

Running OpenMPI

OpenMPI program execution

The OpenMPI programs may be executed only via the PBS Workload manager, by entering an appropriate queue. On the cluster, the **OpenMPI 1.8.6** is OpenMPI based MPI implementation.

Basic usage

Use the mpiexec to run the OpenMPI code.

Example:

```
$ qsub -q qexp -l select=4:ncpus=24 -I
qsub: waiting for job 15210.isrv5 to start
qsub: job 15210.isrv5 ready

$ pwd
/home/username

$ module load OpenMPI
$ mpiexec -pernode ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host r1i0n17
Hello world! from rank 1 of 4 on host r1i0n5
Hello world! from rank 2 of 4 on host r1i0n6
Hello world! from rank 3 of 4 on host r1i0n7
```

Please be aware, that in this example, the directive **-pernode** is used to run only **one task per node**, which is normally an unwanted behaviour (unless you want to run hybrid code with just one MPI and 24 OpenMP tasks per node). In normal MPI programs **omit the -pernode directive** to run up to 24 MPI tasks per each node.

In this example, we allocate 4 nodes via the express queue interactively. We set up the openmpi environment and interactively run the helloworld_mpi.x program. 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 ramdisk /tmp filesystem

```
$ pwd
/tmp/pbs.15210.isrv5

$ mpiexec -pernode --preload-binary ./helloworld_mpi.x
```

```
Hello world! from rank 0 of 4 on host r1i0n17
Hello world! from rank 1 of 4 on host r1i0n5
Hello world! from rank 2 of 4 on host r1i0n6
Hello world! from rank 3 of 4 on host r1i0n7
```

In this example, we assume the executable `helloworld_mpi.x` is present on compute node `r1i0n17` on `ramdisk`. We call the `mpiexec` with the **`--preload-binary`** argument (valid for `openmpi`). The `mpiexec` will copy the executable from `r1i0n17` to the `/tmp/pbs.15210.isrv5` directory on `r1i0n5`, `r1i0n6` and `r1i0n7` and execute the program.

MPI process mapping may be controlled by PBS parameters.

The `mpiprocs` 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, 24 threads per process.

```
$ qsub -q qexp -l select=4:ncpus=24:mpiprocs=1:ompthreads=24 -I
$ module load OpenMPI
$ mpiexec --bind-to-none ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 1 MPI processes per node and 24 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 `mpiexec`.

```
$ qsub -q qexp -l select=4:ncpus=24:mpiprocs=2:ompthreads=12 -I
$ module load OpenMPI
$ mpiexec -bysocket -bind-to-socket ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 2 MPI processes per node and 12 threads per socket, each process and its threads bound to a separate processor socket of the node, on 4 nodes

24 MPI processes per node

Follow this example to run 24 MPI processes per node, 1 thread per process. Note the options to mpiexec.

```
$ qsub -q qexp -l select=4:ncpus=24:mpiprocs=24:ompthreads=1 -I
```

```
$ module load OpenMPI
```

```
$ mpiexec -bycore -bind-to-core ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 24 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="0-23"
```

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
```

OpenMPI Process Mapping and Binding

The mpiexec 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.

MPI process mapping may be specified by a hostfile or rankfile input to the mpiexec program. Although all implementations of MPI provide means for process mapping and binding, following examples are valid for the openmpi only.

Hostfile

Example hostfile

```

r1i0n17.smc.salomon.it4i.cz
r1i0n5.smc.salomon.it4i.cz
r1i0n6.smc.salomon.it4i.cz
r1i0n7.smc.salomon.it4i.cz

```

Use the hostfile to control process placement

```

$ mpiexec -hostfile hostfile ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host r1i0n17
Hello world! from rank 1 of 4 on host r1i0n5
Hello world! from rank 2 of 4 on host r1i0n6
Hello world! from rank 3 of 4 on host r1i0n7

```

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

Rankfile

Exact control of MPI process placement and resource binding is provided by specifying a rankfile

Appropriate binding may boost performance of your application.

Example rankfile

```

rank 0=r1i0n7.smc.salomon.it4i.cz slot=1:0,1
rank 1=r1i0n6.smc.salomon.it4i.cz slot=0:*
rank 2=r1i0n5.smc.salomon.it4i.cz slot=1:1-2
rank 3=r1i0n17.smc.salomon slot=0:1,1:0-2
rank 4=r1i0n6.smc.salomon.it4i.cz slot=0:*,1:*

```

This rankfile assumes 5 ranks will be running on 4 nodes and provides exact mapping and binding of the processes to the processor sockets and cores

Explanation: rank 0 will be bounded to r1i0n7, socket1 core0 and core1 rank 1 will be bounded to r1i0n6, socket0, all cores rank 2 will be bounded to r1i0n5, socket1, core1 and core2 rank 3 will be bounded to r1i0n17, socket0 core1, socket1 core0, core1, core2 rank 4 will be bounded to r1i0n6, all cores on both sockets

```

$ mpiexec -n 5 -rf rankfile --report-bindings ./helloworld_mpi.x
[r1i0n17:11180] MCW rank 3 bound to socket 0[core 1] socket 1[core 0-2]: [. B . . . . .]
[r1i0n7:09928] MCW rank 0 bound to socket 1[core 0-1]: [. . . . .] [B B . . . . .]
[r1i0n6:10395] MCW rank 1 bound to socket 0[core 0-7]: [B B B B B B B B B] [. . . . .]
[r1i0n5:10406] MCW rank 2 bound to socket 1[core 1-2]: [. . . . .] [. B B . . . . .]
[r1i0n6:10406] MCW rank 4 bound to socket 0[core 0-7] socket 1[core 0-7]: [B B B B B B B B B]
Hello world! from rank 3 of 5 on host r1i0n17
Hello world! from rank 1 of 5 on host r1i0n6
Hello world! from rank 0 of 5 on host r1i0n7

```

```
Hello world! from rank 4 of 5 on host r1i0n6
Hello world! from rank 2 of 5 on host r1i0n5
```

In this example we run 5 MPI processes (5 ranks) on four nodes. The rankfile defines how the processes will be mapped on the nodes, sockets and cores. The **-report-bindings** option was used to print out the actual process location and bindings. Note that ranks 1 and 4 run on the same node and their core binding overlaps.

It is users responsibility to provide correct number of ranks, sockets and cores.

Bindings verification

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

```
$ mpiexec -bysocket -bind-to-socket --report-bindings echo
$ mpiexec -bysocket -bind-to-socket numactl --show
$ mpiexec -bysocket -bind-to-socket echo $OMP_NUM_THREADS
```

Changes in OpenMPI 1.8

Some options have changed in OpenMPI version 1.8.

version 1.6.5	version 1.8.1
-bind-to-none	-bind-to none
-bind-to-core	-bind-to core
-bind-to-socket	-bind-to socket
-bysocket	-map-by socket
-bycore	-map-by core
-pernode	-map-by ppr:1:node