Hands-On: Running DL_POLY_4 on Intel Knights Corner

Alin M Elena*

23rd of March 2017, Sofia, Bulgaria

1 Building the code

Molecular dynamics techniques grew rapidly in the last twenty years. The growth was fuelled by development of new scalable mathematical algorithms, availability of powerful hardware and better availability of ready to use software packages. DL_POLY is one of these packages, widely adopted by the computational physics and material science communities.

DL_POLY started its life in 1992 at Daresbury Laboratory, now part of Science & Technology Facilities Council in United Kingdom, with a first public release in 1993. The main developers for the current version are W Smith and IT Todorov. DL_POLY is a general classical molecular dynamics code and was used to simulate macro molecules (both biological and synthetic), complex fluids, materials and ionic liquids. DL_POLY also plays an important role as sandbox for both development of new methods and algorithms for molecular dynamics and testing of emerging hardware technologies[1] and [2]. The core code is written in Fortran 95/2003 standards and optimised for distributed systems using domain decomposition, also OpenMP and CUDA ports exist as contributions to DL_POLY but not part of the official distribution. DL_POLY is free of use for academics pursuing non-commercial research and available for licensing for the rest.

The Intel Xeon Phi co-processor is a novel accelerator technology that provides few attractive features as: many cores, 60 cores with 240 hardware threads for the mid model, low power consumption, the same set of instructions as an Intel CPU, supports popular and standardised programming models as MPI and OpenMP and a theoretical peak of 1 TFlops in double precision.

Start by obtaining a licence for DL_POLY_4 from `http://www.scd.stfc.ac.uk/SCD/44516.aspx`, is free of charge for academic research. Versions of DL_POLY_4 to be used for this exercises are already in **home/alin/sofia** folder.

Step 1. Connect to avitohol, get an interactive session and set your environment to build the code[1]

*Computational Scientist at STFC Daresbury Laboratory, contact alin-marin.elena@stfc.ac.uk

[1] everytime you login to the machine the environment will need to be setup again

Snippet 1: Source the environmet

```
qsub -I -q edu
source /opt/intel/parallel_studio_xe_2017.2.050/psxevars.sh

```

Step 2. Build the code for Xeon processors the commands shown in snippet 2 need to be executed. A copy of the script can be found in **/home/alin/sofia/scripts/build-xeon.sh**. The source code we use at the moment is **dl-poly-stfc-omp.tar.xz**.

Snippet 2: Build the code for Xeon processor

```
#!/usr/bin/env bash

cp /home/alin/sofia/dl-poly-stfc-omp.tar.xz ~/
cd ~/
tar -xvf dl-poly-stfc-omp.tar.xz
cd dl-poly-stfc-omp
mkdir -p build-mpi && cd build-mpi
FC=mpiifort FFLAGS="-DCHRONO -fopenmp  -O3 -xHost -D__OPENMP"  \
cmake ../
make -j4
```

If all went correctly you shall find the **DLPOLY.Z** executable in **$HOME/dl-poly-stfc-omp/build-mpi/bin**. Note this path since will be useful in second part of the exercise.

Step 3. Build the code for Xeon Phi co-processors (native mode) the commands shown in snippet 3 need to be executed. A copy of the script 3 can be found in **/home/alin/sofia/scripts/build-mic.sh**.

Snippet 3: Build the code for Xeon Phi co-processor native

```
#!/usr/bin/env bash

cd ~/
cd dl-poly-stfc-omp
mkdir -p build-mic && cd build-mic
FC=mpiifort FFLAGS="-DCHRONO -fopenmp  -O3 -mmic -D__OPENMP"  \
cmake ../
make -j4
```

If all went correctly you shall find the new **DLPOLY.Z** executable in **$HOME/dl-poly-stfc-omp/build-mic/bin**. Note this path since will be useful in second part of the exercise.

Step 4. Build the code for Xeon Phi co-processors (offload mode) the commands shown in snippet 4 need to be executed. A copy of the script can be found in **/home/alin/sofia/scripts/build-offload.sh**. The source code we use at the moment is **dl-poly-stfc-phi.tar.xz**.

Snippet 4: Build the code for Xeon Phi co-processor offload

```
#!/usr/bin/env bash

cp /home/alin/sofia/dl-poly-stfc-phi.tar.xz ~/
cd ~/
tar -xvf dl-poly-stfc-phi.tar.xz
cd dl-poly-stfc-phi
cd source
make offload
```

If all went correctly you shall find the new **DLPOLY.Z** executable in **$HOME/dl-poly-stfc-phi/execute**. Note this path since will be useful in second part of the exercise.

Step 5. Build the code for Xeon processors the commands shown in snippet 5 need to be executed. This version is a reference version in which we disabled OpenMP, or what is called a pure MPI version. A copy of the script can be found in **/home/alin/sofia/scripts/build-mpi-pure.sh**.

Snippet 5: Build the code for Xeon processor - MPI pure

```
#!/usr/bin/env bash

cd ~/
cd dl-poly-stfc-omp
mkdir -p build-mpi-pure && cd build-mpi-pure
FC=mpiifort FFLAGS="-DCHRONO -O3 -xHost"  \
cmake ../
make -j4
```

If all went correctly you shall find the new **DLPOLY.Z** executable in **$HOME/dl-poly-stfc-omp/build-mpi-pure/bin**. Note this path since will be useful in second part of the exercise.

Step 6. Can you do a MPI pure version of the co-processor version? If answer is no, do not despair a scipt version is in **/home/alin/sofia/scripts/build-mic-pure.sh**

Congratulate yourself if you reached this point. You have managed to build DL_POLY_4 to be able to run on Xeon and Xeon Phi in all possible compinations: native (both CPU and coprocessor), MPI symmetric and offload. In the next section we will actually run the code.

2 Running

2.1 Reference

Step 7 Obtaining reference data on the Xeon. For this step we will use the binary build for host in a previous step. Of course we will need some input data for DL_POLY_4. We will use a protein solvated in water, gramidicin. All files needed are available in **/home/alin/sofia/gramidicin**. Go to the folder you have the DL_POLY_4 binary and copy the input files and run the executable on one mpi process. Instructions can be found, if needed in snippet 6.

Snippet 6: Running for reference data on Xeon

```
cd $HOME/dl-poly-stfc-omp/build-mpi-pure/bin
cp /home/alin/sofia/gramidicin/* .
  # run DL_POLY_4 on with one MPI process
mpirun -n 1 ./DLPOLY.Z
```

This shall take around 30s. If successful one shall see few new files created, in between them one called **OUTPUT**. This file contains all the data needed to characterize the times of our run. For this I have created few helper scripts, copy them to the current folder by

Snippet 7: Copy timing scripts for native builds

```
cp /home/alin/sofia/scripts/omp/*.sh
```

You shall see now for scripts: **linked.sh**, **shake.sh**, **time.sh** and **twobody.sh**. These scripts will extract from the **OUTPUT**, in order, the following times: t_l, t_s, t_{st} and t_F. t_l is the time needed to create the neighbours lists, t_s is the time to compute the holonomic constraints, t_F is the time to compute the two body forces, and t_{st} is the total time to integrate a time step. To extract these times one shall run the scripts as showin in snippet 8.

Snippet 8: Timing data extraction

```
./time.sh 5 10 OUTPUT 42
./linked.sh 5 10 OUTPUT 42
./twobody.sh 5 10 OUTPUT 42
./shake.sh 5 10 OUTPUT 42
```

Step 8. Record the times from above in the next table. Rerun with 2, 4, 8 and 16 processes and after each run extract the times and record them in the same table, in their columns.

#MPI	t_l	t_s	t_F	t_{st}	eff
1					
2					
4					
8					
16					

Step 9. Compute the efficiency. In this step we will compute column **eff** in the above table, using the following formula

$$\text{eff} = \frac{t_{st}(1)}{t_{st}(p)p} \tag{1}$$

where $t_{st}(p)$ meants the time needed to integrate a time step on p MPI processes. You shall have all the data needed to compute the efficiency.

Step 10. Running reference data on Xeon co-processors. In this step we will use the same system as in the previous step and the executable we build in the first part of the exercises for this case. See the snippet 9 for how to copy the data and run the code on mic0.

Snippet 9: Running for reference data on Xeon co-processor

```
cd $HOME/dl-poly-stfc-omp/build-mic-pure/bin
cp /home/alin/sofia/gramidicin/* .
  # run DL_POLY_4 on with one MPI process
I_MPI_MIC=enable mpirun -n 1 -host $(hostname)-mic0 ./DLPOLY.Z
```

This run will take much longer. Can you imagine why? Extract the times using the previous scripts as shown in snippet 8 and record them in a similar table as the one above. How do they compare with the similar case for host only?

Step 11. Rerun DL_POLY_4 on co-processor using 2, 4, 8, 16, 30, 60, 120 MPI processes, extract and record the times and compute the efficiency.

2.2 OpenMP

Step 12. In this step we will benchmark the code using OpenMP and try to obtain a scalability curve for runs with one MPI process and 2 MPI processes. Running the code with one MPI process and one OpenMP thread on CPU. The snippet 10 shows the commands needed to run.

Snippet 10: OpenMP on Xeon processor

```
cd $HOME/dl-poly-stfc-omp/build-mpi/bin
cp /home/alin/sofia/gramidicin/* .
export OMP_STACKSIZE=200M
  # run DL_POLY_4 on with one MPI process and one Thread
OMP_NUM_THREADS=1 mpirun -n 1 ./DLPOLY.Z
```

Use the scripts from the snippet 8 to extract the different times of interest. Remember first to copy the scripts in current location. Record the times in the next table.

Step 13 Rerun the executable changing the number of threads to 2, 4, 8, 16 and 32. Each time extract and record in the table the times. Variable OMP_NUM_THREADS controls the number of OpenMP Threads.

#OMP	t_l	t_s	t_F	t_{st}	eff
1					
2					
4					
8					
16					
32					

Step 14. Compute the efficiency. In this step we will compute column **eff** in the above table. The formula is the same as previously with p being replaced by the number of threads.

Step 15. Can you say what affinity was used for threads? If you do not kow what is thread affinity, this article may help [2]. Hint: set variable KMP_AFFINITY=verbose and rerun one of the previous cases, for example 8 threads.

[2] https://goo.gl/xXgjiS

Step 16 using the following snippet 11 redo the OpenMP table by placing the threads to specific cores.

Snippet 11: Running with specific affinity

```
export OMP_STACKSIZE=200M
export OMP_NUM_THREADS=8
export KMP_PLACE_THREADS="1T"
export KMP_AFFINITY=compact
mpirun -n 1 ./DLPOLY.Z
```

Step 17. Redo the OpenMP table using an affinity of your choice but this time using two MPI processes.

Step 18. Running on Xeon Phi co-processor. Since this step is more involved I have prepares already a script to run on Xeon Phi. The script is in **/home/alin/sofia/scripts/run-mic.sh**. Inspect the content. Full steps to run it are in snippet 12.

Snippet 12: OpenMP on Xeon Phi co-processor

```
cd $HOME/dl-poly-stfc-omp/build-mic/bin
cp /home/alin/sofia/gramidicin/* .
cp /home/alin/sofia/scripts/run-mic.sh .
export OMP_STACKSIZE=200M
ssh $(hostname)-mic0 $(pwd)/run-mic.sh
```

Use the previous scripts to extrat the times and record them.

Step 19. Open the script and replace 4T in it by 1T, 2T and 3T. Run for each new value, extract and record the times. Hint 1T means only one thread for each core will be used, 2T two threads for each core and so on.

For a full benchmark on a Xeon Phi one shall run for each 1T, 2T, 3T and 4T with number of threads: 1, 2, 4, 8, 16, 30 and 60. Since this is time consuming consider it as a homework.

2.3 Offload

Step 20. Using Xeon and Xeon Phi at the same time via offload. For this step we will use the executable created earlier. The snippet 13 shows you how to copy the data and needed scripts. As in the previous step we will use a script **/home/alin/sofia/scripts/run-offload.sh** to run the code for simplicity.

Snippet 13: Sample script to run offloaded code

```
cd $HOME/dl-poly-stfc-phi/execute
cp /home/alin/sofia/gramidicin/* .
cp /home/alin/sofia/scripts/phi/* .
cp /home/alin/sofia/scripts/run-offload.sh .
#now run the code
./run-offload.sh
```

The content of the script is very simple at the moment but we will complicate it later, see snippet 14 (for the complete content look into the actual script).

Snippet 14: Run the code in offload mode

```
#!/usr/bin/env bash

export MIC_LD_LIBRARY_PATH=...

export OMP_STACKSIZE=100M

mpirun -n 1 ./DLPOLY.Z
```

On how many threads did the code run on Xeon? What about the Xeon Phi?

Step 21. Changing the number of threads on each device. Change your previous script and add before the mpirun line the following lines, snippet 15. This will use 8 threads on Xeon and 60 threads on Xeon Phi.

Snippet 15: Change number of threads on devices

```
export MIC_ENV_PREFIX=PHI
export OMP_NUM_THREADS=8
export PHI_OMP_NUM_THREADS=60
```

Use the timing scripts you copied to extract the times as before and record them. Vary the thread numbers on Xeon Phi as before (1,2,4,8,16, 30 and 60). For each case extract and record the times. Can you compute some kind of efficiency?

Step 22 Can you check what affinity is used on the card? Hint: use PHI_KMP_AFFINITY=verbose in your previous script. Add more threads per core (2T, 3T, 4T) and investigate the effects.

Step 23. Offloading to multiple cards. Each node on the cluster has two Xeon Phi cards. In previous steps we used only one of them. We will use the script in **/home/alin/sofia/scripts/run-offload-2.sh** copy it to the same place with the executable, a snippet is shown in 16.

Snippet 16: Offload code to two cards

```
#!/usr/bin/env bash

export MIC_LD_LIBRARY_PATH=...

export OMP_STACKSIZE=100M
export MIC_ENV_PREFIX=PHI
export OMP_NUM_THREADS=8
export PHI_OMP_NUM_THREADS=60

rm -f config2mpi
cat  > config2mpi <<EOF
-n 1 -env OFFLOAD_DEVICES=0 ./DLPOLY.Z
-n 1 -env OFFLOAD_DEVICES=1 ./DLPOLY.Z
EOF
mpirun -configfile config2mpi
```

2.4 MPI Symmetric

Step 24. In this stage we will run the code on both Xeon and Xeon Phi but this time using a different method. We will use the same code but once build for host and once build for co-processor. We already have the executables build, for simplicity we use the reference ones. The following snippet 17 shows the steps needed to run.

Snippet 17:

```
cd $HOME/dl-poly-stfc-omp
mkdir symmetric
cd symmetric
cp /home/alin/sofia/scripts/run-symmtric.sh .
cp /home/alin/sofia/scripts/omp/* .
cp /home/alin/sofia/gramidicin/* .
./run-symmetic.sh
```

Once it is finished one shall extract the times as before and compare them with previous reference runs.
Snippet 18 shows the script use to run one MPI process on each device.

Snippet 18: MPI Symmetric to two cards

```
#!/usr/bin/env bash

export MIC_LD_LIBRARY_PATH=...

rm -f configsym
cat  > configsym <<EOF
-n 1 -host $(hostname) ../build-mpi-pure/bin/DLPOLY.Z
-n 1 -host $(hostname)-mic0 -env LD_LIBRARY_PATH=$MIC_ ../build-mic-pure/bin/DLPOLY.Z
-n 1 -host $(hostname) ../build-mpi-pure/bin/DLPOLY.Z
-n 1 -host $(hostname)-mic1 -env LD_LIBRARY_PATH=$MIC_ ../build-mic-pure/bin/DLPOLY.Z
EOF

export I_MPI_MIC=on
mpirun -configfile configsym
```

References

[1] DL_POLY_4, -, DL_POLY, **http://ccpforge.cse.rl.ac.uk/gf/project/dl-poly/**, 1992-2017. 1

[2] I. T. Todorov, W. Smith, K. Trachenko, and M. T. Dove, *DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism*, J. Mater. Chem., **16**, p. 1911, 2006, doi: 10.1039/B517931A, URL http://dx.doi.org/10.1039/B517931A. 1