

# Running MPICH2

## MPICH2 program execution

The MPICH2 programs use mpd daemon or ssh connection to spawn processes, no PBS support is needed. However the PBS allocation is required to access compute nodes. On Anselm, the **Intel MPI** and **mpich2 1.9** are MPICH2 based MPI implementations.

### Basic usage

Use the mpirun to execute the MPICH2 code.

Example:

```
$ qsub -q qexp -l select=4:ncpus=16 -I
qsub: waiting for job 15210.srv11 to start
qsub: job 15210.srv11 ready

$ module load impi

$ mpirun -ppn 1 -hostfile $PBS_NODEFILE ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host cn17
Hello world! from rank 1 of 4 on host cn108
Hello world! from rank 2 of 4 on host cn109
Hello world! from rank 3 of 4 on host cn110
```

In this example, we allocate 4 nodes via the express queue interactively. We set up the intel MPI environment and interactively run the helloworld\_mpi.x program. We request MPI to spawn 1 process per node. Note that the executable helloworld\_mpi.x 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

```
$ pwd
/lscratch/15210.srv11
$ mpirun -ppn 1 -hostfile $PBS_NODEFILE cp /home/username/helloworld_mpi.x .
$ mpirun -ppn 1 -hostfile $PBS_NODEFILE ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host cn17
Hello world! from rank 1 of 4 on host cn108
Hello world! from rank 2 of 4 on host cn109
Hello world! from rank 3 of 4 on host cn110
```

In this example, we assume the executable helloworld\_mpi.x is present on shared home directory. We run the cp command via mpirun, copying the executable

from shared home to local scratch . Second mpirun will execute the binary in the /lscratch/15210.srv11 directory on nodes cn17, cn108, cn109 and cn110, one process per node.

MPI process mapping may be controlled by PBS parameters.

The mpirprocs and ompthreads parameters allow for selection of number of running MPI processes per node as well as number of OpenMP threads per MPI process.

### **One MPI process per node**

Follow this example to run one MPI process per node, 16 threads per process. Note that no options to mpirun are needed

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=1:ompthreads=16 -I
$ module load mvapich2
$ mpirun ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 1 MPI processes per node and 16 threads per socket, on 4 nodes.

### **Two MPI processes per node**

Follow this example to run two MPI processes per node, 8 threads per process. Note the options to mpirun for mvapich2. No options are needed for impi.

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=2:ompthreads=8 -I
$ module load mvapich2
$ mpirun -bind-to numa ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 2 MPI processes per node and 8 threads per socket, each process and its threads bound to a separate processor socket of the node, on 4 nodes

### **16 MPI processes per node**

Follow this example to run 16 MPI processes per node, 1 thread per process. Note the options to mpirun for mvapich2. No options are needed for impi.

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=16:ompthreads=1 -I
$ module load mvapich2
```

```
$ mpirun -bind-to core ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 16 MPI processes per node, single threaded. Each process is bound to separate processor core, on 4 nodes.

### OpenMP thread affinity

Important! Bind every OpenMP thread to a core!

In the previous two examples with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You might want to avoid this by setting these environment variable for GCC OpenMP:

```
$ export GOMP_CPU_AFFINITY="0-15"
```

or this one for Intel OpenMP:

```
$ export KMP_AFFINITY=granularity=fine,compact,1,0
```

As of OpenMP 4.0 (supported by GCC 4.9 and later and Intel 14.0 and later) the following variables may be used for Intel or GCC:

```
$ export OMP_PROC_BIND=true
```

```
$ export OMP_PLACES=cores
```

## MPICH2 Process Mapping and Binding

The mpirun allows for precise selection of how the MPI processes will be mapped to the computational nodes and how these processes will bind to particular processor sockets and cores.

### Machinefile

Process mapping may be controlled by specifying a machinefile input to the mpirun program. Although all implementations of MPI provide means for process mapping and binding, following examples are valid for the impi and mvapich2 only.

Example machinefile

```
cn110.bullx
cn109.bullx
cn108.bullx
cn17.bullx
```

cn108.bullx

Use the machinefile to control process placement

```
$ mpirun -machinefile machinefile helloworld_mpi.x
Hello world! from rank 0 of 5 on host cn110
Hello world! from rank 1 of 5 on host cn109
Hello world! from rank 2 of 5 on host cn108
Hello world! from rank 3 of 5 on host cn17
Hello world! from rank 4 of 5 on host cn108
```

In this example, we see that ranks have been mapped on nodes according to the order in which nodes show in the machinefile

## Process Binding

The Intel MPI automatically binds each process and its threads to the corresponding portion of cores on the processor socket of the node, no options needed. The binding is primarily controlled by environment variables. Read more about mpi process binding on Intel website. The MPICH2 uses the `-bind-to` option. Use `-bind-to numa` or `-bind-to core` to bind the process on single core or entire socket.

## Bindings verification

In all cases, binding and threading may be verified by executing

```
$ mpirun -bindto numa numactl --show
$ mpirun -bindto numa echo $OMP_NUM_THREADS
```

## Intel MPI on Xeon Phi

The MPI section of Intel Xeon Phi chapter provides details on how to run Intel MPI code on Xeon Phi architecture.