ITEA

INFORMATION TECHNOLOGY

FOR EUROPEAN ADVANCEMENT

(ITEA2 09011)

Optimize HPC Applications on Heterogeneous Architectures

Deliverable: D5-1.5.3 (L2.5.3)

Bullx SCS4 R4 Logiciel de base – version Finale

Version: V1.1

Date: Avril 2014

Authors: Bull SAS

Status: Final

Visibility: Public

HISTORY

Document version #	Date	Remarks	Author
V1.0	March 2014		D.Foueillassar
V1.1	April 2014	CEA remarks integration	M.Ospici

TABLE OF CONTENTS

1. Introduction

This deliverable describes the validation performed for the accelerators support in SCS4 R4 distribution. It firstly presents how accelerators are supported in AE4, and how these accelerators are validated.

Then it shows validation reports related to Nividia K20 and XeonPhi. It finishes by an extraction of MPI User's Guide regarding the usage of Nvidia and XeonPhi accelerators when programming with MPI.

The software is part of the SCS4 R4 delivery and is already distributed to customers since end of 2013.

2. Accelerators in SCS4 R4

The SCS4 R4 distribution supports two types of accelerators. The Xeon Phi from Intel and the NVIDIA GPU. NVIDIA GPUs supported are TESLA (M, C and K series).

Xeon Phis are supported in both offload mode (i.e. an executable runs on the host machine and some part of the code is offloaded to the Xeon Phis) and Native Mode (i.e. an executable can run natively on the Xeon Phis). For more details about Xeon Phi support, see the perfcloud report D5-1.5.1.

NVIDIA hardware can't be used "native mode" but only in offload mode, in SCS4 R4, NVIDIA driver and CUDA toolkit are installed to be able to use NVIDIA hardware.

3. Validation Procedure

3.1 NVIDIA GPU

We validate GPU hardware using three tests. These tests are performed for each new GPU and for each new machine.
Description of the three tests:

- SHOC The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite (https://github.com/vetter/shoc/wiki)

  SHOC consists in several small benchmarks designed to test GPU with programs written in both CUDA or OpenCL. We use four benchmarks.

  - The first, BusSpeedX, tests the PCI Express bandwidth between GPU and CPU memory
  - The second (MaxFlops) performs a stress test of the GPUs in order to see if we can reach good performances and good stability.
  - The third performs a [S|D]GEM to see if we can reach good performance
  - The last benchmark validates the OpenCL stack (kernelCompil)

- CUDA memtest

  Check the memory of the GPUs. This is mainly a stability test.

- Linpack

  Verification if we can reach good performance with a large benchmark.

  The linpack is optimized for the target hardware architecture. For example, for a K40, we use a linpack provided by NVIDIA optimized for the K40 hardware architecture.

When all these tests are executed successfully with good performances and good stability, we consider the GPU validated for the SCS4 R4.

3.2 Intel Xeon PHI

For the SCS4 R4 distribution we have based the validation of the Xeon Phis on the native mode.
We test:

- The base configuration (IP addresses for Xeon Phi, Ethernet interfaces bridge)
- SLURM and the PROLOG / EPILOG mechanism
- BullxMPI for Xeon Phi using IMB and OSU benchmarks

More details about the native mode can be found in the perfcloud report D5-1.5.1 or in the SCS4 R4 documentation

4. B515 + K40 validation report

This section details the NVIDIA GPU validation. As example, we show on this section the validation report for a K40 GPU on a B515 machine with the benchmarks described previously.

4.1 Test environment

Machine : B515 : 2 Xeon E5-2470
NVIDIA driver version: 319.23
CUDA 5.0
GPUs : 2xTesla K40
ECC disabled
clock: standard : memory 3GHz, core 745MHz
bullxMPI: v.1.2.4.1

4.2 Test results

4.2.1 Benchmark SHOC

(for AE4 distribution, SHOC is located in /opt/accelerators-tools/SHOC/)

– Host ↔ GPUs data transfers

```
numactl --membind=1 --cpunodebind=1 ./BusSpeedReadback -s 3 -d 1   and
numactl --membind=0 --cpunodebind=0 ./BusSpeedReadback -s 3 -d 0
```

max transfer rate : 10,3GB/s, standard deviation between tests <0,03GB/s

```
numactl --membind=1 --cpunodebind=1 ./BusSpeedDownload -s 3 -d 1   and
numactl --membind=0 --cpunodebind=0 ./BusSpeedDownload -s 3 -d 0
```

max transfer rate : 10.4GB/s, standard deviation between tests <0,02GB/s

→ Performance and stability excellent for both 'host → GPU' and 'GPU → host' data transfers.

test PASSED

4.2.2 GPUs performance

*** MaxFlops**

```
./MaxFlops -s 3 -d 0
and
./MaxFlops -s 3 -d 1
```

For GPU 0 and GPU 1 :
MAdd4-SP = 4,01**Tflops**, standard deviation = 8Gflops (single precision test)
MAdd4-DP = **1,5Tflops**, standard deviation = 3Gflops (double precision test)

*** Matrix Product**

```
./SGEMM -d 0 -s 3      and
./SGEMM -d 1 -s 3
```

For GPU 0 and GPU 1

SGEMM-N = **3080 G**flops, standard deviation = **0,6** (single precision test)
DGEMM-N = **1076 G**flops, standard deviation = **0,2** (double precision test)

For both MaxFlops and matrix products, the performances are stable there is an improvement compared to K20X

test PASSED

4.2.3 OpenCL specific test

KernelCompile : execution OK
→ OpenCL stack Operational,

test PASSED

4.2.4 Cudamemtest

/opt/accelerators-tools/gpu_memtest/sanity check OK for both GPU 0 and GPU 1

test PASSED

4.2.5 Linpack (HPL kepler from Nvidia)

(not in AE4, nvidia proprietary)

Experimental configuration :
HPL.dat :

```
$cat HPL.dat
bHPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out output file name (if any)
6 device out (6=stdout,7=stderr,file)
1 # of problems sizes (N)
70000 35000 25000 799900 780000 43007 148992 1024 40960 20960 364160 359424 276480 138240 Ns
1 # of NBs
1024 1024 320 512 64 128 192 256 320 384 448 576 640 704 768 256 64 128 192 256 NBs
0 PMAP process mapping (0=Row-,1=Column-major)
1 # of process grids (P x Q)
1 1 Ps
```

```
2 Qs
16.0 threshold
1 # of panel fact
2 PFACTs (0=left, 1=Crout, 2=Right)
1 # of recursive stopping criterium
2 16 NBMINs (>= 1)
1 # of panels in recursion
2 4 NDIVs
1 # of recursive panel fact.
1 2 0 RFACTs (0=left, 1=Crout, 2=Right)
1 # of broadcast
0 2 BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1 # of lookahead depth
1024 Columns-per-DGEMM
1 SWAP (0=bin-exch,1=long,2=mix)
96 % of DGEMM for GPU
1 L1 in (0=transposed,1=no-transposed) form
0 U in (0=transposed,1=no-transposed) form
1 Equilibration (0=no,1=yes)
8 memory alignment in double (> 0)
```

Results

```
================================================================================
T/V N NB P Q Time Gflops
--------------------------------------------------------------------------------
WR10240C2R2 70000 1024 1 2 101.32 2.257e+03
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 0.0042133 ...... PASSED
================================================================================

Finished 1 tests with the following results:
1 tests completed and passed residual checks,
0 tests completed and failed residual checks,
0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

The linpack test reaches 2,257Tflops for 2 K40 class GPUs

Test PASSED

4.3 Conclusion

Linpack performance on K40 is ~9% better compared to the K20X. This is consistent with the theoretical performances announced by nvidia.
There is a “turbo” functionality with the K40. With nvidia-smi, the core frequency can be set to 875MHz (instead of the standard 745MHz).
At 875MHz, the pciexpress bandwidth increases from 10 to 12GB/s.
Nevertheless, when this option is enabled for highly intensive applications, like linpack, the GPU needs to constantly update its frequency in order to stay on its power limit (235W).
Consequently, the linpack performance decrease with this option (from 2,257TFlops to 1,99TFlops.

5. XeonPhi validation report

5.1 Test environment

bullxmpi-mic-1.2.7.1-1
slurm-2.6.0

Features F9, F41, F47 are concerning XeonPhi

5.1.1 Feature description:

F9 : Enhance user access control on MIC
The memory of the job is only accessible to the owner of the job throughout the execution of the job on 1 MIC. Prolog / Epilog Slurm scripts specific to the use of MIC were delivered. These scripts are called at salloc. The user, who uses the MIC is created on the fly and the MIC is restart (it takes more than 1 min).
F41 : IB network access from Xeon Phi.
Allows the execution of a MPI app on multiple nodes equipped with Xeon Phi
F47 : MPI native mode available on Xeon Phi processors.
The application process, MPI or not, are running on the Xeon Phi in the same way as the host machine CPUs. A BullxMPI-MIC library compiled specifically for Xeon Phi was delivered.

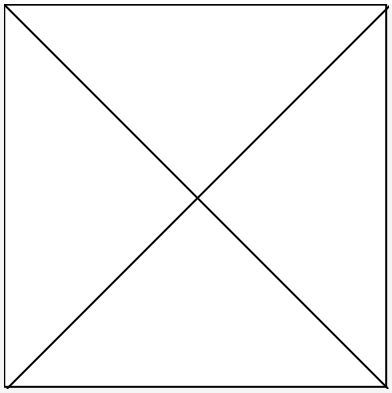
With build1 , it has not been possible to validate the F47 (native MPI mode). Slurm and bullxmpi -mic are indeed essential to MPI on MIC (16461), but the documentation Slurm and MPI (BM user's guide & BullxMPI User's guide) gave no information on the use of MIC (using DCR 16111) and the TC - BullxMPI mic was missing (16072) . Same for the F9 requiring scripts Prolog / Epilog specific Slurm for the use of MIC . These scripts were delivered with build2 and documentations Slurm and MPI (BM & BullxMPI user's Guide User's Guide) and the TC - bullxMPI mic .
Since build2 , it is possible to make tests run with MPI BullxMPI -mic .
Some issue was found during the first tests of the F9 and F47 with build2 . Some were fixed with build3 but there are still major flaws : 16983 (seg fault with IMB), Fortran defects passed on MIC (17001 and 17021) and default in C + + : 17113. A fix is being delivered to correct most of these issues).
MPI jobs must be started with salloc / mpirun . Srun can not be used . This restriction is documented in the SRB .

5.1.2 Status :

F9 : Status OK
F41 : **Status** "In progress" for IB network access. To enable this feature, it takes several nodes equipped with Xeon phi, more material received recently.
F47 : Status "In progress" for native MPI mode, waiting for a FIX for the correction of defects 16983, 16981, 17021, 17113.

- Configuration of Valx: 1 B515 node equipped with a single mic (mic-0) for testing build1 and build2, then adding a 2nd mic (mic-1) for testing build3

Tests	NB tests available	NB tests planned	NB lauched	NB succeeded tests	NB failed tests	[%] tests launched vs. planned	[%] tests succeede vs. plann
MIC	89 (oui)	89	89	87	2	100	98
TOTAL	**89**	**89**	**89**	**87**	**2**	**100**	**98**



Tests were run in 2 modes:
¬ co-processor mode (co-processor, also called offload or accelerator): In this mode, the application processes (MPI included) are executed only on the CPU of the host machine and delegate some of their work accelerators (such as for NVIDIA GPUs)
¬ Native mode (native mode): the application processes running on the Xeon Phi in the same way as the host CPU machines, and this even for MPI applications.

Balance tests on scs 4 R4 build3

Closed defects:
16072: validation:mic:AER4 build1:cannot compile for MIC with bullxmpi
16091: validation:bullxmpi:AER4 build1:mpirun error "...composer_xe_2013.2.146/mpirt/bin/intel64/mpivars.sh: No such file or directory"
16664 validation:bullxmpi:AE4.0 build2:MIC:warning message claiming not using IB
16716 validation:bullxmpi:AE4.0build2:MIC:osu_put_bw test failed "An error occurred in MPI_Win_complete"
16717 validation:bullxmpi:AE4.0build2:MIC:/etc/slurm/init_mic.sh not synchronous with MIC availability
16774 validation:bullxmpi:AE4.0build2:MIC:MPI_ROOT not defined when loading module bullxmpi-mic
16720 validation:bullxmpi:AE4.0build2:MIC:IMB-MPI1 abort "Segmentation fault"
16697 validation:bullxmpi:AE4.0build2:MIC:osu_alltoall stays blocked for 4 processes or more
16667 validation:bullxmpi:AE4.0 build2:MIC:Initialisation takes a huge 1mn22s time, *la documentation Slurm BM user's guide documente ce fonctionnement.*
16698 validation:AE4.0bulid2:bullxmpi:mandatory use of /etc/slurm/init_mic.sh missing in examples
16699 validation:AE4.0bulid2:bullxmpi:ssh to mic only possible when slurm allocation active and after

init_mic.sh invocation

Opened defects:
17113 validation:bullxmpi:AE4.0 build3:MIC:Cannot execute a program that uses the C++ MPI interface with bullxmpi-mic
17021/17020 validation:bullxmpi:AE4.0 build3:MIC:Compilation Fortran90 with card "use mpi" failed with module bullxmpi-mic
16996 validation:bullxmpi:AE4.0 build3:MIC:MIC can be 'online' while not available for connections
16983 validation:bullxmpi:AE4.0 build3:MIC:Intel MPI Benchmark Accumulate test abort " Segmentation fault"
16981 validation:bullxmpi:AE4.0 build3:MIC:SIGKILL on timeout for test osu_alltoall crashes a MIC
16806 validation:bullxmpi:AE4.0 build2(->build3):MIC:Intel MPI Benchmark IMB-EXT blocked in Unidir_Get

5.2 Conclusion

XeonPhi native mode validation is not totally finished (March 2014) but two new Xeon Phi have been provided to validation and all known bugs correction should be made available end of March. Delivery to customers is already done under early shipment, general availability planned in June.

6. MPI User's Guide: Using Accelerators in MPI Programs

bullx MPI supports the Bull accelerator hardware (such as B505).
Two types of accelerator are provided with bullx scs 4:

- GPUs from NVIDIA
- Xeon Phi from Intel.

This section describes how these two accelerators can be programmed, and provides an overview of how to use these accelerators on a bullx MPI application.

6.1 NVIDIA GPUs

NVIDIA provides a proprietary environment called CUDA.

To understand the cuda environment, an extensive documentation can be found in http://developer.nvidia.com/category/zone/cuda-zone
Documentation is also available in the CUDA installation directory: /opt/cuda/cuda_version/doc

The CUDA programming model is based on threads. With CUDA, the programmer is encouraged to create a very large number of threads (several thousand). Threads must be structured into block. The card will thus schedule these threads in free compute cores.

The Khronos Group (a consortium of companies and research groups) has defined a specification called OpenCL. OpenCL defines a programming model close to the CUDA programming model. Because the specification is open, several companies have proposed an implementation of OpenCL for their hardware.

The bullx scs 4 software supports OpenCL for NVIDIA accelerator. Thus, it is possible to write an OpenCL programs and run it on NVIDIA accelerators. 22 bullx MPI User's Guide

6.2 Xeon Phi (MIC)

Note:

Xeon Phi supports several execution models. In this bullx scs 4 release, we support both the accelerator mode (also called offload or co-processor mode) and native mode for the Xeon Phi. On native mode, there are MPI processes on Xeon Phi, where in accelerator mode there are only MPI processes on Xeons, which delegate part of their work to their accelerators.

The Xeon Phi accelerators are automatically configured to be fully usable in native mode: each Xeon Phi holds an IP address. In native mode, the binaries of the application must be compiled for the Xeon Phi architecture: Xeon Phi binaries are not compatible with the hosts processors. Users can compile an application for the Xeon Phi by using the -mmic compiler flag. To run a native application, it is not sufficient to compile the application. The application must be available on the Xeon Phi.

The most flexible option is to copy the application on the Xeon Phi (via scp for example). The Xeon Phi must be available: see *Section 2.8.3 bullx MPI and Accelerators* for more information about this copy. Xeon Phi jobs can take long time to complete due to EPILOG/PROLOG scripts.

For more information about Xeon Phi and SLURM see *Section 2.9* in *the bullx BM User's Guide*. For more information about using bullx MPI on Xeon Phi see *Section 2.8.3. bullx MPI and Accelerators*

Although the Xeon Phis are configured for the native mode, they can be also used in co-processor mode. To use a Xeon phi as a co-processor, the programmer should use the new Intel LEO directives (Language Extensions for Offload). These directives, which exist for both Fortran and C are implemented in the Intel compilers v.13.

With LEO, some directives are designed to launch data transfers, other are designed to compile and offload code to the MIC side.

With LEO, it is possible to offload an OpenMP section: OpenMP threads will be created on the MIC side. Because the MIC can have more than 60 cores, and each core can manage four threads, OpenMP thread number must be adapted to this architecture. Nevertheless, it can be challenged to develop an OpenMP section that scale on a large numbers of cores.

Intel TBB (Threading Building Blocks) and Intel Cilk Plus can also be used in C programs (not Fortran) to program the Xeon Phi.

The description of these programming models is not in the scope of this document. Programmer must read the Intel compiler documentation.

LEO uses a low-level library called COI. COI can be used directly to offload code to the MIC. Nevertheless, this library is not well documented yet and we discourage the use of this method to offload code on the MIC.

6.3 bullx MPI and Accelerators

Using accelerators with MPI means that several MPI processes will use one or more accelerator during the program execution. There are some execution options to know in order to have good performance for both MIC and GPU.

GPU
On GPU hardware with several CPU socket and several GPUs (like the B505 blade), best data transfer performance is achieved when a MPI process is pinned on the CPU socket closest to the GPU used. Several CUDA functions (such as cudaSetDevice or cudaChooseDevice) can be used on the MPI program to choose the correct GPU.

Xeon Phi Native mode
Bullxmpi-mic is a version of Bullxmpi designed for jobs using Xeon Phi nodes. This means that some or all of the MPI ranks can run on Xeon Phis, while others can run on Xeons.
To use it, one needs to load Intel compilers (at least the 2013 version) and Bullxmpi-mic in the user environment by executing the following commands (in this exact order):

```
source /opt/intel/composerxe/bin/compilervars.sh
```

```
module load bullxmpi-mic/bullxmpi-mic-1.2.5.1
```

To compile a MPI executable for Xeon Phi, mpicc (or mpif90) must be used with the -mmic option. This generates Xeon Phi executables and selects Xeon Phi versions of shared libraries discarding x86_64 ones.

For example:

```
mpicc -mmic prog.c -o mic_prog
```

mic_prog is a Xeon Phi executable and must be available on the Xeon Phi before launching it via mpirun. In bullx scs 4, the Xeon Phis are available after the reservation (salloc) and after the execution of the init_mic.sh script within the reservation. Thus, users must copy the application (mic_prog, in our example) when the Xeon Phis are available. Users can also choose to add mic_prog on a NFS mount to avoid an explicit copy to the Xeon Phi. For more information about NFS mount see the *bullx MC Administration Guide,*

Note

A Xeon Phi is available when:
- it is allocated (via salloc)
- the init_mic.sh command is launched after the allocation Attention: init_mic.sh takes ~1 min to complete.

Xeon Phis are generally seen as normal nodes:

- they have hostnames (following the pattern '${host_nodename}-mic${X}', with X in 0,1)
- they can be connected to by ssh
- NFS mounts can be configured by the system administrator (see *Appendix G* in the *bullx MC Administration Guide* for details)
- MPI jobs can be launched on them

However they do not have SLURM daemons running on them (see *Section 2.9 MIC configuration and usage* in the *bullx BM User's Guide*), which implies that:

- Xeon Phi nodes do not appear in the result of the sinfo command and cannot be allocated directly
- they are a resource of their host nodes in SLURM. The user must allocate those hosts nodes instead
- MPI jobs are launched on them only using mpirun, not using srun

Launching a job on Xeon Phis requires these 4 steps:
1. Allocating nodes with Xeon Phi coprocessors in SLURM using the --gres option.

2. Running /etc/slurm/init_mic.sh to ensure that Xeon Phis are available (~1min to complete)

3. Copy the application to the Xeon Phi (if no NFS mounts or no extra overlays are configured, see *Appendix G* in the *bullx MC Administration Guide*)

4. Running the MPI program with mpirun (because srun is not available for native Xeon Phi applications)

At the end of the allocation, the EPILOG script is launched. The Xeon Phi node can stay ~1min in completing state (see *Section 2.9 of the bullx BM User's Guide* for more details).

Note

Xeon Phi nodes must be automatically allocated in exclusive mode. SLURM should be configured in this way, see *Section 2.9 of the bullx BM User's Guide* for more details.

For example, to launch a job on all the processors of four Xeon Phis, you need to allocate two nodes with two Xeon Phis each:

```
salloc -N2 --gres=mic:2 mpirun mic_prog
```

Bullxmpi-mic selects automatically Xeon Phi nodes of a job for a Xeon Phi executable, and Xeon nodes for a x86_64 executable. You can launch hybrid jobs after compiling a program into two executables (one for Xeon and one for Xeon Phi) and launching both mpirun using the ':' character as a separator between the commands.

For example, on a cluster with two Xeon Phi per nodes, the following commands are equivalent:

```
salloc -N2 --gres=mic:2
/etc/slurm/init_mic.sh
mpirun -np 16 -bynode xeon_prog : -np 60 -bynode mic_prog
```

and

```
salloc -N2 --gres=mic:2 -w node1,node2 /etc/slurm/init_mic.sh mpirun -np 16 -bynode -H node1,node2
xeon_prog : -np 60 -bynode -H node1-mic0,node1-mic1,node2-mic0,node2-mic1 mic_prog
```

Note

We assume in these examples that mic_prog is on a NFS mount, so the step 3 (copy mic_prog to the Xeon Phis) is useless.

Both launch 16 ranks on 2 Xeons and 60 on their 4 Xeon Phi (here -bynode picks nodes in a round robin fashion). Nodes are chosen manually in the last command, whereas in the first one mpirun guesses nodes to pick: Xeons nodes allocated by SLURM for the Xeon executable and corresponding Xeons Phis nodes for the Xeon Phi executable. Chapter 2. Using bullx MPI Parallel Library 25

6.4 Xeon Phi co-processor Mode

Note

For the co-processor mode, the application must be compiled for the host (not for the Xeon Phi). The application runs on the host and offload some part of the computation to the Xeon Phi. This mode is close to the GPU mode.

In this mode, Xeon Phis are used as coprocessors to a node, which means that the MPI runtime is not aware of them: the Xeon binary offloads the MIC code. Therefore the normal flavor of Bullxmpi has to be used, not Bullxmpi-mic. To use Xeon Phi in this mode it is advisable to:

1. Set the environment:

```
source /opt/intel/composerxe/bin/compilervars.sh
module load bullxmpi/bullxmpi-1.2.6.1
```

2. Compile your app containing OpenCL or offload pragmas normally.
(See /opt/intel/${composerxe}/Samples/en_US/[C++ | Fortran]/mic_samples/ for examples):

```
mpicc offload_prog.c -o offload_prog
```

3. Reserve the Xeon Phi with **--**gres and initialize them:

```
salloc -N1 --gres=mic:2 /etc/slurm/init_mic.sh
```

Note

If the Xeon Phis are intended to be used only in offload mode for each node in the cluster, the system administrator can disable the PROLOG/EPILOG scripts (to remove the time needed to reboot the Xeon Phis). See *section 2.9.4 Managing security of the Xeon Phis* in the *bullx BM User's Guide*, for more details about the impacts of disabling PROLOG/EPILOG.

4. Launch your job with srun or mpirun:

```
mpirun -N 2 offload_prog
```

There are two executions issues with the Xeon Phi co-processor.

- The first issue is the affinity problem, which is basically the same than for GPU hardware. LEO provides ways to set up the correct GPU for a particular MPI task.
- The second issue is the thread affinity on the MIC. With the Intel OpenMP software stack, it is possible to control the number of threads created by each MPI task on the MIC (with the MIC_OMP_NUM_THREADS environment variable), or to control thread affinity (MIC_KMP_AFFINITY). These variables are defined in the Intel Compilers documentation.

The user can specify thread affinity for each MPI process launched through environment variables. See Intel Documentation for Xeon Phi for more details.

6.5 Usage example

```
mpirun -x MIC_PREFIX=MIC -x MIC_OMP_NUM_THREADS=48 -x
MIC_KMP_AFFINITY=exeplicit,proclist=[4-51],verbose ./mpi_omp_mic_test
```

When launching several executables with one mpirun command, it can be useful to control where OpenMP threads will be bound. See the standard documentation provided with the Intel compilers for more details about the runtime OpenMP provided by the Intel compilers (especially the KMP_AFFINITY syntax).