Intel ® MPI on Intel ® Xeon Phi™ - Lab Instructions

Lab 0 - Prerequisites

Assumptions:

- Intel® Manycore Platform Software Stack (Intel® MPSS) Gold is installed
- User accounts exist on the coprocessor
- ssh/scp is available or WinCP tools
- NFS is available
- The compiler is installed in directory
  `export ICCROOT=/opt/software/intel/composerxe`
- Intel MPI is installed in directory
  `export I_MPI_ROOT=/opt/software/intel/impi/4.1.1.036`
- Intel Trace Analyzer and Collector (ITAC) is installed in directory
  export `VT_ROOT=/opt/software/intel/itac/8.1.2.033`

The path to the Intel MPI reference manual is `$I_MPI_ROOT/doc/Reference_Manual.pdf`. Use it for details about the environment variables. It does not yet include specific Intel® Xeon Phi™ coprocessor details. For coprocessor details, see:
`$I_MPI_ROOT/doc/Reference_Manual/Intel_Xeon_Phi_Coprocessor_Support.htm`
The trace analyzer reference guide is availabɯle at
`$VT_ROOT/doc/ITA_Reference_Guide.pdf`. For the collector libraries, see

`$VT_ROOT/doc/doc/Release_Notes_Addendum_for_MIC_Architecture.txt`, currently in Beta without Intel® Xeon Phi™ coprocessor details.

First check your user environment for the compiler, Intel MPI, and ITAC:

```
# which icc mpiicc traceanalyzer
```

If one of the tools is not available, setup the appropriate environment for the compiler, Intel MPI, and ITAC by executing the corresponding source files, and check again:

```
# source $ICCROOT/bin/compilervars.sh intel64

# source $I_MPI_ROOT/intel64/bin/mpivars.sh

# source $VT_ROOT/intel64/bin/itacvars.sh impi4

# which icc mpiicc traceanalyzer
```

Add the paths in $MIC_LD_LIBRARY_PATH and $VT_ROOT/mic/slib to the LD_LIBRARY_PATH. Otherwise the OpenMP lib or the ITAC collector lib will not be found in the hybrid or ITAC labs, respectively.

```
# export MIC_LD_LIBRARY_PATH=/opt/software/intel/composerxe/lib/mic

# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MIC_LD_LIBRARY_PATH
```

The Intel MPI run command starts processes on the Intel® Xeon PhiTM coprocessor only when the environment variable I_MPI_MIC is set. It is required for all labs:

```
# export I_MPI_MIC=1

# export I_MPI_FABRICS=shm:tcp
```

The generic host name for the Intel® Xeon Phi™ coprocessor is `` `hostname`-mic0' ``, or a short version of it, and can be found in `/etc/hosts`. Throughout this document the short version `mic0` is used;

Lab 1 - Bascis

PURE MPI

Each Intel MPI distribution includes a test directory which contains a simple MPI program coded in C, C++, or Fortran.

Enter directory 1_Basics where you will find a copy of the source file test.c from the Intel MPI distribution. You will use this code and two variants for your first runs with Intel MPI on the Intel® Xeon PhiTM coprocessor.
Compile and link the source file with the Intel compiler for the host with the usual Intel MPI script:

```
# mpiicc -o test test.c
```

Compile and link the source file for Intel® Xeon Phi™ coprocessor using the "-mmic" compiler flag. Because of the flag the Intel MPI script will provide the Intel MPI libraries for Intel® Xeon Phi™ coprocessor to the linker (add the verbose flag "-v" to see it):

```
# mpiicc -mmic -o test.MIC test.c
```

The ".MIC" suffix is added by the user to distinguish the coprocessor binary from the host one. It could be any suffix!

You may want to set the communication fabrics for intra and inter node communication (<intra:inter>) explicitly. In principle this is not required, because Intel® MPI will select the best available network automatically. However, error messages "ERROR - load_iblibrary" may be printed out. These are just messages generated by Intel MPI when analyzing the available interconnects between the host and the Intel® Xeon PhiTM coprocessor without finding Infiniband (it will fall back to tcp). You can avoid the error messages by setting explicitly:

```
# export I_MPI_FABRICS=shm:tcp
```

As a starter run the Xeon binary with 2 MPI processes alone:

```
# mpirun -n 2 ./test
```

Now run your first Intel MPI program on Intel® Xeon PhiTM coprocessor in coprocessor-only mode:

```
# mpirun -host mic0 -n 2 ./test.MIC
```

An alternative would be to login onto the coprocessor and run the test from there in a straightforward manner. Try it if you like.

Pulling it together you can run the test code on Intel® Xeon® processor and Intel® Xeon Phi™ coprocessor as one MPI program in symmetric mode. Each argument set (command line sections separated by ":") is defined independently; therefore, 4 MPI processes are chosen on the Intel® Xeon Phi™ coprocessor in this example:

```
# mpirun -host 'hostname' -n 2 ./test : -host 'hostname'-mic0 -n 4 ./test.MIC
```

Please notice: In the symmetric mode you must provide the "-host" flag for the MPI processes running on the Xeon host!

As an alternative to the previous command you can declare a machinefile with hosts and number of processes per hosechot defined. Use this machinefile together with the environment flag I_MPI_MIC_POSTFIX. The value of I_MPI_MIC_POSTFIX is automatically added to the executable on the Intel® Xeon Phi™ coprocessor:

```
# echo `hostname`:2 > machinefile
# echo mic0:4 >> machinefile
# export I_MPI_MIC_POSTFIX=.MIC
# mpirun –machinefile machinefile –n 6 ./test
```

NOTE: On this cluster `I_MPI_MIC_POSTFIX may not work!`

As preparation for the next lab on hybrid programs, the mapping/pinning of Intel MPI processes will be investigated in the following. Set the environment variable I_MPI_DEBUG equal or larger than 4 to see the mapping information, either by exporting it:

```
# export I_MPI_DEBUG=4
```

mpirunor by adding it as a global environment flag ("`-genv`") onto the command line close to `mpirun` (without "="):

```
# mpirun -genv I_MPI_DEBUG 4 …
```

For pure MPI programs (non-hybrid) the environment variable I_MPI_PIN_PROCESSOR_LIST controls the mapping/pinning. For hybrid codes the variable I_MPI_PIN_DOMAIN takes precedence. It splits the (logical) processors into non-overlapping domains for which this rule applies: "1 MPI process for 1 domain".

Repeat the Intel MPI test from before with I_MPI_DEBUG set. Because of the amount of output the usage of the flag "**-prepend-rank**" is recommended which puts the MPI rank number in front of each output line:

```
# mpirun -prepend-rank -n 2 ./test
# mpirun -prepend-rank -host "hostname"-mic0 -n 2 ./test.MIC
# mpirun -prepend-rank -host `hostname` -n 2 ./test : -host "hostname"-mic0 -n 4
./test.MIC
```

Now set the variable I_MPI_PIN_DOMAIN with the "-env" flag. Possible values are "auto", "omp" (which relies on the OMP_NUM_THREADS variable), or a fixed number of logical cores. By exporting I_MPI_PIN_DOMAIN in the shell or using the global "-genv" flag, the variable is identically exported to the host and the Intel® Xeon Phi™ coprocessor. Typically this is not beneficial and an architecture adapted setting with "-env" is recommended:
te

```
# mpirun -prepend-rank -env I_MPI_PIN_DOMAIN auto -n 2 ./test
# mpirun -prepend-rank -env I_MPI_PIN_DOMAIN auto -host mic0 -n 2 ./test.MIC
# mpirun -prepend-rank -env I_MPI_PIN_DOMAIN 4 -host `hostname` -n 2 ./test :
-env I_MPI_PIN_DOMAIN 12 -host 'hostname'-mic0 -n 4 ./test.MIC
```

HYBRID

Now we want to run our first hybrid MPI/OpenMP program on the Intel® Xeon Phi™ coprocessor. A simple printout from the OpenMP threads was added to the Intel MPI test code:

```
# diff test.c test-openmp.c
```

Compile and link with the "-openmp" compiler flag. Note: result binary will upload to the Intel® Xeon Phi™ coprocessor automatically:

```
# mpiicc -openmp -o test-openmp test-openmp.c
# mpiicc -openmp -mmic -o test-openmp.MIC test-openmp.c
```

Because of the "-openmp" flag, Intel® MPI will link the code with the thread-safe version of the Intel MPI library (libmpi_mt.so) by default.

Run the Intel MPI tests from before:

```
# unset I_MPI_DEBUG # to reduce the output for now
# export MIC_LD_LIBRARY_PATH=/opt/software/intel/composerxe/lib/mic
# mpirun -prepend-rank -n 2 ./test-openmp
# mpirun -prepend-rank -host 'hostname'-mic0 -n 2  -env
LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH ./test-openmp.MIC
# mpirun -prepend-rank -host `hostname` -n 2 ./test-openmp : -host
'hostname'-mic0 -n 4  -env LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH ./test-openmp.MIC
```

Lot of output! The default for the OpenMP library is to assume as many OpenMP threads as there are logical processors. For the next tests, explicit **OMP_NUM_THREADS** values (different on host and Intel® Xeon Phi™ coprocessor) will be set.

In the following test the default OpenMP affinity is checked. Please notice that the range of logical processors is always defined by the splitting the threads based on the **I_MPI_PIN_DOMAIN** variable. This time we also use **I_MPI_P IN_DOMAIN=omp**, see how it depends on the **OMP_NUM_THREADS** setting:

Modifier **verbose** tells the Intel OpenMP* runtime libraries to print out messages concerning the supported affinity, including information about the number of packages, number of cores in each package, number of thread contexts for each core, and OpenMP* thread bindings to physical thread contexts.

```
 #mpirun -prepend-rank -env KMP_AFFINITY verbose -env OMP_NUM_THREADS 8 -env
I_MPI_PIN_DOMAIN auto -n 2 ./test-openmp 2>&1 | sort

#mpirun -prepend-rank -env KMP_AFFINITY verbose -env OMP_NUM_THREADS 4 -env
I_MPI_PIN_DOMAIN omp -env LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH  -host
'hostname'-mic0 -n 2 ./test-openmp.MIC 2>&1 | sort
 # mpirun -prepend-rank -env KMP_AFFINITY verbose -env OMP_NUM_THREADS 4 -env
I_MPI_PIN_DOMAIN 4 -host 'hostname' -n 2 ./test-openmp : -env KMP_AFFINITY
verbose -env OMP_NUM_THREADS 6 -env I_MPI_PIN_DOMAIN 12 -env
LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -host 'hostname'-mic0 -n 4 ./test-openmp.MIC
```

Remember that it is usually beneficial to avoid splitting of cores on Intel® Xeon Phi™ coprocessor between MPI processes. Either the number of MPI processes should be chosen so that **I_MPI_PIN_DOMAIN=auto** creates domains which cover complete cores or the environment variable should be a multiply of 4.

Short description of the KMP_AFFINITY and **MIC_KMP_AFFINITY**

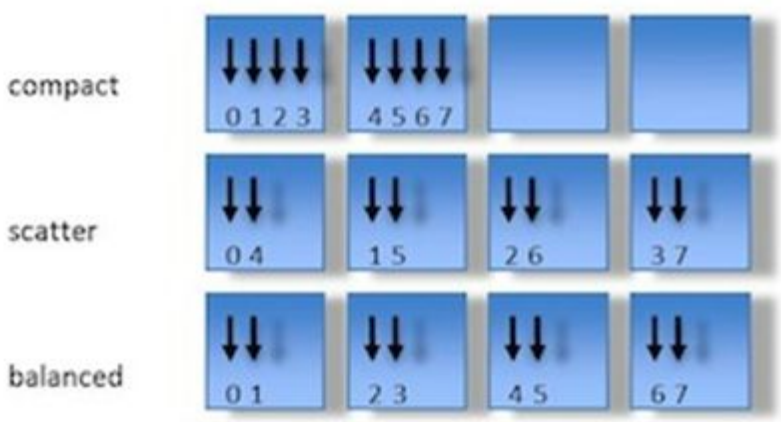
- Set this environment variable to influence thread affinity generally
- OpenMP programs are affected on CPU and MIC (regardless of whether the MIC is used as an SMP or offload)

export KMP_AFFINITY=<type> (for CPU)

export MIC_KMP_AFFINITY=<type> (for MIC)

Type	Effect
compact	Pack threads close to each other.
explicit	Use the proclist modifier to pin threads.
none	Does not pin threads.
scatter	Round-robin threads to cores.
balanced (Phi only)	Use scatter, but keep OMP thread ids consecutive.

- Imagine a system with 4 cores and 4 hardware threads/core
- Placement of 8 threads is illustrated for the 3 types
- Compact type does not fully utilize all cores; not recommended



Use "**scatter**", "**compact**", or "**balanced**" (Intel® Xeon Phi™ coprocessor specific) to modify the default OpenMP affinity

```
# mpirun -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,scatter
-env OMP_NUM_THREADS 8 -env I_MPI_DOMAIN auto -n 2 ./test-openmp 2>&1 | sort

# mpirun -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,compact
-env OMP_NUM_THREADS 8 -env I_MPI_DOMAIN auto -n 2 ./test-openmp 2>&1 | sort
```

```
# mpirun -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,compact
-env OMP_NUM_THREADS 4 -env I_MPI_PIN_DOMAIN omp -env
LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -host 'hostname'-mic0 -n 2
./test-openmp.MIC 2>&1

# mpirun -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,scatter
-env OMP_NUM_THREADS 8 -env I_MPI_PIN_DOMAIN 24 -env
LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -host mic0 -n 2 ./test-openmp.MIC 2>&1 |
sort

# mpirun -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,balanced
-env OMP_NUM_THREADS 8 -env I_MPI_PIN_DOMAIN 24 -env
LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -host mic0 -n 2 ./test-openmp.MIC 2>&1 |
sort
```

```
# mpirun -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,compact
-env OMP_NUM_THREADS 4 -env I_MPI_PIN_DOMAIN 4 -host 'hostname' -n 2
./test-openmp : -env KMP_AFFINITY verbose,granularity=thread,balanced -env
OMP_NUM_THREADS 6 -env I_MPI_PIN_DOMAINS 12 -env
LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -host 'hostname'-mic0 -n 4
./test-openmp.MIC 2>&1 | sort
```

Notice, that as well as other options the OpenMP affinity can be set differently per Intel MPI argument set, i.e. different on the host and Intel® Xeon Phi™ coprocessor.

OFFLOAD

Now we want to run the Intel MPI test program with some offload code on Intel® Xeon Phi™ coprocessor. The simple printout from the OpenMP thread is now offloaded to Intel® Xeon Phi™ coprocessor

```
# diff test.c test-offload.c
```

Compile and link for the Xeon host with the "`-openmp`" compiler flag as before. The latest compiler automatically recognizes the offload pragma and creates the binary for it. If required offloading could be switched off with the "`-no-offload`" flag:

```
# mpiicc -openmp -o test-offload test-offload.c
```

Execute the binary on the host:

```
#export MIC_LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH:/opt/intel/mic/myo/lib

# OFFLOAD_REPORT=2 mpirun -prepend-rank -env KMP_AFFINITY
granularity=thread,scatter -env OMP_NUM_THREADS 4 -env
LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -n 2 ./test-offload
```

Repeat the execution, but grep and sort the output to focus on the essential information:

```
# mpirun -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,scatter -env
OMP_NUM_THREADS 4 -n 2 ./test-offload 2>&1 | sort
```

All OpenMP threads will be mapped onto identical Intel® Xeon Phi™ coprocessor threads! The variable **I_MPI_PIN_DOMAIN** cannot be used because the domain splitting would be calculated according to the number of logical processors on the Xeon host!

The solution is to specify explicit proclists per MPI process:

```
# OFFLOAD_REPORT=2 mpirun -prepend-rank -env KMP_AFFINITY
granularity=thread,proclist=[1-16:4],explicit -env OMP_NUM_THREADS 4 -n 1
./test-offload : -env KMP_AFFINITY granularity=thread,proclist=[17-32:4],explicit
-env OMP_NUM_THREADS 4 -n 1 ./test-offload
```

Repeat the execution, but grep and sort the output to focus on the essential information:

```
# mpirun -prepend-rank -env KMP_AFFINITY
verbose,granularity=thread,proclist=[1-16:4],explicit -env OMP_NUM_THREADS 4 -n 1
./test-offload : -env KMP_AFFINITY
verbose,granularity=thread,proclist=[17-32:4],explicit -env OMP_NUM_THREADS 4 -n
1 ./test-offload 2>&1 |sort
```

Lab 2 - Hybrid MPI/OpenMP

Enter the directory 2_MPI_OpenMP.
Execute the following commands to build the Poisson binaries for the Intel® Xeon® host and the Intel® Xeon Phi™ coprocessor. The compilation will use the "-openmp" flag to create the hybrid version of the code. The OpenMP threads are used in the file compute.c:

```
# make clean; make
# make clean; make MIC
```

Execute the Poisson application on the host, the Intel® Xeon Phi™ coprocessor and in symmetric mode on both. The code accepts the following flags:
"-n x" change size of grid. The default is x=1000.
"-iter x" change number of max iterations. The default is x= 4000.
"-prows x" change number of processor rows. The default is computed.

```
# mpirun -env OMP_NUM_THREADS 12 -n 1 ./poisson -n 3500 -iter 10
# mpirun -env OMP_NUM_THREADS 12 -env LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -host
mic0 -n 1 ./poisson.MIC -n 3500 -iter 10
# mpirun -env OMP_NUM_THREADS 12 -host `hostname` -n 1 ./poisson -n 3500 -iter 10
: -env OMP_NUM_THREADS 12 -env LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH -host mic0 -n
1 ./poisson.MIC -n 3500 -iter 10
```

Vary the number of MPI processes and OpenMP threads (most likely different on the host and coprocessor) to optimize the performance. Use the knowledge about MPI process mapping and OpenMP thread affinity from the basic lab.