

# MPI

## Setting up MPI Environment

The Anselm cluster provides several implementations of the MPI library:

---

MPI Thread  
Library support

---

The Partial  
highly thread  
optimized up  
and to  
stable MPI\_THREAD\_SERIALIZED  
bullet  
lxmpi 1.2.4.1

The Full  
Intel thread  
MPI port  
4.1 up  
to  
MPI\_THREAD\_MULTIPLE

The <Full  
href="http://www.open-  
mpi.org" >  
Open port  
MPI up  
1.6.5 to  
MPI\_THREAD\_MULTIPLE,  
BLCR  
c/r  
support

---

MPI Thread  
Library support

---

The Full  
Open-Thread  
MPI support  
1.8.1 up  
to  
MPI\_THREAD\_MULTIPLE,  
MPI-  
3.0  
support  
port

The Full  
mpich2 thread  
1.9 support  
up  
to  
MPI\_THREAD\_MULTIPLE,  
BLCR  
c/r  
support  
port

---

MPI libraries are activated via the environment modules.

Look up section modulefiles/mpi in module avail

```
$ module avail
```

```
----- /opt/modules/modulefiles/mpi -----
bullxmpi/bullxmpi-1.2.4.1  mvapich2/1.9-icc
impi/4.0.3.008            openmpi/1.6.5-gcc(default)
impi/4.1.0.024            openmpi/1.6.5-gcc46
impi/4.1.0.030            openmpi/1.6.5-icc
impi/4.1.1.036(default)   openmpi/1.8.1-gcc
openmpi/1.8.1-gcc46
mvapich2/1.9-gcc(default) openmpi/1.8.1-gcc49
mvapich2/1.9-gcc46        openmpi/1.8.1-icc
```

There are default compilers associated with any particular MPI implementation.  
The defaults may be changed, the MPI libraries may be used in conjunction with

any compiler. The defaults are selected via the modules in following way

Module	MPI	Compiler suite
PrgEnv-gnu	bullxmpi-1.2.4.1	bullx GNU 4.4.6
PrgEnv-intel	Intel MPI 4.1.1	Intel 13.1.1
bullxmpi	bullxmpi-1.2.4.1	none, select via module
impi	Intel MPI 4.1.1	none, select via module
openmpi	OpenMPI 1.6.5	GNU compilers 4.8.1, GNU compilers 4.4.6, Intel Compilers
openmpi	OpenMPI 1.8.1	GNU compilers 4.8.1, GNU compilers 4.4.6, GNU compilers 4.9.0, Intel Cor
mvapich2	MPICH2 1.9	GNU compilers 4.8.1, GNU compilers 4.4.6, Intel Compilers

Examples:

```
$ module load openmpi
```

In this example, we activate the latest openmpi with latest GNU compilers

To use openmpi with the intel compiler suite, use

```
$ module load intel
$ module load openmpi/1.6.5-icc
```

In this example, the openmpi 1.6.5 using intel compilers is activated

## Compiling MPI Programs

After setting up your MPI environment, compile your program using one of the mpi wrappers

```
$ mpicc -v
$ mpif77 -v
$ mpif90 -v
```

Example program:

```
// helloworld_mpi.c
#include <stdio.h>

#include<mpi.h>

int main(int argc, char **argv) {

    int len;
    int rank, size;
    char node[MPI_MAX_PROCESSOR_NAME];

    // Initiate MPI
```

```

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);

// Get hostame and print
MPI_Get_processor_name(node,&len);
printf("Hello world! from rank %d of %d on host %sn",rank,size,node);

// Finalize and exit
MPI_Finalize();

return 0;
}

```

Compile the above example with

```
$ mpicc helloworld_mpi.c -o helloworld_mpi.x
```

## Running MPI Programs

The MPI program executable must be compatible with the loaded MPI module. Always compile and execute using the very same MPI module.

It is strongly discouraged to mix mpi implementations. Linking an application with one MPI implementation and running mpirun/mpiexec from other implementation may result in unexpected errors.

The MPI program executable must be available within the same path on all nodes. This is automatically fulfilled on the /home and /scratch filesystem. You need to preload the executable, if running on the local scratch /lscratch filesystem.

### Ways to run MPI programs

Optimal way to run an MPI program depends on its memory requirements, memory access pattern and communication pattern.

Consider these ways to run an MPI program: 1. One MPI process per node, 16 threads per process 2. Two MPI processes per node, 8 threads per process 3. 16 MPI processes per node, 1 thread per process.

One MPI\*\* process per node, using 16 threads, is most useful for memory demanding applications, that make good use of processor cache memory and are not memory bound. This is also a preferred way for communication intensive applications as one process per node enjoys full bandwidth access to the network interface.

Two MPI\*\* processes per node, using 8 threads each, bound to processor socket is most useful for memory bandwidth bound applications such as BLAS1 or FFT, with scalable memory demand. However, note that the two processes will share access to the network interface. The 8 threads and socket binding should ensure maximum memory access bandwidth and minimize communication, migration and numa effect overheads.

Important! Bind every OpenMP thread to a core!

In the previous two cases with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You want to avoid this by setting the KMP\_AFFINITY or GOMP\_CPU\_AFFINITY environment variables.

16 MPI\*\* processes per node, using 1 thread each bound to processor core is most suitable for highly scalable applications with low communication demand.

## Running OpenMPI

The **bullxmpi-1.2.4.1** and **OpenMPI 1.6.5** are both based on OpenMPI. Read more on how to run OpenMPI based MPI.

## Running MPICH2

The **Intel MPI** and **mpich2 1.9** are MPICH2 based implementations. Read more on how to run MPICH2 based MPI.

The Intel MPI may run on the Intel Xeon Phi accelerators as well. Read more on how to run Intel MPI on accelerators.