort -c -o foo.o foo.f90</td>
<td>mpiifort -o myprog.exe myprog.o foo.o</td>
</tr>
<tr>
<td>C</td>
<td>mpicc -c -o bar.o bar.c</td>
<td>mpicc -o cprog.exe cprog.o bar.o</td>
</tr>
<tr>
<td>C++</td>
<td>mpiCC -c -o stuff.o stuff.cpp</td>
<td>mpiCC -o CPPprog.exe CPPprog.o stuff.o</td>
</tr>
</tbody>
</table>

Of course, suitable application-specific include paths, macros and library paths need to be added (typically via the `-I`, `-D` and `-L` switches, respectively), as well as compiler-specific optimization and/or debugging switches.

### Special cases

When doing autoconf or CMake builds, some MPI implementation’s wrappers do not cooperate nicely with the internal testing done by the `configure` or `cmake` commands. In this case there is no choice but to use the compilers directly and

- for compilation, add the `$MPI_INC` flag so the MPI headers are found
- for linkage using C, add the `$MPI_LIB` flag so the MPI libraries are found and linked against; for linkage with Fortran, add the `$MPI_F90_LIB` flag, and for C++ the `$MPI_CXX_LIB` flag.

### Executing MPI binaries

On LRZ's HPC systems, three different run modes are possible for execution of MPI binaries:

- Interactive runs, which use a moderate number of MPI tasks and run for a short time (typically up to 1-2 hours)
- Production runs, which may use a large number of MPI tasks and run up to the queue limit of a production batch queue (typically 24 hours or more)

Please do not use the interactive nodes for parallel runs. Misuse of the resources will lead to a **removal of the violating program** by LRZ staff without further notice; repeated violations by a given user may lead to the account being **revoked**. For all other modes, the execution is performed by the batch system; the batch settings selected by the user within constraints fixed by LRZ, determine the resource usage.

### How to start up MPI programs

The following table gives an overview of the commands which must be used to start up MPI programs for the various modes. The table limits itself to the **default MPI implementation** on each platform; for alternative MPI implementations please consult the appropriate subdocument. Some of the startup commands also are linked to appropriate platform-specific documentation, while the startup methods `mpiexec` and `mpirun` are described further below. `srun` should only be used when really needed.
LRZ System | Interactive | Production
--- | --- | ---
SuperMUC-NG, Linux-Cluster using SLURM (see also example batch scripts) | mpiexec / (srun) (inside salloc shell on MPP login nodes) | mpiexec / (srun) (inside SLURM job script)

**Startup mechanisms**

This section refers to "standard" startup mechanisms, meaning commands that are documented inside the MPI standard document; specific start mechanisms assure that programs running under control of a batch system are started up correctly, therefore the commands given in the table above should be used.

**SPMD mode**

The standard way of starting up MPI programs in SPMD mode is to use the `mpiexec` command:

```shell
mpiexec -n 128 ./myprog.exe
```

will execute the single program `myprog.exe` with 128 MPI tasks on as many cores (provided sufficient resources are available!). In some cases it may also be possible to use the legacy `mpirun` command:

```shell
mpirun -np 128 ./myprog.exe
```

Please consult the vendor-specific subdocument for details or vendor-specific extensions on the startup mechanism.

**MPMD execution**

For multiple program multiple data mode, the standard way to start up is to specify multiple clauses to `mpiexec`:

```shell
mpiexec -n 12 ./calculate.exe : -n 4 ./control.exe
```

will start up 16 MPI tasks in its MPI_COMM_WORLD, where 12 tasks are run with the binary `calculate.exe` and 4 tasks are run with the binary `control.exe`. The binaries must of course have a consistent communication structure. However, not all MPI implementations support the MPMD execution syntax in their `mpiexec` commands.

**Hybrid parallel programs**

For execution of hybrid parallel MPI programs (for example in conjunction with OpenMP), the startup mechanism depends on the MPI implementation as well as the compiler used; also, it may be necessary to link with a thread-safe version of the MPI libraries. While a setup like

```shell
export OMP_NUM_THREADS=4
mpiexec -n 12 ./myprog.exe
```

might work, starting 12 tasks using 4 threads each (with a resource requirement of 48 cores), there's a good chance that performance will be bad due to incorrect placement of tasks and/or threads. So please consult the vendor-specific subdocument and/or the vendor-specific documentation for further information on how to optimize hybrid execution.

**Environment variables**

Many, but not all MPI implementations export environment variables which are defined in the shell to all tasks. However, the following problems may arise:

1. Some or all variables are not exported. In this case, the MPI implementation usually has a method to specify which variables should be exported via a special switch to the `mpiexec` command, or a special environment variable.
2. Special variables like LD_LIBRARY_PATH or LD_PRELOAD may cause failures for the execution of the `mpiexec` command itself; the symptoms may be crashing or hanging `mpiexec` instances. In this case the solution normally also will be to make use of the special `mpiexec` switch already mentioned above.

Please consult the implementation specific documents for further information.

**MPI-2 and other special topics**

This subsection is in preparation and will contain links to additional pages describing specific MPI-2 features.

**Troubleshooting MPI**

**General MPI Documentation**

**Standard document**

- [MPI 3.1 (June 2015, 2.8 MB PDF)](https://www.mpi-forum.org/docs/mpi-3.1/mpi31.pdf)
Note: The above documents are also available from the MPI Forum's web server, and at least the most recent version of the standard can be purchased as a printed book.

Off-site MPI information

- The MPI Home page provides general information about MPI.
- Development of the standard is done by the MPI Forum.
- There exists a Wikipedia article about MPI which includes some example programs.

Tutorials

- Parallel programming courses at HLRS (requires registration)
- Introduction to MPI (University of Cambridge computing services)
- A User's Guide to MPI, by Peter Pacheco. A brief tutorial introduction to important features of MPI for C programmers (original is located at ftp://math.usfca.edu/pub/MPI/mpi.guide.ps).
- Examples from the book: "Using MPI"
- Examples from the book: "Using MPI-2"

Please also consult the LRZ HPC training page for the latest course materials.