# Tutorial 2.Comparison of parallelisation strategies used in molecular dynamics programs.

## Objective

*The aim of this tutorial is to highlight the differences in parallel scaling between two parallelisation strategies: atom decomposition (replicated data) and domain decomposition. In addition, the tutorial will give the student practice in running parallel programs on an HPC computer, collecting and plotting benchmark results.*

## Application codes

*For the exercises two versions of a popular moelcular dynamics code will be used:* **DL_POLY classic** *(for replicated data) and* **DL_POLY v4** *(for domain decomposition). This program was chosen because it is relatively simple to install and run and allows us easily to test directly the main parallelisation strategies.*
*For more information about* **DL_POLY Classic** *see*

## Program Input

*As input we have a chosen a simple system of Lennard-Jones particles which can be used to model simple, atomic liquids (e.g. liquid argon). No electrostatic interactions are included so performance should be entirely due the parallelisation strategy to non-bonded forces.*

## Computer system

*We shall use Cineca's Eurora cluster for the exercises. This system has 2 8-core Intel SandyBridge processors and 2 NVIDIA M2030 GPUs or 2 Xeon PHIs per node (but the GPUs and XeonPHIs are not used in this tutorial) and uses the PBS batch system for submitting jobs. See http://hpc.cineca.it for more details.*

## Exercises brief description.

1. *Replicated data strategy with DL_POLY Classic. Simple strong, scaling example. Calculate parallel efficiency, performance in ns/day.*
2. *Domain decomposition strong scaling. Same as exercise 1 but using DL_POLY v4.*
3. *(Optional) Analyse the MPI communication profiles of both versions of the program to understand better the different parallelisation algorithms.*

# Exercise 1.  Replicated data strategy with DL_POLY (classic)

## Preparation

*Download and unpack the simulation files for DL_POLY classic. We recommend you use the $CINECA_SCRATCH directory:*

```
cd $CINECA_SCRATCH
tar xvf exercises.tar.gz
cd exercises/T1
```

*You will find different sub-directories corresponding to starting configurations with different numbers of Argon atoms. Choose one of these directories*

*Each directory will have the following three files:*

1. *CONFIG - atomic coordinates*
2. *CONTROL - run parameters*
3. *FIELD - force field parameters.*

*Unless you know DL_POLY well, you should only edit the CONTROL file. In particular you might like to change the parameter:*

```
steps                    10000
```

*In order to change the number of time steps to be simulated.*

## Simulation runs

*You will find an example job script for PBS which you should modify before submitting to the PBS batch system. Copy this into one of the sub-directories:*

```
cp exercise.pbs Ar128K
cd Ar128K
( modify as appropriate exercise.pbs)
qsub exercise.pbs
```

*First do a test run  with, say, 1 core to see how long a typical run might last and to give an estimate of the walltime parameter to be passed to PBS.*

*Run  DL_POLY classic for n=2,4,8,12,24, cores etc .*

**Important***:* **DL_POLY writes always to a file called OUTPUT so this should be renamed between runs.**

## Performance Analysis

*The performance of a parallel, MD program can be expressed in various ways:*

1. *The inverse of the wall time for a given problem size (e.g. no. of time steps).*
2. *ns/day, i.e the simulation time in ns which can performed in 24 hours of wall time.*
3. *The parallel efficiency or speed-up.*

*The second option is generally more useful for researchers wishing to plan an MD project, while the third is often requested by resource providers and gives information on how well the program has been parallelised or is being used.*

### Obtaining the walltime

*The walltime W for a DL_POLY run can be found near the end of the OUTPUT as:*

```
 time elapsed since job start =         253.886 seconds
```

*You should collect the wall times as from the output files and create a table, e.g*

| *#nprocs* | *walltime/s* |
|---|---|
| *1* | *380* |
| *2* | *315* |
| *4* | *283* |

*If you have many files you could use a UNIX script containing lines such as:*

```
tail OUTPUT.* | awk '/elapsed/{print $7}'
```

*You could also obtain the wall time from a PBS job script (bash):*

```
module load dl_poly/1.9
start_time=$(date +"%s")
mprun -np 4 DLPOLY.X
end_time=$(date +"%s")
walltime=$(($end_time-$start_time))
echo "walltime $walltime"
```

### Performance in ns/day

*For walltime W (seconds) this is given by:*

P = no. of. time steps * time step (ns) * 86400 / W
(86400 = seconds in 24h)

For DL_POLY the time step is given in picoseconds (1ns = 10^3 ps).

### Parallel Efficiency

The Parallel Efficiency En at n cores is given by:

En = 100 * Pn / (n * P1)

where Pn is the performance at n cores, P1 for 1 core.

### Plotting performance data

This can be conveniently done in gnuplot. For example if results.dat contains two columns, # cores and walltime, the performance can be plotted as:

```
module load gnuplot
plot “results.dat” u 1: (1/$2)
```

For performance in ns/day, assuming 1000 time steps and a time step of 1e-3 ps (=1e-6 ns),

```
nsteps=1000
dt=1e-6
daysecs=86400 # no. of secs in 24hrs
set ylabel “ns/day”
set xlabel “#cores”
plot “results.dat” u 1:(nsteps*dt*daysecs/$2)
```

You might like to compare this behaviour with ideal scaling behaviour. For example, if we find that the walltime (w1) for 1 proc is 380s you can do this:

```
w1=380
plot “results.dat” u 1:(nsteps*dt*daysecs/$2) tit “simulated”, u 3:
($2*nsteps*dt*daysecs/w1) tit “ideal”
```

**Question** : Assuming you had a grant of 100k core hours. How many nano-seconds of simulation could you hope to simulate with this time at the maximum performance? How many calendar days would it take ? Now consider using half the no. of cores of maximum performance. How long can you simulate now (in nano seconds)?

**Question**. *For which system size do you get the best scaling behaviour and why ?*

## Exercise 2. Domain decomposition with DL_POLY v4.x

*In this exercise you should repeat the runs you did in Exercise 1 but using DL_POLY v4.x which uses domain decomposition.*
*Consider the following questions:*

1. *How does the parallel scaling compare to the replicated data?*
2. *How does the paralllel scaling compare with system size? As a rule of thumb, how many atoms/core do you need before the program stops scaling?*

## Exercise 3. (optional) Profiling analysis of MD codes with Scalasca

*In this exercise we will use a parallel profiling tool, Scalasca, to identify the differences in parallel behaviour of the two codes in terms of communication or synchronization calls.*

### step 1. Run the scalasca version of DL_POLY in a PBS script

```
# for DL_POLY Classic
# exe=DL_POLY.X.scalasca
# for DL_POLY 4
# exe=DL_POLY.Z.scalasca

module load autoload scalasca
scalasca -analyze mpirun -np 4 $exe
```

### step 2. Analyze scalasca-instrumented output

*A successful run should generate a directory called epik_DLPOLY_4_sum (or similar). Analyze this directory with the examine option of scalasca*.

```
scalasca -examine epik_DLPOLY_4_sum
```

*Repeat this procedure for both versions of DL_POLY.*

## Analysis

*You might like to open two scalasca sessions (one for each program) so you can compare them directly. Scalasca allows you to compare various metrics -  some you could look at include:*

- *Load balancing - Select Computational balance (Metric tree)+System tree*
- *Bytes transferred. (Metric tree)*

*Question: How does the load balancing compare between the two cases ?*
*(i.e. the ratio of the process with the largest load compared to that with the smallest)*

**Question***: Which version transfers more bytes via collective calls and which via point-to-point?*
**Question:** *In DL_POLY2, which MPI call transfers the most data?*