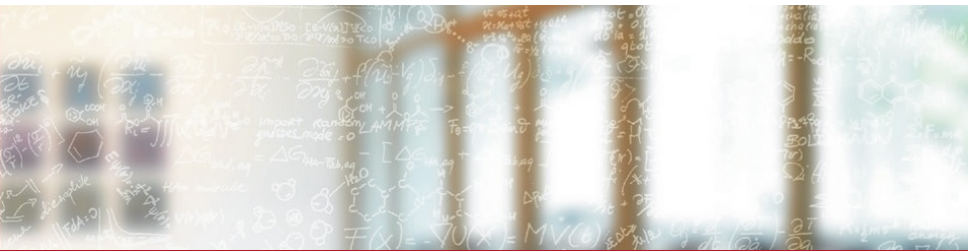




**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



# Message Passing Interface (MPI)

Summer School 2017 – Effective High Performance Computing

Tim Robinson, CSCS

July 19–20, 2017

# Why MPI?

- Distributed memory - use more nodes and cores
- Industry standards endorsed by HPC community
- Several implementations exist:
  - MPICH, OpenMPI, IntelMPI, **CrayMPI**,...

# Course prerequisites

- Basic C and/or Fortran knowledge
- Understand cluster architecture and tools
- Minimum understanding on Network metrics:
  - Bandwidth: ratio of message size / time (GB/s)
  - Latency: minimal time to send one bit ( $\mu s$ )
- To think parallel !

# Course Objectives

- The understanding of MPI's essential concepts
- Be able to use all basic features of MPI
- Be able to write highly parallel HPC code
- Knowing what advanced features exist in MPI
- The understanding of its pitfalls and tricky features

# Behind the course

- Full standard:  
<http://www.mpi-forum.org>
- Tutorials:  
<https://computing.llnl.gov/tutorials/mpi/>
- Books:
  - Parallel Programming with MPI - Oct 96
  - MPI: The Complete Reference - Sept 98
  - Using MPI - 2nd Edition - Nov 99

**A lot of references and tutorials on Internet!**

# General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Topology
- Datatypes

# General Course Structure



- An introduction to MPI
  - MPI
  - Distributed memory
  - Using MPI in a program
  - MPI implementation insight
  - MPI features
  - Practicals
- Point-to-point communications
- Collective communications
- Topology
- Datatypes



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# An introduction to MPI

---



# Message Passing Interface

The Message Passing Interface (MPI) is a library specification for message-passing. It is a standard API (Application Programming Interface) that can be used to create parallel applications. The MPI standardization effort makes use of the most attractive features of a number of existing message passing systems, rather than selecting one of them and adopting it as the standard.

## Key aspects

- A programming model NOT a programming language
  - A set of functions to exchange messages between processes
  - A standard that defines the behaviors of the MPI functions
  - Bindings for C and Fortran
- ⇒ MPI is not a library, per se, rather a spec of what such a library should be

# History

- Early 80's many communication libraries existed: PVM, LAM, P4,...
- 92: agreement to develop one generic library, MPI was born.
- Many companies helped finance the standard: IBM, Cray,...
- 94: First version of the standard was released, MPI-1
- 95: MPICH and LAM-MPI were the first implementations
- 98: Second version of the standard was released, MPI-2
- 02: MPI implementations were MPI-2 compliant
- 08: Third version of the standard was raised, MPI-3

# MPI Standard

- This is a standard, not a user's guide
- It is designed to be unambiguous, not easy to follow.

## MPI-1

- Basic facilities: pt2pt, collective, topology, datatypes,...
- Most people use only a small fraction of it!

## MPI-2

- Parallel I/O, dynamic process management, remote memory operations

## MPI-3

- Fortran 2008 bindings, removes deprecated C++ bindings

# Message Passing Paradigm

- Resources are Local (differently from shared memory model)
- Each process runs in a “isolated” environment. Interactions require exchange of messages
- Messages can be: instructions, data, synchronization
- Message Passing works also in a Shared Memory system
- Time to exchange messages is much larger than accessing local memory

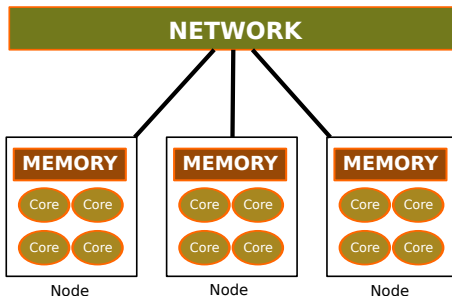
Message Passing is a **COOPERATIVE** approach, based on **THREE** basic operations:

- SEND (a message)
- RECEIVE (a message)
- SYNCHRONIZE

# Distributed memory

## Distributed Memory

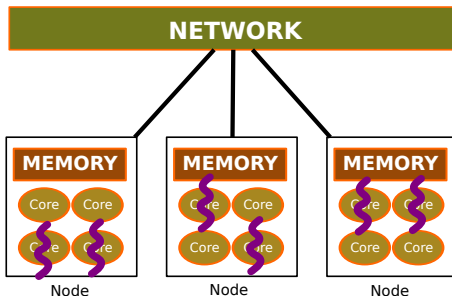
- A program is run as separate, independent processes
- Independent processes do not share data
- Processes interact only by message passing



# Distributed memory

## Distributed Memory

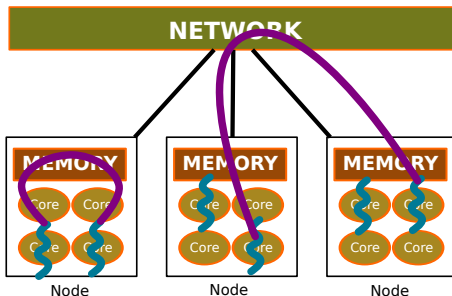
- A program is run as separate, independent processes
- Independent processes do not share data
- Processes interact only by message passing



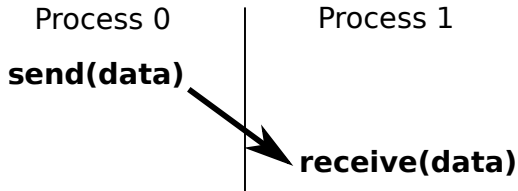
# Distributed memory

## Distributed Memory

- A program is run as separate, independent processes
- Independent processes do not share data
- Processes interact only by message passing



# Send/Recv



- description of data?
- process identification?
- when an operation is completed?



# Using MPI in a program

- Header files
- Initialize and finalize MPI
- Process identification
- Simple communication model
- Example of a simple source code

# Header files

All Subprogram that contains calls to MPI subroutine must include the MPI header file.

Pseudo-code

```
#include <mpi.h>
```

Fortran 77

```
include 'mpif.h'
```

Fortran 90

```
USE MPI
```

The header file contains definitions of MPI constants, types and functions

# MPI initialize and finalize

- Every MPI program starts by calling MPI\_Init:

Pseudo-code

```
MPI_Init(argc, argv)
```

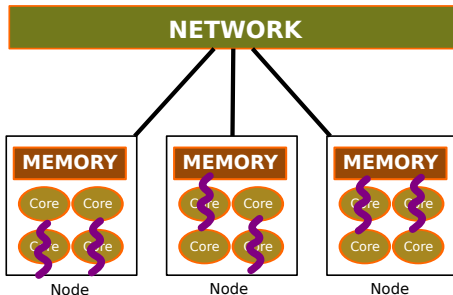
- Every MPI program ends by calling MPI\_Finalize

Pseudo-code

```
MPI_Finalize()
```

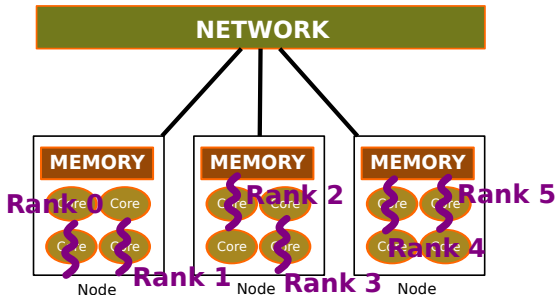
# MPI communicators and ranks

- Every process has an MPI rank and belongs to an MPI communicator



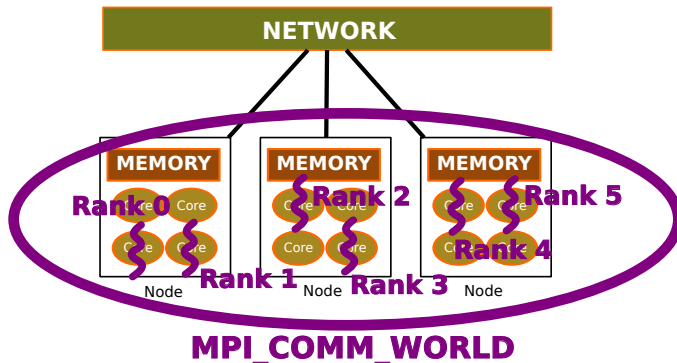
# MPI communicators and ranks

- Every process has an MPI rank and belongs to an MPI communicator
- An MPI rank is an identification number



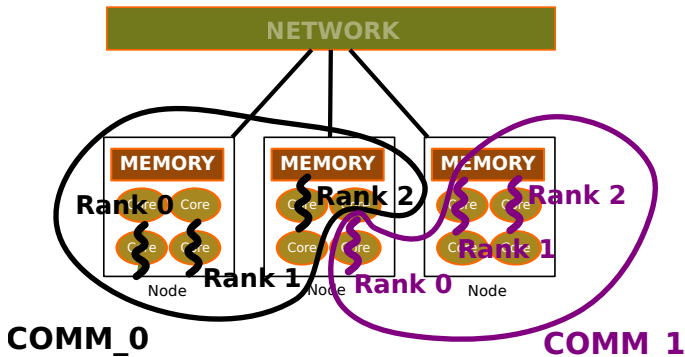
# MPI communicators and ranks

- Every process has an MPI rank and belongs to an MPI communicator
- An MPI rank is an identification number
- An MPI communicator is a set of MPI ranks



# MPI communicators and ranks

- Every process has an MPI rank and belongs to an MPI communicator
- An MPI rank is an identification number
- An MPI communicator is a set of MPI ranks
- Ranks are numbered locally to communicator



# Process identification

- How many processes are associated with a communicator?

Pseudo-code

```
MPI_Comm_size(MPI_Comm comm, size)
```

- How to get the rank of a process?

Pseudo-code

```
MPI_Comm_rank(MPI_Comm comm, rank)
```



# Simple MPI communication model

Process with rank 1 sends to process with rank 2

Pseudo-code

```
Rank 1: MPI_Send(<send data buffer>, 2, MyCommunicator)
Rank 2: MPI_Recv(<recv data buffer>, 1, MyCommunicator)
```

# Simple MPI communication model

Process with rank 1 sends to process with rank 2

Pseudo-code

```
Rank 1: MPI_Send(<send data buffer>, 2, MyCommunicator)
Rank 2: MPI_Recv(<recv data buffer>, 1, MyCommunicator)
```

- same communicator: MyCommunicator
- send and recv buffer should be compatible:
  - receive buffer should be large enough
  - data type should match

# Simple MPI communication model

Process with rank 1 sends to process with rank 2

Pseudo-code

```
Rank 1: MPI_Send(<send data buffer>, 2, MyCommunicator)
Rank 2: MPI_Recv(<recv data buffer>, 1, MyCommunicator)
```

- same communicator: MyCommunicator
  - send and recv buffer should be compatible:
    - receive buffer should be large enough
    - data type should match
- 
- Rank 2 is prepared to receive data from Rank 1
    - `MPI_Recv` is called in “the right order” (avoid deadlock)
    - Rank 2 knows the maximum bound on the buffer size

⇒ **Parallel!**

# Example of MPI source code

C/C++

```
#include<mpi.h>
#include<assert.h>
int main(int argc, char *argv[]){
    int data[64];
    int nranks, my_rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nranks);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    assert(nranks % 2 == 0); // if not?

    if (my_rank % 2 == 0) {
        MPI_Send(data, 64, MPI_INT, my_rank+1, 0,
                 MPI_COMM_WORLD);
    } else {
        MPI_Recv(data, 64, MPI_INT, my_rank-1, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    MPI_Finalize();
}
```

# MPI implementation insight

⇒ **Implementation dependent!**

## Launcher: mpirun/srun

- Starts all process on all compute nodes (ssh)
- Attributes rank numbers
- Applies specific options like process pinning

## Library functions

- Setup process (rank, size) from launcher
- Setup underlying network library (TCP, RDMA, ...)
- Process event coming from the application
  - send, receive, wait, test, cancel, ...

# MPI features

- Different flavours of point-to-point communications
  - Blocking, non-blocking, synchronous, ...
- Collective operations among ranks
  - Broadcast, scatter, gather, reduce, alltoall, ...
- Topology for managing rank numbering
  - Cartesian topology, graph topology, ...
- User specific data type (like C structure)
- Parallel I/O
  - read and write files in parallel

# Practicals

## Exercises: 01.MPI\_Intro

1. Hello World!
2. Hello World! with rank number

## Reminder

```
srun -n 2 ./my_application my_args
```

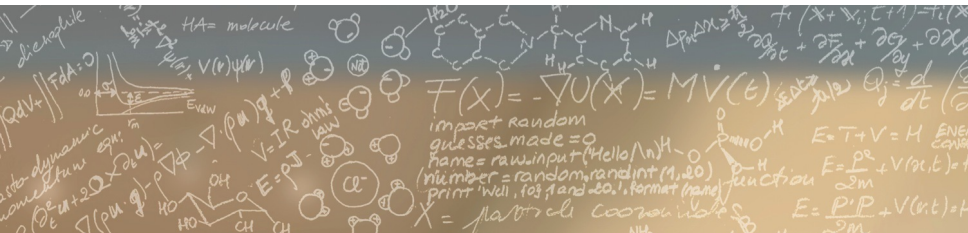
Starts with 2 MPI ranks.



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



**Thank you for your attention.**