N-Body parallelization with MPI
DRAFT

Nicholas Molyneaux

November 24, 2015

1 N-Body

The n-body simulations are relevant for many fields of science, from planet and galaxy interactions all the way down to molecular dynamics. The computational challenge comes from the fact that all bodies have some effect on all the others. The effect comes from the gravitational pull each body has on the others, the force one body of mass m_1 has on another of mass m_2 depends on the distance between each body r and the gravitational constant G. The formula is the following:

$$\|\boldsymbol{F}\| = G\frac{m_1 m_2}{r^2}$$

where $G = 6.674 \times 10^{-11}\,\mathrm{Nm^2kg^{-2}}$. The direction of the force is given by the center of masses of the objects in the following way, where p_1 and p_2 are the positions of both masses.

$$F_x = \|\boldsymbol{F}\|\frac{p_{2,x} - p_{1,x}}{r}$$
$$F_y = \|\boldsymbol{F}\|\frac{p_{2,y} - p_{1,y}}{r}$$

1.1 Complexity

A classical brut-force approach to this problem has a complexity of $\boldsymbol{\mathcal{O}(n^2)}$, where n is the number of bodies in the system. This will rapidly become unbearable, since for 1000 bodies the calculations take the order of one second on a laptop. Since this problem is time dependant, the calculations must be performed at each time step, hence the importance of having a parallel application to perform the body interactions is obvious. The copmutation of the interactions at each step is straight forward, using two loops the force on each body induced by the other bodies is calculated.
One well-known solution to this problem is the Barnes-Hut algorithm. This method calculates the forces acting between one body, and a group of bodies represented by the average of the group's masses and positions. With this algorithm, the complexity is reduced to $\boldsymbol{\mathcal{O}(nlogn)}$. In this case, the world (i.e. collection of bodies) is stored in a structure called "quad-tree", where each node stores the average of the bodie's positions and masses. At the leaves there is either a body if it fits into the quadrant, or nothing if no body is present. To calculate the interactions between all bodies, we traverse all bodies, and based on a distance criteria we either calculate the interaction with the exact body or the interaction with an "averaged body".

1.2 Computations VS Communications

For a simple brut-force approach, the computations follow $\mathcal{O}(n^2)$ at each time step. To guarantee correct results, the acceleration, velocity and position of all bodies must be updated in a synchronized manner, hence one cannot move forward in time until all forces have been calculated. Once the new positions are calculated for all bodies on each node, these updated positions must be broadcast to all other nodes so they can continue the compuations at the next time step. Therefore the communication complexity is $\mathcal{O}(n)$, since the positions of the bodies must be passed around between nodes. The ratio of computations

to communications can be approximated as $\mathcal{O}(n)$ for the parallel brute-force approach.
For Barnes-Hut's algorithm, this ratio is harder to estimate. Since the application is no longer automatically load-balanced, the communications will depend on the tree and how the graph is updated at each time step. Since for distant groups we consider only averages, exchanging all positions at each time step is not necessary and expensive in communications. To give a precise estimation, the method for load balancing the work on each worker must be defined.

1.3 Amdahl's Law

Brut-Force approach For the brut force approach, since the data is stored in arrays, very little preprocessing is required. The fraction of non-parallelizable code is very low, it is only loading the data from a file, and writing the positions to a file. Some quick measurements give a fraction $f = 0.005$ as the serial part. Hence the optimistic upper bound for the speed can be calculated and plotted. For the improved estimation, the communication fraction can be estimated to $f_{comm} = 0.015$, which takes into account the time to distribute the inital situation to the nodes. The data which must be distributed is the mass, initial velocities and positions to all nodes. Hence the new "realistic" speedup graph is shown below, Figure 1. This speedup graph is still far too optimistic as it does not take into account the communication at each time step.

Quad-Tree approach For a quad-tree method, since building the inital tree will take a long time, this should be performed by the main processor, then sent to each worker. Hence the initial communication will contain the whole tree and might be more demanding then the brut-force case. With this regard, the optmistic upperbound will not be as high as previously. On the other hand, the computations are much faster since they are in $\mathcal{O}(nlogn)$. Figure 1 shows a first estimate of the theoretical maximum speedup, based on the estimation that building the quad-tree will take some time.

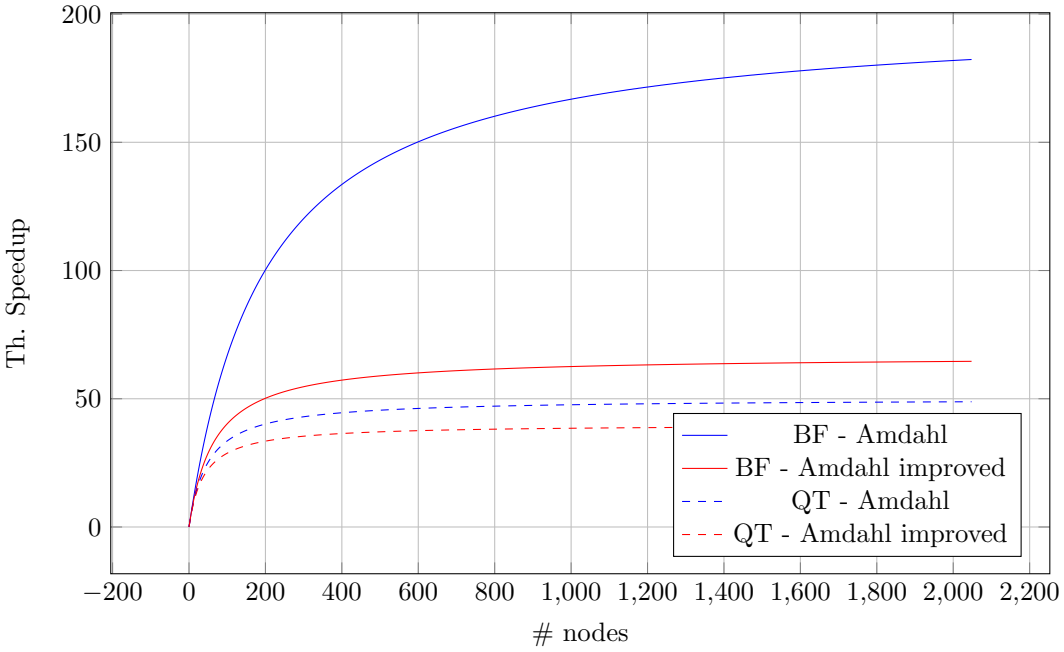


Figure 1: Theoretical speedup chart for both the brut-force approach and the quad-tree method. These are estimations based on some simple timings on a serial implementation.

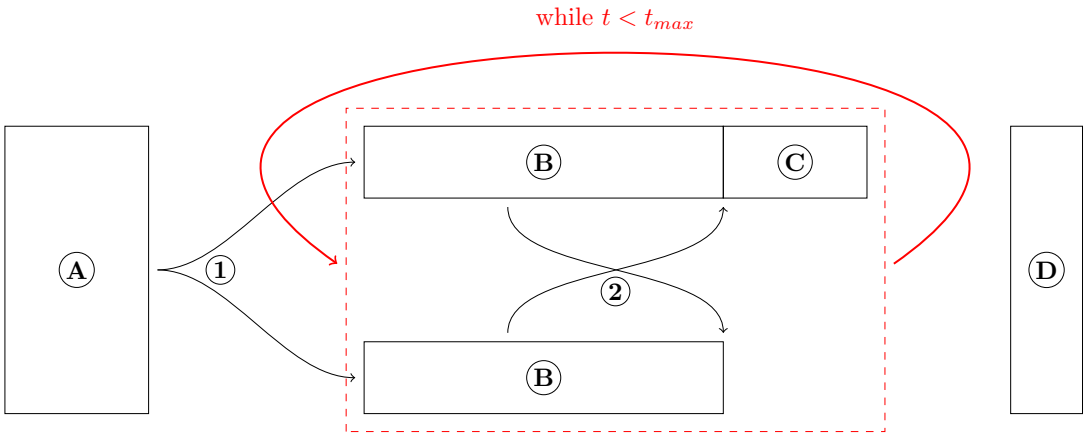
2 Parallelization strategy

The choice for MPI was justified by the need for industry and the universal aspect of the message passing interface paradigm. For the n-body problem, the messages to be passed are the mainly the

updated positions of the bodies which were calculated on the other nodes. An efficient way of storing the data must be conceived to minimize the transfer times and allow the Barnes-Hut algorithm to run. As mentionned previously, two strategies are implemented. A brute-force approach and the Barnes-Hut algorithm. The strategy for the first is clear whereas for the second further work needs doing.

2.1 Brut-Force flowgraph

With this methodology the only data which is exchanged is the updated positions. For this brut-force approach, this data must be updated at each time step, hence less data cannot be transfered. The output of the positions at each time step could be done in parallel using MPI's write methods, but at this stage this is not done.



(A) - Load data, splits the data
(B) - Computes the displacement for bodies on node
(C) - Main node writes the data to file
(D) - Finalization
(1) - Sends the initial data to all nodes using MPI_Bcast and MPI_Scatter
(2) - Sends updated positions to all nodes using MPI_Allgather

Figure 2: Flowchart for the brut-force parallel implementation. Once the initial data is sent to all nodes, they update the positions of all bodies which are assigned to them, then send to all of the other nodes the updates positions. At the end of a time step, the positions are synchronized and a new time step can be calculated. The main node will write the positions to a file at each time iteration.

2.2 Brut-Force timing diagram

In this case, the slowest path is straightforward to estimate. The assumption that the intial distribution of the data between the nodes is done sequentially. Once the data is distributed and the first time step calculated, the idle time of each node is only the time it takes for the master node to write the data to a file. This step could also be parallelized by either using an "MPI_Write", or getting each node to write a file locally and then regrouping all the data to the main node.

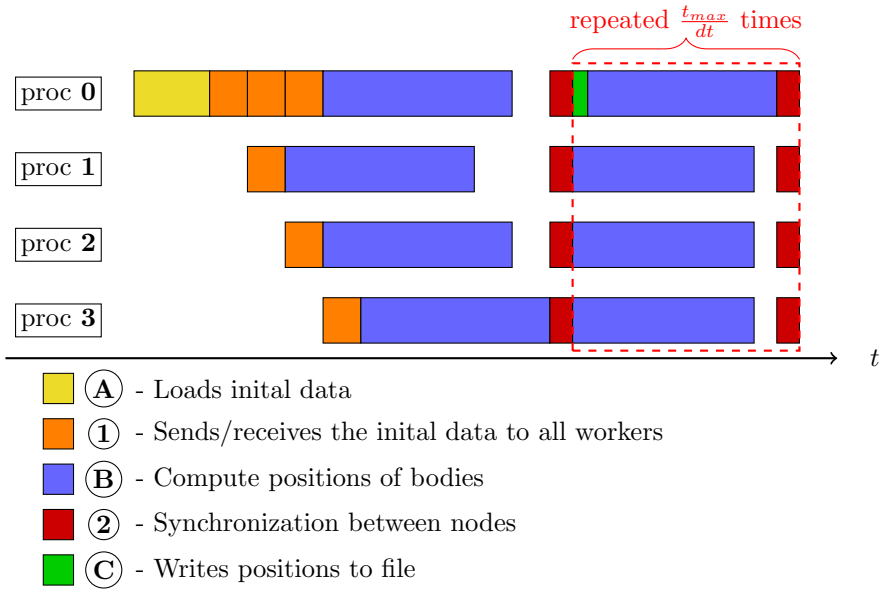


Figure 3: Timing diagram for the BF approach. The block of computations is repeated until the final time is reached. The circled characters next to the coloured boxes refer to Figure 2, to emphasize that they are the same steps.

2.3 Quad-Tree

The flowgraph and timing diagram will be very similar. The main differences will reside in the initialization processes which requires to build the full tree and the type and amout of data exchange between each node.

3 Performance

The perormances are expecetd to be far lower than the theoretical ones calculated previously since the communication times are neglected in the actual estimations.