MPI Assignment: N-body Simulations

Stefano Sandonà

Vrije Universiteit Amsterdam, Holland

1 The N-body problem

Given N celestial objects (bodies) with some properties (mass, initial velocity, radius, ...), the N-body problem consists on the prediction of the individual motions of these bodies. This is done by measuring the forces that they exert on each other (Coulomb gravity,...). As a result, we obtain a simulation of the behavior of the system over time.

2 The sequential algorithm

Simplify structure:

```
for each timestep do
        Compute forces between all bodies
        Compute new positions and velocities
```

As first thing, we have to discretize the time, so the application knows what to do at certain timestamps. Than for each step, the algorithm computes all the forces between all the bodies. With the resulting forces, the new velocity and positions of all the bodies are updated.

2.1 Force computation

This step of the algorithm is the most expensive and determine its complexity. The forces that bodies exert on each other has to be calculated once per pair of bodies.

```
for (b=0; b<bodyCt; ++b) {
                for (c=b+1; c<bodyCt; ++c) {
                        ...
```

That means with N bodies, we have (N-1)+(N-2)+(N-3)+...+1 pairs, so O(N^2) force computations for every timestamp.

3 Parallel N-body algorithm

An efficient parallelization of this algorithm is not trivial because for every step we need updated informations from all bodies that are part of our system to calculate the new values.

3.1 Bodies distribution

One of the major problems of the parallel algorithms is the load imbalance. If we don't distribute the work fairly between the machines involved in the computation, we'll have some machines idle while others will still work. This problem affects a lot the performance of the application because the overall execution time depends on the last machine that terminate the computation. As each body has to interact with all the others, we have the same amount of work per body, so we just have to find a way to distribute equally the N bodies among all machines. Using a simple *for* construct we can calculate the chunks of bodies to assign to each computational node (number of bodies per node and the boundaries of the chunks).

```
int sum = 0;
for (i = 0; i < numprocs; i++) {
    bodies_per_proc[i] = bodyCt / numprocs;
    if (rem > 0) {
        bodies_per_proc[i]++;
        rem--;
    }
    displs[i] = sum;
    sum += bodies_per_proc[i];
}
```

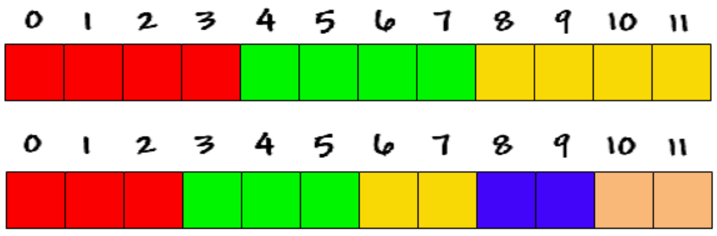

Figure 1: 12 bodies and 3 computational nodes, 12 bodies and 5 computational nodes

3.2 The Work repartition

3.2.1 Naive repartition

This is the most critical part of the implementation, from this depends the efficiency of the whole algorithm. Using a naive approach for every step we could simply use the MPI_Allgatherv operation, to send to all other nodes our updated chunk of bodies and receive all the other chunks from other nodes. Than compute the forces that all other bodies exert on the actual chunk and update the actual chunk bodies positions and speeds.

A problem of this approach is the big amount of data transferred in each iteration (N bodiesType structures) and the fact that we are computing the same forces between some pairs of bodies in more than one machine. In fact, computing the force that the body A exert on the body B, implicitly we are computing also the force that the body B exert on the body A (force of B on A is negative of A on B). Given a situation like the one described on Figure 2, we can see on Tables 1 and 2 the forces computed by the first two nodes.

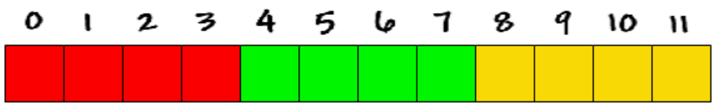

Figure 2: 12 bodies and 3 computational nodes

As we can notice from Tables 1 and 2, forces between the pairs highlighted in red are calculated by both Node 0 and Node 1. For each node we calculate the forces for 38 pairs, that means 114 pairs (38 x 3) in total. The number of possible pairs is 66, so we are doing about 1.7 times the needed work. The

Table 1: Node 0 - Calculated forces

bodies[0]	0-1	0-2	0-3	0-4	0-5	0-6	0-7	0-8	0-9	0-10	0-11
bodies[1]	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9	1-10	1-11	
bodies[2]	2-3	2-4	2-5	2-6	2-7	2-8	2-9	2-10	2-11		
bodies[3]	3-4	3-5	3-6	3-7	3-8	3-9	3-10	3-11			

Table 2: Node 1 - Calculated forces

bodies[4]	4-0	4-1	4-2	4-3	4-5	4-6	4-7	4-8	4-9	4-10	4-11
bodies[5]	5-0	5-1	5-2	5-3	5-6	5-7	5-8	5-9	5-10	5-11	
bodies[6]	6-0	6-1	6-2	6-3	6-7	6-8	6-9	6-10	6-11		
bodies[7]	7-0	7-1	7-2	7-3	7-8	7-9	7-10	7-11			

detailed computation is explained on the Equation 1).

$$\begin{aligned} Computation &= O(\lceil\frac{N}{P}\rceil * \lceil\frac{N}{P}\rceil * (P-1)) + O((\lceil\frac{N}{P}\rceil - 1)!) \\ &= O(\frac{N^2}{P}) \end{aligned} \tag{1}$$

With this approach for each iteration we send O(N/P) bodies and we receive O(N-N/P) bodies, that means an O(N) communication. For the computation we have the O(N^2/2P) for the force computation, and an O(N-N/P)for each the velocities and the positions computation.

3.2.2 Optimized repartition

The aim of a good repartition is to not repeat the same work on different nodes. Following this objective the idea is, given a pair of chuncks, half of the forces that two bodies from different chunks exert on each other are calculated by one node, and half by the other. The ripartition is clarify on the Figures 3, 4 and 5.

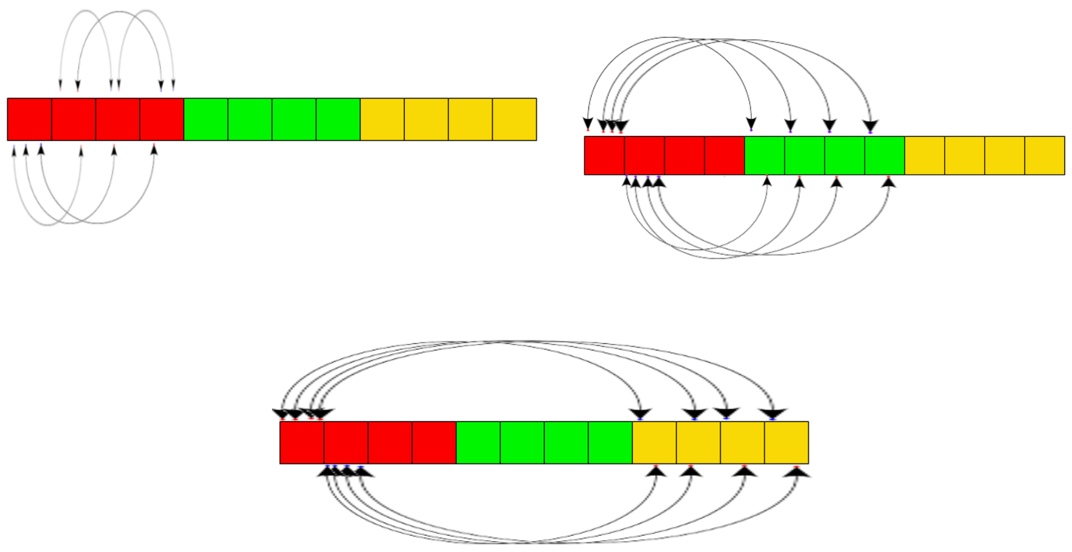

Figure 3: Forces computed by Node 0 (6+8+8=22)

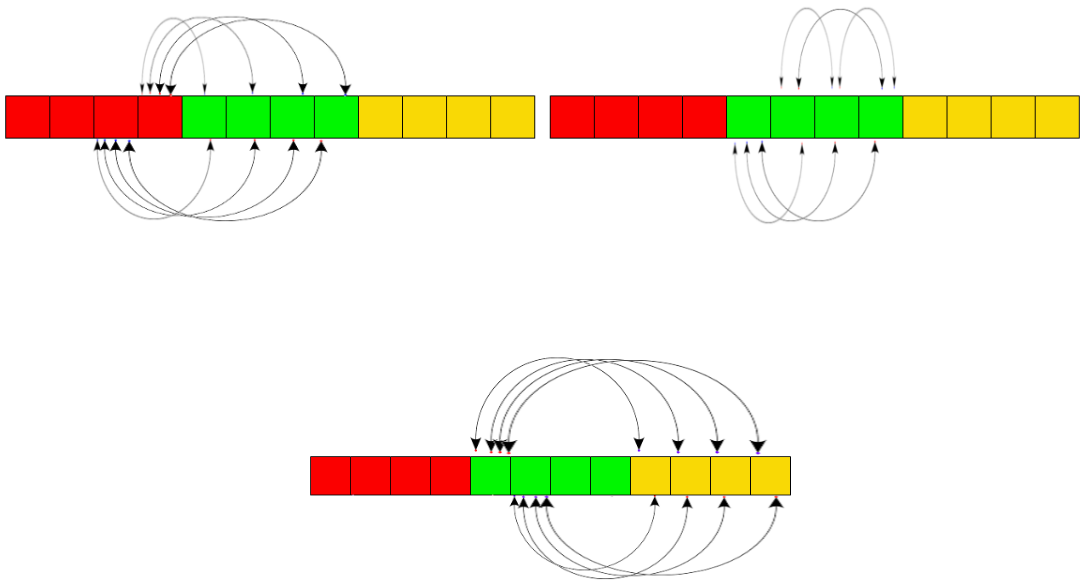

Figure 4: Forces computed by Node 1 (8+6+8=22)

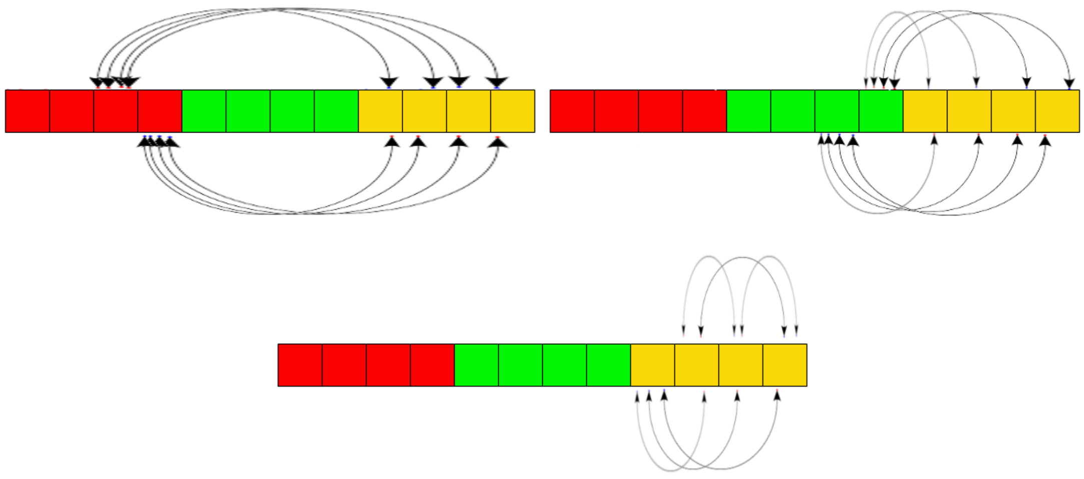

Figure 5: Forces computed by Node 2 (8+8+6=22)

Proceeding in this way, the forces between pairs of bodies are calculated only one time per step. Each node computes the forces of 22 pairs. We have 3 nodes, so 3 times 22 is 66 that correspond to the number of possible pairs. No useful work is done. The computational time is explained in the Equation 2.

$$\begin{aligned} Complexity &= O(\lceil\frac{N}{P}\rceil * (\lceil\frac{N}{P}\rceil * \frac{1}{2}) * (P-1)) + O((\lceil\frac{N}{P}\rceil - 1)!) \\ &= O(\frac{N^2}{2P}) \end{aligned} \tag{2}$$

3.3 The final solution

3.3.1 Communication VS Computation

The communication is a common bottleneck on lots of parallel implementations. The data exchange between nodes should be reduced at the minimum in order to obtain good performances. The problem with the N body algorithm is that at every step we need the updated information (positions, velocities) of all other bodies of the system in order to correctly compute the forces of the assigned chunk. That means an exchange of data for every step. To gain performances we have to find a good compromise between the data exchange and the work to do on each node. In particular, we have to maximize the ratio *computation/communication*. Due to the fact that the force computation is the most expensive part of the algorithm and that we want to exhange as less data as possible, a good compromise is the following. At the begin, we broadcast to all the nodes the bodies (MPI_Bcast). Than for each step, each node calculates the forces for the assigned pairs (the one described on Section 3.2.2), after that uses an MPI_Allreduce operation to sum all the calculated forces by all the nodes and as last thing it computes the velocities and positions for all the bodies. Proceeding i this way, we exchange only an array of N elements containing the forces calculated for the assigned pairs. The major problem of this approach, is that we calculate the positions and velocities of all the bodies in every node, that represent a duplication of work. An alternative could be to calculate in each node only the velocities and the positions of the bodies of the assigned chunk, but doing this we have to share with the other nodes also this information. The decision so, is between computing the N velocities and positions in all the nodes or exchange more data with the other nodes.

3.3.2 Approach one

Phase A (Figure 6) is executed at the begin. Phases B, C, D, E (Figures 7, 8, 9, 10) are executed inside a loop (for each step).

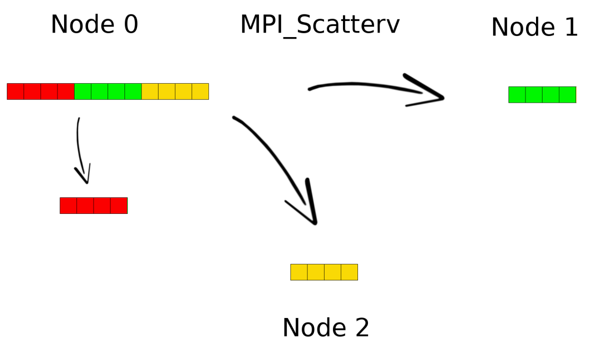

Figure 6: A - Scatter

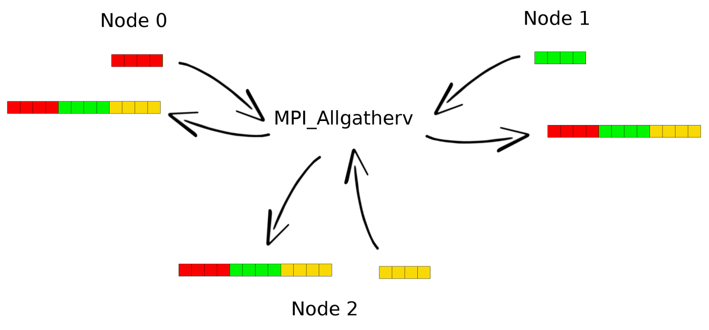

Figure 7: B - Collect updated bodies from other nodes

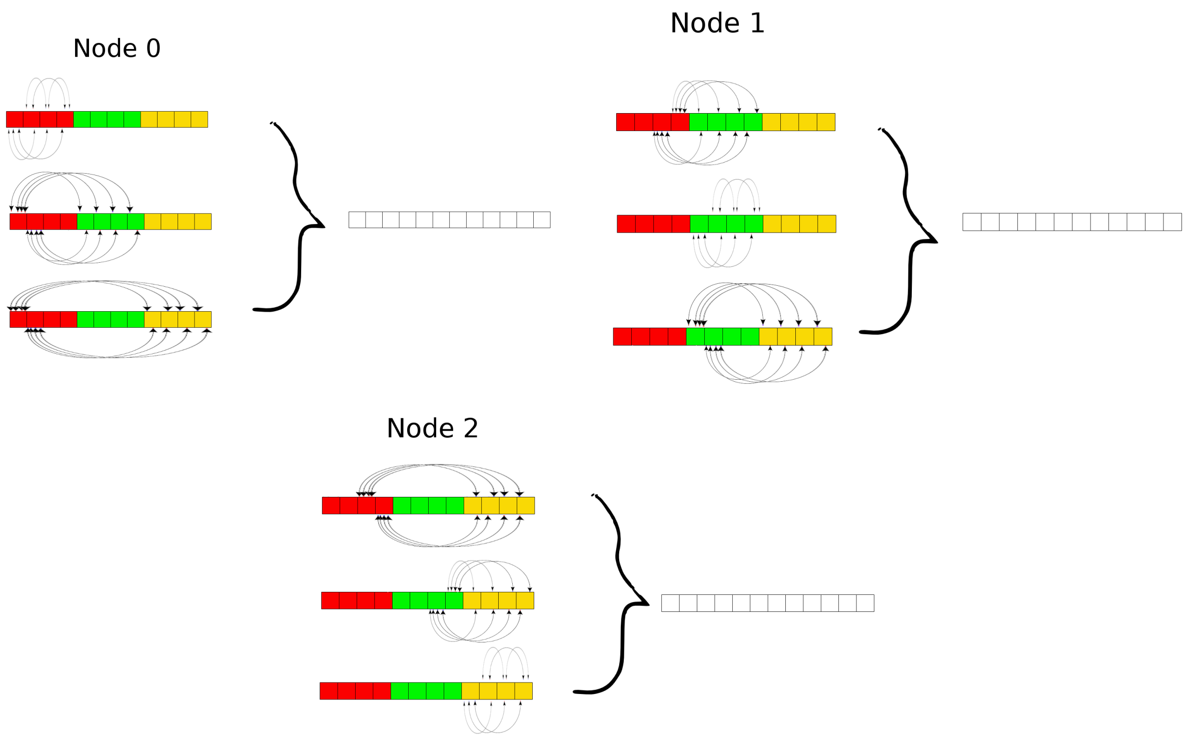

Figure 8: C - Forces computation

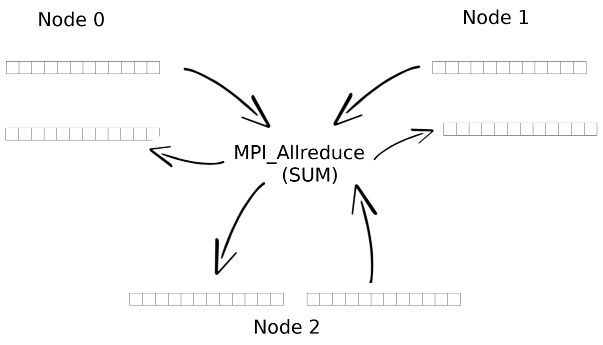

Figure 9: D - Reduce forces

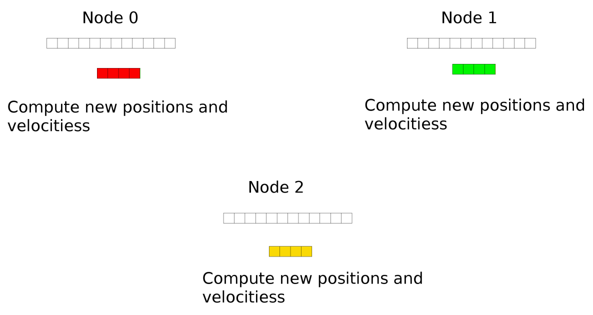
Node 0
Compute new positions and velocitiess
Node 1
Compute new positions and velocitiess
Node 2
Compute new positions and velocitiess

Figure 10: E - Update positions and velocities for bodies of the assigned chunk

3.3.3 Approach two

Phase A (Figure 11) is executed at the begin. Phases B, C, D (Figures 12, 13, 14) are executed inside a loop (for each step).

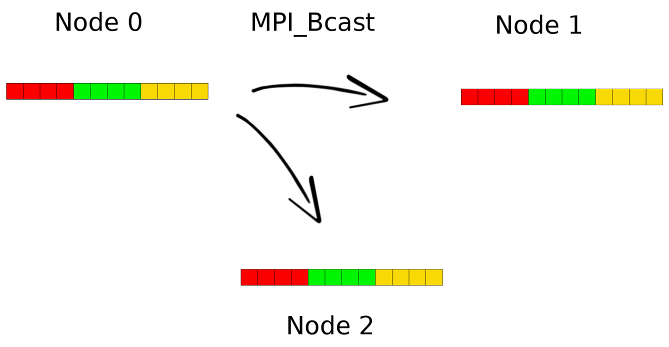

Figure 11: A - Broadcast

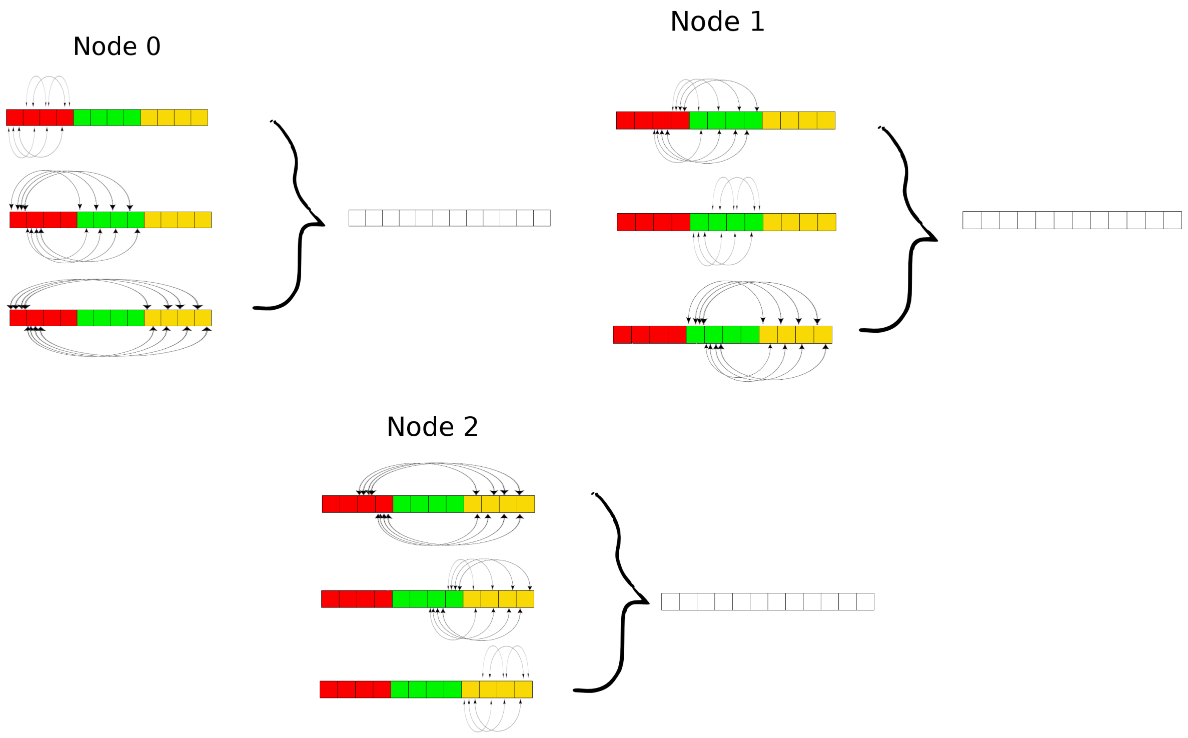

Figure 12: B - Forces computation

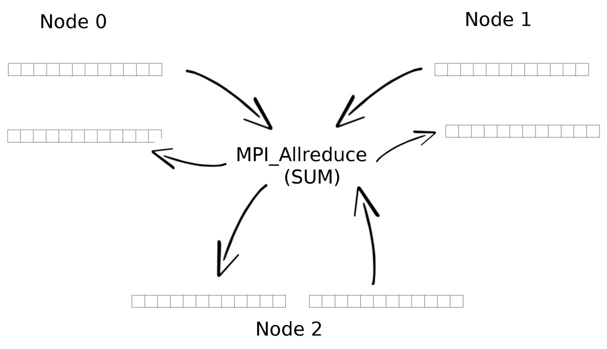

Figure 13: C- Reduce forces

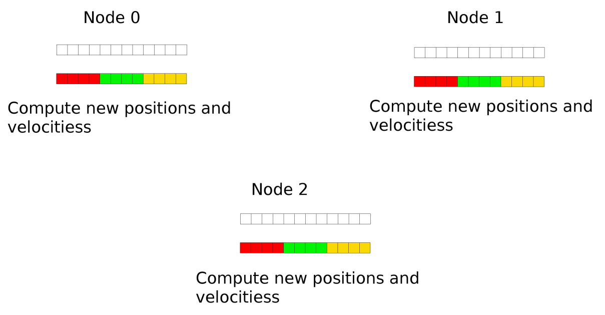
Node 0
Compute new positions and velocitiess
Node 1
Compute new positions and velocitiess
Node 2
Compute new positions and velocitiess

Figure 14: D - Update positions and velocities for all the bodies

3.3.4 Solutions comparison

After a comparison of the two solutions for different size of inputs (steps and number of bodies), it's clear that the amount of data exhanged is the key point. The Figure 15 shows just two examples of problem size with which we can see how the execution time of the first approach is lower than the execution time of the other and so how the speedups are better on the first approach. The first approach so is choosen.

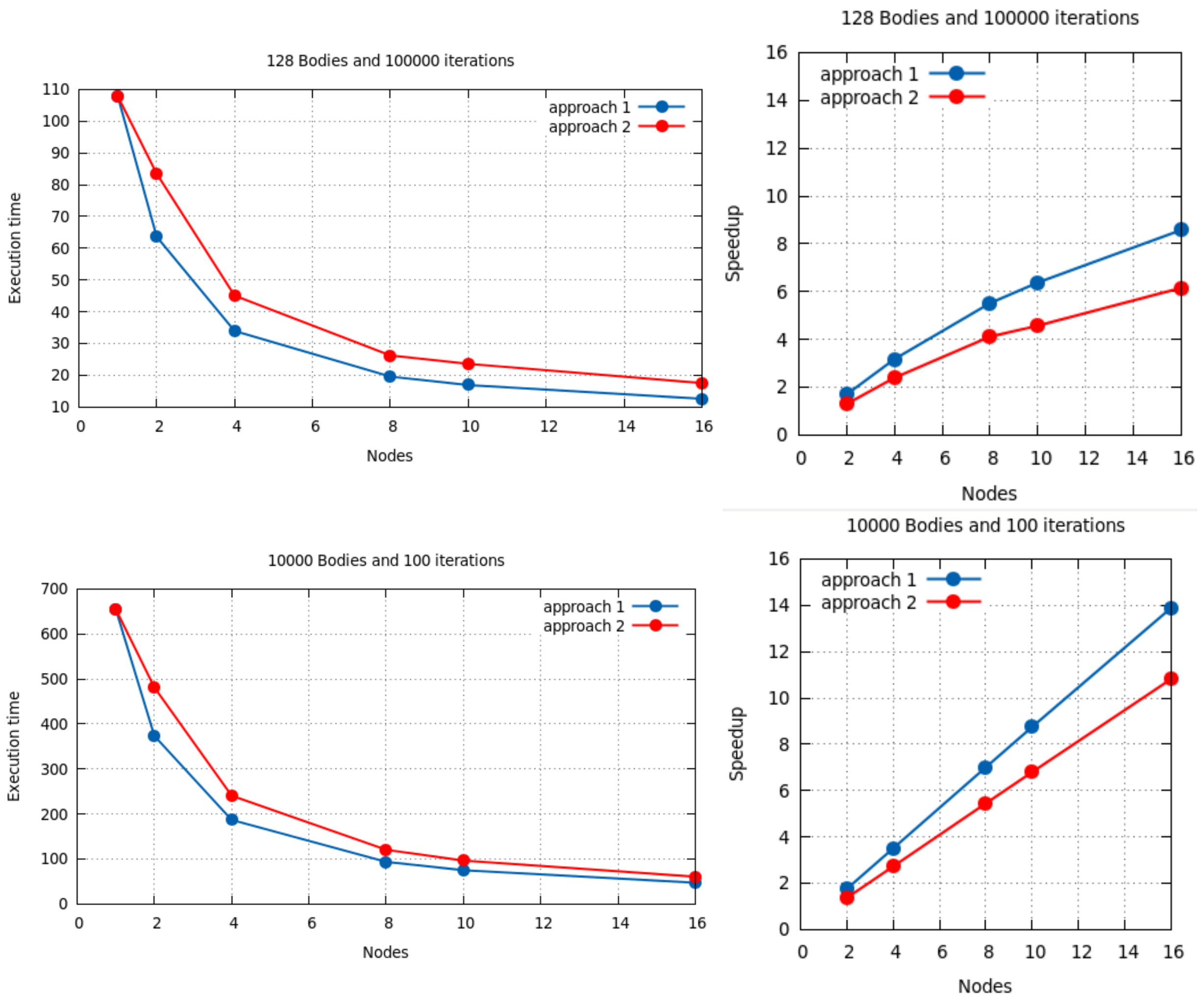

Figure 15: Execution Time and Speedups of the two approaches

3.3.5 Results

In this section all results obtained are reported. On the Figure 16 we have an example with a big number of iterations (fixed) and different numbers of bodies. On the Figure 17 we have an example with a small number of iterations (fixed)and different numbers of bodies. As we can see, more we increase the size of the problem and more the implementation obtains good speedups. On paralle programming we have to deal with big size problems, so it doesn't matter if with small problems we don't have good results.

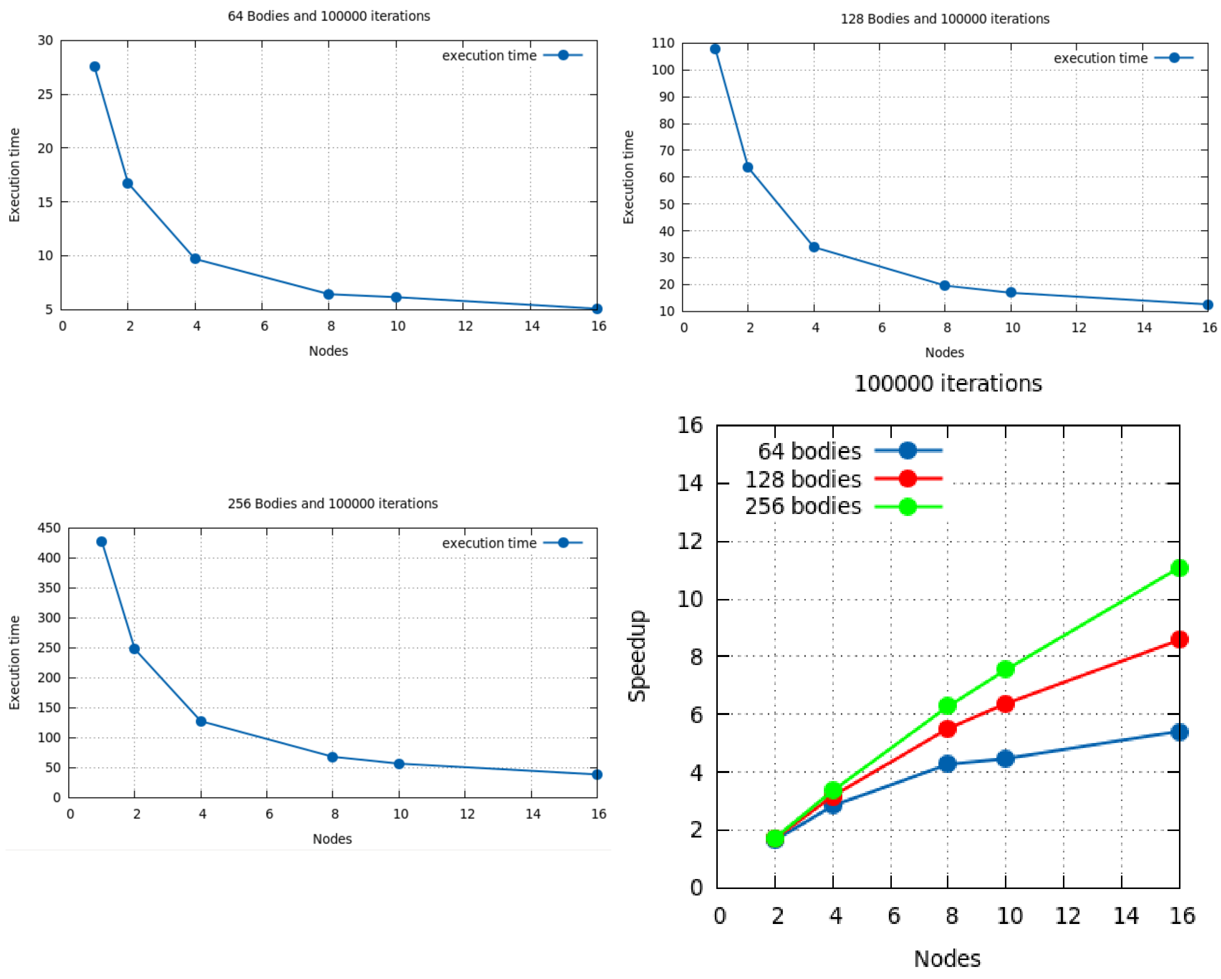

Figure 16: Execution Time and Speedups, 100000 iterations

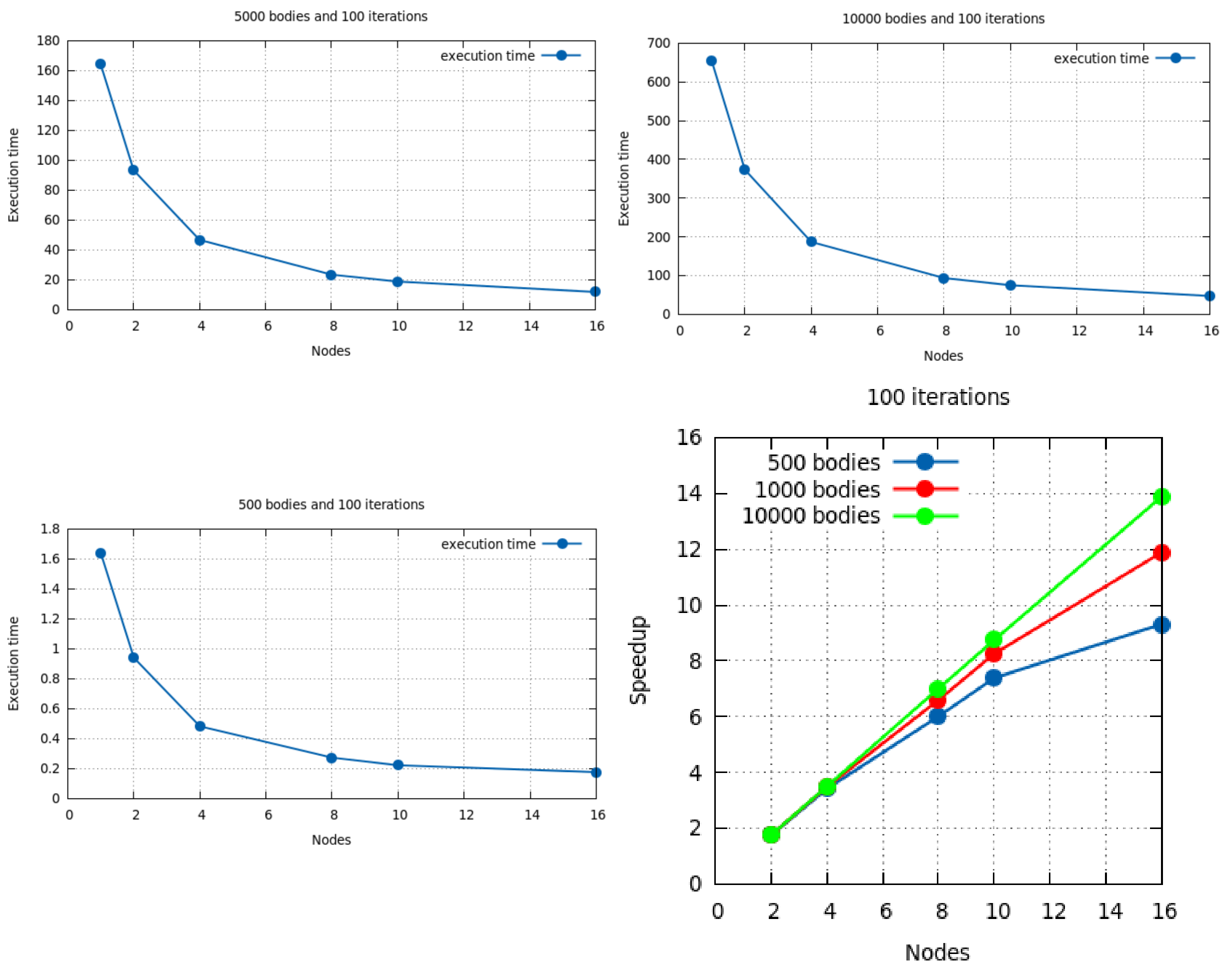

Figure 17: Execution Time and Speedups, 100 iterations

On the following tables we can find the

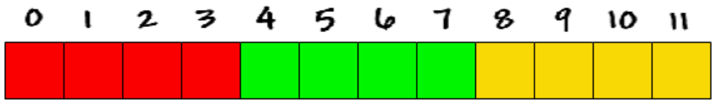

Figure 18: 12 bodies and 5 computational nodes

Table 3: 100000 iterations - Execution times

nodes\bodies	64	128	256
1	27.533	107.856	427.855
2	16.752	63.612	247.778
4	9.706	33.939	127.529
8	6.435	19.566	68.082
10	6.155	16.916	56.687
16	5.081	12.584	38.576

Table 4: 100000 iterations - Speedups

nodes\bodies	64	128	256
2	1.64	1.69	1.72
4	2.83	3.17	3.35
8	4.27	5.51	6.28
10	4.47	6.37	7.54
16	5.41	8.57	11.08

Table 5: Node 0

block0	0-1	0-2	0-3	1-2	1-3	2-3		
block1	0-4	0-5	0-6	0-7	1-4	1-5	1-6	1-7
block2	0-8	0-9	0-10	0-11	1-8	1-9	1-10	1-11

Table 6: Node 1

block0	2-4	2-5	2-6	2-7	3-4	3-5	3-6	3-7
block1	4-5	4-6	4-7	5-6	5-7	6-7		
block2	4-8	4-9	4-10	4-11	5-8	5-9	5-10	5-11

Table 7: Node 2

block0	2-8	2-9	2-10	2-11	3-8	3-9	3-10	3-11
block1	6-8	6-9	6-10	6-11	7-8	7-9	7-10	7-11
block2	8-9	8-10	8-11	9-10	9-11	10-11		

Table 8: Node 0

block0	0-1	0-2	1-2
block1	0-3	0-4	0-5
block2	0-6	0-7	
block3	0-8	0-9	
block4	0-10	0-11	

Table 9: Node 1

block0	1-3	1-4	1-5	2-3	2-4	2-5
block1	3-4	3-5	4-5			
block2	3-6	3-7				
block3	3-8	3-9				
block4	3-10	3-11				

Table 10: Node 2

block0	1-6	1-7	2-6	2-7
block1	4-6	4-7	5-6	5-7
block2	6-7			
block3	6-8	6-9		
block4	6-10	6-11		

Table 11: My caption

block0	1-8	1-9	2-8	2-9
block1	4-8	4-9	5-8	5-9
block2	7-8	7-9		
block3	8-9			
block4	8-10	8-11		

Table 12: Node 3

block0	1-10	1-11	2-10	2-11
block1	4-10	4-11	5-10	5-11
block2	7-10	7-11		
block3	9-10	9-11		
block4	10-11			