PROJECT PROPOSAL PHPC-2016

A High Performance Implementation of the 2D nBody Gravitational problem: Benchmark of the Barnes-Hut algorithm compared to the Brute-Force algorithm.

Principal investigator (PI)	Gael Lederrey
Institution	EPFL-CSE
Address	/
Involved researchers	Only PI
Date of submission	May 15, 2016
Expected end of project	June 15, 2016
Target machine	Deneb
Proposed acronym	NBODY-BHVSBF

Abstract

The nBody problem has been well studied by many scientists in many different fields such as atoms simulation or astrophysics. In this project, we present the implementation of the Barnes-Hut algorithm. A distributed memory version using the MPI library will also be presented. A benchmark comparison is done with the Brute-Force algorithm, serial and parallel versions. The whole project can be found at this url: https://github.com/glederrey/nBody-PHPC-2016-EPFL

1 Scientific Background

1.1 Algorithms

The nBody problem covers the whole scale of science from the subatomic particles to the gigantic galaxies. It is a problem that has been studied a lot and new papers are still being published about this problem nowadays. In order to solve this problem, it exists a really simple algorithm called Brute-Force. It consists in looping on all the bodies and inside this loop, it loops again on all the bodies:

Algorithm 1 Brute-Force

```
1: n = Number of bodies
2: Initialize n bodies
3: while t < t_end do
4:     for i = 1 to n do
5:         for j = 1 to n do
6:             Apply forces from body j on body i
7:     for k = 1 to n do
8:         Update the body k
```

We can clearly see that the loop to calculate all the forces is in $\mathcal{O}(n^2)$. Updating the bodies will take only $\mathcal{O}(n)$ time. But most of the time will be spent in the calculation of the forces. For a very large number of bodies this algorithm will be very slow. Therefore another algorithm can be used: Barnes-Hut.
The Barnes-Hut algorithm is easy to understand. First, it consists in building a quadtree and place each body in the leaves of this tree. Each parent node will store the center of mass of its leaves. Then, we can apply the forces for each body. Instead of calculating all the forces between the current body and all the other bodies, it uses the center of the mass of multiple bodies if there are far away from the current body. The pseudo-code is given below:

Algorithm 2 Barnes-Hut

```
1: n = Number of bodies
2: Initialize n bodies
3: while t < t_end do
4:     for i = 1 to n do
5:         Add body i in the tree
6:     for j = 1 to n do
7:         Calculate all the forces applied to body j.
8:     for k = 1 to n do
9:         Update the body k
```

Adding a body in the tree takes $\mathcal{O}(\log n)$ time (height of the tree). Therefore, adding all the bodies will take $\mathcal{O}(n \log n)$ time. The calculation of the forces will take $\mathcal{O}(n) \times \mathcal{O}(\log n) = \mathcal{O}(n \log n)$ time. And finally, updating the bodies will take $\mathcal{O}(n)$ time. Therefore, this algorithm will take $\mathcal{O}(n \log n)$ time which is much faster than the Brute-Force algorithm for a high number of bodies.

For both algorithms, the communication between the process is proportional to the number of bodies. Therefore, we can say that communication is in $\mathcal{O}(n)$ time.

1.2 Gravitation

In order to take a real test case for the nBody problem, we choose the Gravitational nBody problems. The forces are given by:

$$\vec{F}_{1\to 2} = G \cdot \frac{m_1 m_2 (\vec{x}_2 - \vec{x}_1)}{\|\vec{x}_2 - \vec{x}_1\|^3} \tag{1}$$

where G is the gravitational constant and is equal to $6.674 \cdot 10^{-11}$ $[Nm^2kg^{-2}]$. As we can see with this equation, the forces decreases proportionally to $1/r^2$, r being the distance between two bodies. This allows the center of the mass approximation in the Barnes-Hut algorithm. Indeed, if two bodies are close to each other and far away from a third body. The two forces of these bodies on the third one will be almost the same as the force of the center of the mass of the two far bodies.

1.3 Too close and too far?

Two problems occur with the nBody problem:

- What do we do when a body is moving too far form the other bodies?

- What do we do when two bodies are really close?

We decided to address these two problems in a very simple manner. First, we define a maximum and minimum distance. If a body is outside of the square with maximum distance as the edge length, then we delete it from the list of bodies. And if two bodies have a distance smaller than the minimum distance, we do a perfect inelastic collision between them.

2 Project Description

In this project we want to implement a high performance version of the Barnes-Hut algorithm in C++ [3]. A simple version of the Brute-Force algorithm will also be implemented in order to be used as a benchmark for the Barnes-Hut algorithm. At the end, four different codes will be produced:

- Serial version of the Brute-Force algorithm
- Distributed memory version, using MPI [4], of the Brute-Force algorithm
- Serial version of the Barnes-Hut algorithm
- Distributed memory version, using MPI [4], of the Barnes-Hut algorithm

The three first codes will be used as benchmark for the distributed memory version of the Barnes-Hut algorithm. Indeed, to prove that this version is the fastest, we need to make sure it overcomes the Brute-Force algorithm.

For the two distributed memory codes, we determine the Amdahl's law. We will then compare the strong scaling and the weak scaling. For the distributed memory version of the Barnes-Hut algorithm, we will study the effect on the Load-Balancing on the processors.

3 Implementations

These four applications are implemented in C++ [3]. As it was said in the previous section, the distributed memory versions are using the MPI library [4]. Everything has been compiled using `gcc` version 4.8.4. The code have been debugged using the general debugger `gdb`. And `valgrind` has been used to remove all the memory leaks.

3.1 Optimization

For each of the codes, we compile it once without using any optimization flags and once with the following two flags: `-Ofast` and `-ftree-vectorize`. The non-optimized version will not be used. It was just for testing purpose.

3.1.1 Brute-Force

Not much efforts were put onto the optimization of the Brute-Force algorithm since it exists only for a benchmark purpose. For the MPI version of the code, the first (outer) loop is divided into the number of processes. The second (inner) loop is still on the n bodies. After the update of the local bodies, everything is sent back to the master process to update the vector of all the bodies. It also redistribute equally the number of local bodies to all the processes to keep a good load-balancing.

3.1.2 Barnes-Hut

For the Barnes-Hut algorithm, loop reordering and vectorization is not useful since it only uses pointers. The following is only a supposition since the code has not been written yet. We will try to do the construction of the tree in parallel on all the processes. We will also try to make the calculation of the forces parallel. After the update of all the bodies, everything will be send to the master process in order to recreate the vector of all the bodies. We will also try to find a metric to tell when it is good to recreate the quadtree. Indeed, the creation of the tree can take some time, so it is good if it does not have to be recreated at each iteration.

4 Algorithms' time

In this section, we want to prove that the Brute-Force algorithm is in $\mathcal{O}(n^2)$ and that the Barnes-Hut algorithm is in $\mathcal{O}(n \log n)$. In the figure 1, the times taken by the two algorithms are shown for different number of bodies. Different values for the precision parameter θ were used. We can clearly see that the Brute-Force algorithm has the same slope as the red line for $\mathcal{O}(n^2)$. We can also see that the Barnes-Hut algorithm with $\theta = 0$ has a the same slope. This is normal since it doesn't aggregate the far bodies. But if $\theta > 0$, we can see that the slope becomes the same as the red dotted line for $\mathcal{O}(n \log n)$, especially with a high number of bodies. Therefore, the theoretical times for the two algorithms are validated.

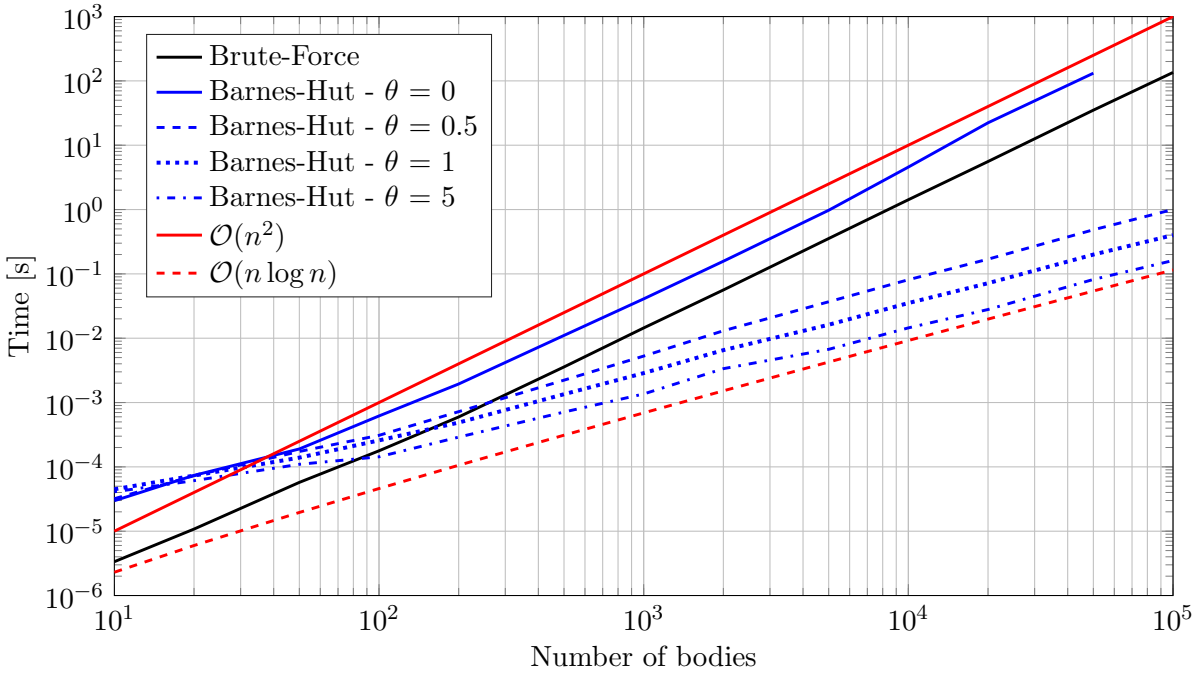

Figure 1: Time of the main iteration for the Brute-Force and the Barnes-Hut algorithms. Different values for the precision, θ, have been used. The theoretical times are given in red.

An interesting fact is to see that for a small number of bodies, the Brute-Force algorithm is faster than the Barnes-Hut algorithm. This may be due to the optimization made by the compiler and the implementation of the Barnes-Hut algorithm. However, with 10^5 bodies, we see that the Barnes-Hut algorithm, $\theta = 1$, is around 300 times faster than the Brute-Force algorithm.

5 Amdahl's law

The Amdahl's law is given by:

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}} \tag{2}$$

where S_p is the theoretical speedup, $1 - \alpha$ is the parallelizable part of the code and p is the number of processors. Ideally, we should have $S_p = p$. But this does not happen in reality.

5.1 Brute-Force

In the Brute-Force algorithm, everything can be parallelized except the loading of the data at the beginning and the communication between the process. Since this algorithm is in $\mathcal{O}(n^2)$, the loading time at the beginning will be negligible compared to the time taken for the calculation of the forces. Therefore, the speedup will be really close to the perfect speedup, *i.e.* $S_p = p$.

5.2 Barnes-Hut

In the Barnes-Hut algorithm, we can also parallelize everything except the loading of the data and the communication. But since this algorithm is in $\mathcal{O}(n \log n)$, the fraction of serial code will be much higher compared to the Brute-Force algorithm. Therefore, we can except a smaller speedup.

5.3 Results

Table 1 presents different times taken by the code for the Brute-Force and the Barnes-Hut algorithms. t_L corresponds to the loading time, $\bar{t}_{it}$ is the mean iteration time and $\bar{t}_{com}$ is the mean communication time. The total time taken by the algorithm will be:

$$t_{tot,q} = t_L + q \cdot (\bar{t}_{it} + \bar{t}_{com}) \tag{3}$$

where q is the number of iterations. Therefore, we can calculate the fraction of serial code for a given number of iterations:

$$\alpha_q = \frac{t_L + q \cdot \bar{t}_{com}}{t_{tot,q}} \tag{4}$$

The fraction α_q for $q = 100$ and $q = 1000$ is given in the table 1. For the moment, the communication time is not taken into account because the MPI code for the Barnes-Hut algorithm has not been done. Therefore, we can only take into account the loading time. This table will be done again and the Amdahl's law will be corrected after the implementation of the MPI for the Barnes-Hut algorithm.
Finally, with this fraction α_q, we can plot the Amdahl's law as it is given in the figure 2. We can clearly see that the theoretical speedup for the Barnes-Hut algorithm is much lower than for the Brute-Force algorithm.

Algorithm	n	t_L $[s]$	$\bar{t}_{it}$ $[s]$	$\bar{t}_{com}$ $[s]$	α_{100}	α_{1000}
Brute-Force	10^4	0.0588	1.300	/	$4.52 \cdot 10^{-4}$	$4.52 \cdot 10^{-5}$
	10^5	0.554	130.1	/	$4.26 \cdot 10^{-5}$	$4.26 \cdot 10^{-6}$
	10^6	4.709	13234.7	/	$3.56 \cdot 10^{-6}$	$3.56 \cdot 10^{-7}$
Barnes-Hut	10^4	0.121	0.0361	/	$3.34 \cdot 10^{-2}$	$3.34 \cdot 10^{-3}$
	10^5	0.498	0.539	/	$9.23 \cdot 10^{-3}$	$9.23 \cdot 10^{-4}$
	10^6	4.730	5.921	/	$7.99 \cdot 10^{-3}$	$7.99 \cdot 10^{-4}$

Table 1: Times used to calculate the serial part of the code for the Amdahl's law.

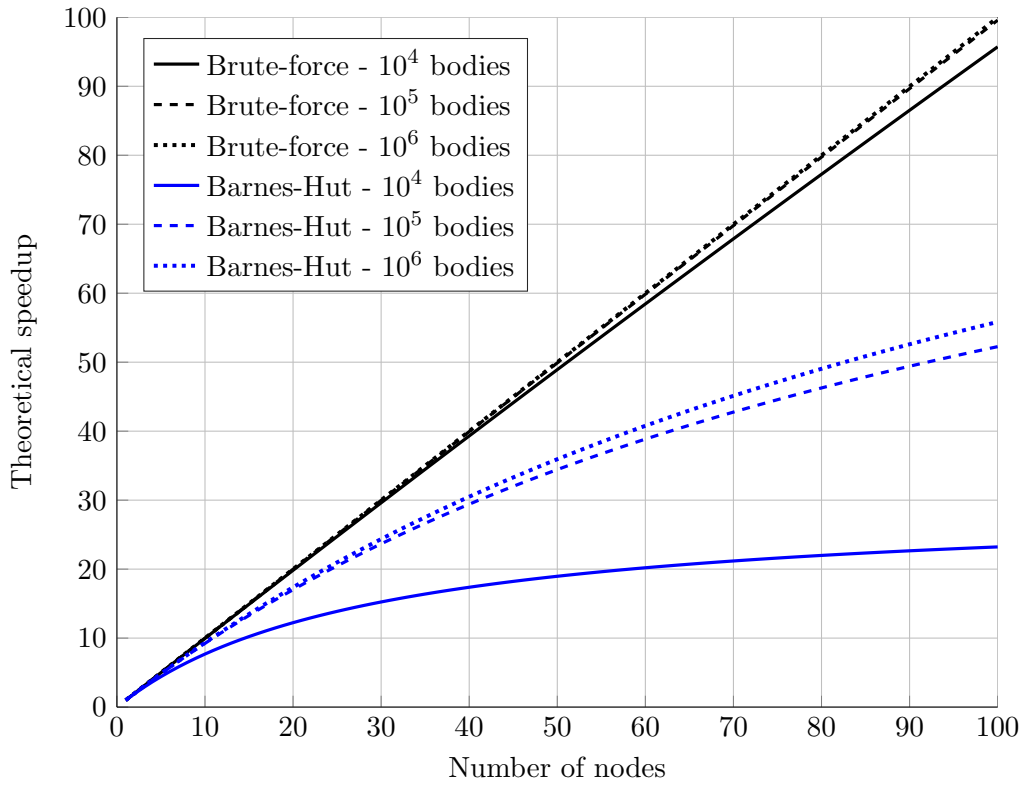

Figure 2: Result of the Amdahl's law prediction of the theoretical speedup for the Brute-Force and the Barnes-Hut algorithm. This graph is computed with α_{100}.

References

[1] Berkley - CS267: Lecture 24, Apr 11 1996: Fast Hierarchical Methods for the N-body Problem, Part 1

[2] Wikipédia - Newton's law of universal gravitation

[3] Stroustrup B.,*Programming – Principles and Practice Using C++*, Addison-Wesley, May 2014

[4] The MPI Forum, *MPI: A Message-Passing Interface Standard*, Technical Report, 1994