

# Message Passing with MPI

Joachim Hein  
LUNARC & Centre of Mathematical Sciences  
Lund University

1

Compiling and running MPI code  
on COSMOS at LUNARC

2

## Overview

- Connecting to COSMOS using Lunarc HPC desktop
- The Lunarc module environment
- Building MPI executables on Lunarc systems
- Executing MPI jobs on Lunarc systems
- Write your first MPI program

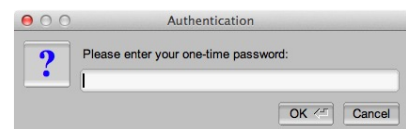
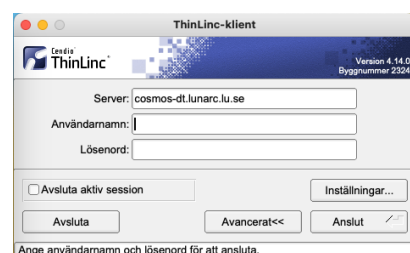
3

## Connecting to Aurora: Lunarc HPC Desktop

- Download and install the client from: <http://www.cendio.com/downloads/clients/>
- Launch the client
- Enter "cosmos-dt.lunarc.lu.se" in the server field.
- Enter your user-id & password
- Click [Connect]
- Enter the one time password from your PocketPass app

Detailed descriptions:

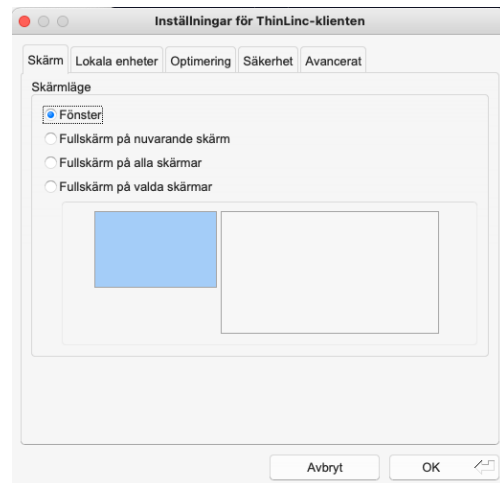
- [https://lunarc-documentation.readthedocs.io/en/latest/getting\\_started/authenticator\\_howto/](https://lunarc-documentation.readthedocs.io/en/latest/getting_started/authenticator_howto/)
- [https://lunarc-documentation.readthedocs.io/en/latest/getting\\_started/using\\_hpc\\_desktop/](https://lunarc-documentation.readthedocs.io/en/latest/getting_started/using_hpc_desktop/)



4

## Some Useful Tricks

- Disable “fullscreen” before connecting
- F8 (fn F8 on MBP) during a DT session could be your friend



5

## Building MPI executables

C, C++ and Fortran

6

## Building MPI executables

- You use standard compilers (e.g. Intel, GCC, ...) to compile the source
- Link against the MPI library to build an MPI executable
- LUNARC/NAISS systems use modules to make these steps convenient

7

## Quick primer to modules

- Show all available modules  
`module spider`
- Load a recent foss toolchain (GCC 12.2.0, OpenMPI 4.1.4, ...)  
`module load foss/2022b`
- We also provide Intel toolchains
- To get rid of all modules  
`module purge`

8

## Hands on: Compiling MPI with OpenMPI

- Compile and link your code using wrapper scripts

Language	Command
Fortran	<code>mpifort</code>
C	<code>mpicc</code>
C++	<code>mpiCC</code>

- The wrappers will call the compiler and tell it about the MPI-lib
- For a foss toolchain they will call `gcc`, `g++` or `gfortran`

10

## Example: Fortran source with OpenMPI

- Can use options of underlying compiler
- Example:
 

```
mpifort -O3 -march=native -o program program.f90
```

  - takes the source: `program.f90`
  - calls `gfortran` compiler
  - creates an executable: `program`
  - uses optimisation level: `O3`
  - optimises for present architecture (instruction set)

11

## Example: C-source with OpenMPI

- Can use options of underlying compiler
- Example:
 

```
mpicc -O3 -march=native -o program program.c
```

  - takes the source: `program.f90`
  - calls gcc compiler
  - creates an executable: `program`
  - uses optimisation level: `O3`
  - optimises for present architecture (instruction set)

12

## Cmake

- CMake has a utility: **FindMPI**
  - locate the MPI library
  - set up compiler and linker flags
- FindMPI set up the following Cmake variables (v 2.8.12)
  - `MPI_<lang>_FOUND`
  - `MPI_<lang>_COMPILER`
  - `MPI_<lang>_INCLUDE_PATH`
  - `MPI_<lang>_LINK_FLAGS`
  - `MPI_<lang>_COMPILE_FLAGS`
  - `MPI_<lang>_LIBRARIES`
- `<lang>` is one of C, CXX and Fortran

13

## CMake example for MPI in Fortran

```
project(mpihello)
enable_language(Fortran)
include(FindMPI)
add_executable(mpihello.x mpihello.f90)

target_include_directories(mpihello.x
                          PUBLIC "${MPI_Fortran_INCLUDE_PATH}")
target_link_libraries(mpihello.x "${MPI_Fortran_LIBRARIES}")
set(CMAKE_Fortran_FLAGS
    "${CMAKE_Fortran_FLAGS} ${MPI_Fortran_COMPILE_FLAGS}")
set(CMAKE_EXE_LINKER_FLAGS
    "${CMAKE_EXE_LINKER_FLAGS} ${MPI_Fortran_LINK_FLAGS}")
```

14

## Running MPI programs

- Make sure the modules (compiler and MPI library) used for building the executable are loaded (shared objects!)
- Message passing jobs need to run inside the batch submission system
- Batch submission systems are vital for running message passing applications
- Expect a similar set-up on any well managed service
- Use the course account for quick turn around of jobs

15

## Standard MPI job

Run executable: **processor\_mpi**

```
#!/bin/bash
#SBATCH -N 2                # number of nodes
#SBATCH --tasks-per-node=48 # number of processes per node
#SBATCH -t 00:05:00         # job-time - here 5 min
#SBATCH -J data_process     # name of job
#SBATCH -o process_mpi_%j.out # output file
#SBATCH -e process_mpi_%j.err # error messages
#SBATCH -A lu2023-7-75      # only for the course
#SBATCH --reservation=mpi-course # only for the course - change for other course days

cat $0
module purge
module load foss/2022b
mpirun --bind-to core ./processor_mpi      #run the program
```

16

## Running MPI4PY scripts

17



## We need a few extra modules

- Load a recent foss toolchain (GCC 12.2.0, OpenMPI 4.1.4, ...)

```
module load foss/2022b
```

- We need Python, NumPy, SciPy, MPI4PY

```
module load Python/3.10.8
```

```
module load SciPy-bundle/2023.02
```

```
module load mpi4py/3.1.4
```

- To get rid of all modules

```
module purge
```

- Required in jobscript

18

## Standard MPI4PY job

Run executable: **processor\_mpi.py**

```
#!/bin/bash
#SBATCH -N 2                # number of nodes
#SBATCH --tasks-per-node=48 # number of processes per node
#SBATCH -t 00:05:00        # job-time - here 5 min
#SBATCH -J data_process     # name of job
#SBATCH -o process_mpi_%j.out # output file
#SBATCH -e process_mpi_%j.err # error messages
#SBATCH -A lu2023-7-75      # only for the course
#SBATCH --reservation=mpi-course # only for the course - change for other course days

cat $0
module purge
module load foss/2022b
module load Python/3.10.8 SciPy-bundle/2023.02 mpi4py/3.1.4
mpirun --bind-to core python3 ./processor_mpi      #run the python script
```

19

# Interacting with SLURM

Submission, queue monitoring and modification

20

## Submission with **sbatch**

- Use sbatch to submit your job script to the job-queue
- Example:

```
[fred@cosmos Timetest]$ sbatch runjob.sh  
Submitted batch job 7197
```

- Submit script “runjob.sh”
- Successful submission returns a job-id number

21

## Monitoring the queue with **squeue**

- Use **squeue** to monitor the job queue

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
7303	lu48	hybrid_n	fred	PD	0:00	32	(Priority)
7302	lu48	hybrid_n	fred	PD	0:00	32	(Priority)
7301	lu48	hybrid_n	fred	PD	0:00	32	(Resources)
7304	lu48	preproce	karl	PD	0:00	6	(Priority)
7300	lu48	hybrid_n	fred	R	0:24	32	cn[001-032]
7305	lu48	preproce	karl	R	0:37	6	cn[081-086]
7306	lu48	hybrid_n	fred	R	0:37	6	cn[081-086]
7307	lu48	testsimu	sven	R	0:07	1	cn081

- Typically lots of output – use options of **squeue** to filter

22

## Options of **squeue**

- Showing jobs for a specific user

```
squeue -u fred
```

will show the jobs of user “fred” only

- Option `--start` gives the estimated job start time
  - This can shift in either direction

23

## Deleting jobs with `scancel`

- You can cancel a queued or running job
- Determine job-id, e.g. with `squeue`
- Use `scancel`

```
scancel 7103
```

- terminates job 7103, if running
- removes from the queue

24

## Exercise

- See the section “Running and building MPI codes” in the practicalities document
- Make sure you can
  - Connect to COSMOS
  - Compile C and/or Fortran code
  - Interact with SLURM
  - Run MPI code (C, Fortran and/or Python)

25