Guide for Intel Xeon Phi (MIC) Usage

In this page:

Architecture

Intel Xeon Phi is the first Intel Many Integrated Core (Intel MIC) architecture product. Each card consists of 60 physical cores (@1.1 Ghz) and each core is able to handle up to 4 threads using hyperthreading. Each core has one Vector Processing Unit able to deliver, for each clock cycle:

- 8 Fused Multiply and Add (FMA) floating point operations in double precision
- 16 Fused Multiply and Add (FMA) floating point operations in single precision.

So the Phi has a **peak** performance of

- 1056 GFlops in double precision
- 2112 Gflops in single precision

Each Phi coprocessor has a RAM memory of 8 GB, and a **peak** bandwidth of 352 GB/s.

Compilation

The MPSS environment (Intel® *Manycore Platform Software Stack*) is available also on the front-end. Therefore, you do not need to be logged **inside** a compute node hosting the MIC cards to compile a code to run on MIC. Anyway you still have to set environment for mic:

```
module load intel (i.e. compiler suite)
module load mkl (if necessary – i.e. math libraries)
source $INTEL_HOME/bin/compilervars.sh intel64 (to set up the environment variables)
```

Now you can compile your code. Pay attention that, depending on the way you intend to run your code (offload or native), you have to follow different procedures:

1) For codes meant to be run with the **MIC offload attributes**, you have to add the proper pragmas in your source code and compile it as usual. For example, use the Intel C++ compiler on the "hello_offload.cpp" code:

```
icpc -openmp hello_offload.cpp -o exe-offload.x
```

2) For **MIC-native codes**, you have to actually cross-compile by adding the **–mmic** flag. For example, use the Intel C++ compiler on the "hello_native.cpp" code:

```
icpc hello_native.cpp -openmp -mmic -o exe-native.x
```

Please note that if you need to run native codes linking the mkl libraries, you need to source the additional, proper configuration script (mklvars.sh) with the "mic" switch":

```
source $INTEL_HOME/composer_xe_2015/mkl/bin/mklvars.sh mic (if necessary - i.e. math libraries)
icc mycode_withmkl_native.c -openmp -mmic -l<mkl_libs> -o mycode_withmkl_native.x
```

Execution

Offload programs are executed directly on the MIC node, from an interactive batch session or even by a batch script (requesting MIC cards with the nmics parameter). Note that the sourcing of the compilervars.sh script is important for making the node see the MICs during execution.

Offload programs execution through an **interactive batch session**

```
qsub -A <account_name> -I -l select=1:ncpus=1:nmics=1
       qsub: waiting for job 31085.node129 to start
       qsub: job 31085.node129 ready

cd $PBS_O_WORKDIR

module load intel
source $INTEL_HOME/bin/compilervars.sh intel64
./exe-offload.x
```

Offload programs execution through a **batch script**

```
#!/bin/bash
#PBS -o job.out
#PBS -j eo
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:nmics=1
#PBS -A <my_account>
#

cd $PBS_O_WORKDIR

module load intel
source $INTEL_HOME/bin/compilervars.sh intel64
./exe-offload.x
```

MIC-native programs need to be executed inside the MIC card itself. In order to log into a MIC card you have to:

- login to a MIC node with a PBS interactive session requesting at least 1 mic (nmics=1);
- use the "get_dev_list" script (available by loading the "superc" module) in order to get the name of the specific MIC card assigned to you. The script will produce in output an hostfile named <job_id>_dev_hostfile containing the lists of the assigned cards;
- connect through ssh into the MIC card (in the example node254-mic0)

Mic-native programs execution through an **interactive batch session**

```
qsub -A <account_name> -I -l select=1:ncpus=1:nmics=1
qsub: waiting for job 10876.io01 to start
 qsub: job 10876.io01 ready
...
cd $PBS_O_WORKDIR
module load superc
get_dev_list
cat ${PBS_JOBID}_dev_hostfile
 node254-mic0
...
ssh node254-mic0  (*)
$

(*) In order to SSH access the mic card you have to create the public key of the Galileo username
in your $HOME from login node
```

At this point you will be prompted in the home space of the MIC card you've logged into. Here, the usual environment variables are **not** set, therefore the module command won't work and your scratch space (which is mounted on the MIC card) has to be indicated with the full path instead of $CINECA_SCRATCH.

For executing your native-MIC program, **you need** to set the LD_LIBRARY_PATH environment variable manually, by adding the path of the intel libraries specific for MIC execution:

```
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/lib/mic:${LD_LIBRARY_PATH}
```

You may need to add also path for mkl and/or tbb (*Intel® Thread Building Blocks*) MIC libraries:

```
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/mkl/lib/mic:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/tbb/lib/mic:${LD_LIBRARY_PATH}
```

When everything is ready, you can launch your code as usual.

```
cd /gpfs/scratch/userexternal/<myuser>
./exe.native.x
```

MPI Compilation

In order to compile an application suited for MICs, you need the MPSS environment (Intel® *Manycore Platform Software Stack*) to be set

```
module load intel (i.e. compiler suite)
module load intelmpi (i.e. mpi library)
source $INTEL_HOME/bin/compilervars.sh intel64 (to set up the environment variables)
```

Now you can compile your code. For **MIC-native codes**, you have to actually cross-compile by adding the –mmic flag. For program written in C use "mpicc", for program written in Fortran you have to use the "**mpifc**" command

```
mpicc -O3 -mmic mpi_code.c
...
mpifc -O3 -mmic mpi_code.f
```

MPI Execution

MIC-native codes can be launched from MIC node, once you get your MIC card through qsub (in the example node254-mic0)

MPI mic-native programs execution through an **interactive batch session**

```
qsub -A <account_name> -I -l select=1:ncpus=1:nmics=1
qsub: waiting for job 31085.nodeio1 to start
 qsub: job 31085.nodeio1 ready
...
cd $PBS_O_WORKDIR
module load superc
get_dev_list
cat ${PBS_JOBID}_dev_hostfile
node254-mic0
...
```

When you know your MIC card (in the example node254-mic0) you can lanch your MPI program (in the example using 30 tasks). Before MPI on MIC must be enabled setting the **I_MPI_MIC** environment variable

```
module load intel
module load intelmpi

export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/lib/mic:${LD_LIBRARY_PATH}
export I_MPI_MIC=enable

mpirun.mic  -host node254-mic0 -np 30  ./a.out
```

(*) you have to create the public key of the Galileo username in your $HOME from login node

Attention: use only the "mpirun.mic" command, "mpiexec" doesn't work correctly

If you need pass some variables you have to use the -"genv" flag

```
export I_MPI_MIC=enable
mpirun.mic -genv I_MPI_DEBUG 0 -genv I_MPI_PIN 1 -host node254-mic0 -np 30  ./a.out
```

if you want to use two MIC cards (so you have to ask for two MICs by setting "nmics=2" in your qsub request) you can set the number of tasks per card via the -perhost command

```
export I_MPI_MIC=enable
mpirun.mic  -host node254-mic0,node254-mic1 -perhost 15 -np 30  ./a.out
```

Alternatively, you can use the hostfile produced by the "get_dev_list" command (<job_id>_dev_hostfile)

```
export I_MPI_MIC=enable
mpirun.mic -machinefile ${PBS_JOBID}_dev_hostfile -np 30  ./a.out
....
cat ${PBS_JOBID}_dev_hostfile
 node254-mic0
 node254-mic1
```

MPI mic-native programs execution through **a batch script**

```
#!/bin/bash
#PBS -o job.out
#PBS -j eo
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:nmics=1
#PBS -A <my_account>

cd $PBS_O_WORKDIR

module load superc
get_dev_list

module load intel
module load intelmpi
#if necessary – i.e. math libraries
module load mkl

export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/lib/mic:${LD_LIBRARY_PATH}
#if necessary – i.e. math libraries
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/mkl/lib/mic:${LD_LIBRARY_PATH}

export I_MPI_MIC=enable
mpirun.mic  -machinefile ${PBS_JOBID}_dev_hostfile ./exe.native.x
```

(*) you have to create the public key of the Galileo username in your $HOME from login node

Hybrid (OpenMP-MPI) Execution

You can compile your **MIC-native codes** as shown before using **mpicc** and **-openmp** and **-mmic** flags.

```
mpicc -O3 -openmp -mmic hyb_code.c
...
mpifc -O3 -openmp -mmic hyb_code.f
```

And then launch your code, using batch script as shown before, with mpi task distribution between MIC and exporting all environment variables nedeed.

```
...
export I_MPI_MIC=enable
mpirun.mic  -machinefile ${PBS_JOBID}_dev_hostfile -perhost 1 -np 2 \
-genv OMP_NUM_THREADS 120 ./a.out
```

In this example each MIC has one mpi task, each of them present 120 different threads.

Some examples

Here you'll find some example, together with source code for native mode (OpenMP, MPI, Hybrid parallelization) on MIC.

Accounting

At present the use of the MICs and other accelerators is not accounted, only the time spent on the cpus is considered.

More details about "Accounting" can be found in the UserGuide (http://www.hpc.cineca.it/content/accounting-0).

H

Tofixpiexec doesn't worortran does't work

© C