MASTER IN HIGH PERFORMANCE

COMPUTING

High-Performance Computing Technologies

Candidate: **Malenza Giulio**

Candidate: **Nazari Zainab**

Candidate: **Masini Alessandro**

Anno Accademico 2022/2023

1 MPI optimization

Starting from the work of Zaynab and Alessandro on the sequential and openmp optimization, the goal of this section was to provide the code with an mpi parallelization, in order to execute simulations on distributed architecture.
To this aim, two strategies were developed. The first one consisted in distributing the particles across the mpi processors and then sending messages during the force computations, with each particle having to interact with all the others. This strategy optimizes memory since each processor has only a portion of particles; for this reason, it is recommended for systems with many particles. However, it requires a large (equal to the number of processors) number of message exchanges, so it may be inefficient with small systems. The pseudocode showing how to develop loops inside the force_compute routine is the following:

Algorithm 1 Nested loops force computes messages

```
for l = 0, nprocs do
    for i = 0, natoms_loc do
        for j = 0, natoms_recv do
            Perform calculations
        end for
    end for
    Copy the atoms to send into a buffer
    Send particles across processors
end for
```

The second strategy consisted in sending all the particles before entering the for loops with a `MPI_Broadcast`, then during the computations each processor executes only one part of the calculation. At the end of the calculation, the information is sent back to the starting processor, which sums it with a `MPI_Reduce`.

Algorithm 2 broadcast and reduce mpi operations

```
MPI_Bcast(...)
for i = 0, natoms − 1, i+ = nprocs do
    ii = i + rank
    if ii >= natoms − 1 then
        for j = ii, natoms do
            Perform calculations
        end for
    end if
end for
MPI_Reduce(...)
```

This second strategy was tested using two systems with 108 and 2916 argon atoms. Simulations were performed using the computational nodes of the Ulysses supercomputer. Each node has 16 cores divided between two sockets. The simulations were run in parallel using

both mpi and openmp. In figure(1) the results of simulations can be seen. The system Argon_2916 has a better scaling because it is bigger and therefore parallelization is more effective. In table (1) the times are shown.

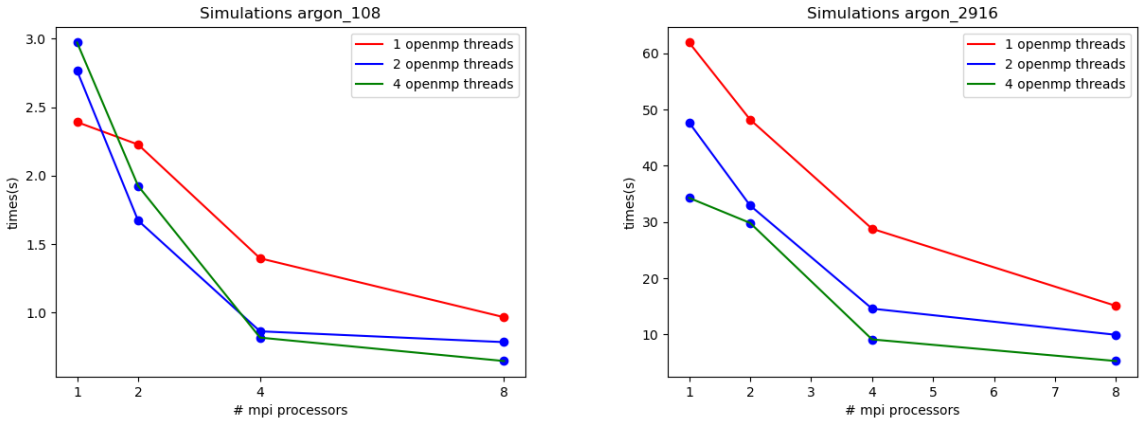

Figure 1: Scaling of the simulations varying numbers of mpi and openmp threads

Table 1: Simulations times

mpi tasks	openmp tasks	time(s) Argon_108	time(s) Argon_2916	Sp Argon_2916
1	1	2.391	61.899	1.0
2	1	2.227	48.212	1.283
4	1	1.396	28.800	2.149
8	1	0.966	15.088	4.105
1	2	2.765	47.678	1.0
2	2	1.672	32.941	1.444
4	2	0.863	14.599	3.259
8	2	0.784	9.952	4.781
1	4	2.972	34.287	1.0
2	4	1.925	29.834	1.149
4	4	0.817	9.131	3.755
8	4	0.646	5.292	6.479