N-Body Gravitational Simulation - Theoretical Analysis

HERNANDEZ Sergio Daniel
Parallel and high-performance computing, EPFL Lausanne, Switzerland
May 14, 2018

***Abstract*—The aim of N-Body gravitational simulations is to mimic the dynamics and evolution of the large scale structures we observe on the universe. These systems typically contain N bodies where N can be up to hundreds of millions or billions. The complexity of simulating the dynamics of these systems scales rapidly ($O(N^2)$) and presents a challenge for high performance scientific computing. We present in this work the gain in performance for an N-Body gravitational simulation implemented with a Tree algorithm ($O(Nlog(N))$) as well as the analysis of the gain in performance with parallelism in this problem. The latest stable version from the C4science repository [1].**

I. INTRODUCTION - CONTEXT

Michele Trenti and Piet Hut (2008) [3] built a complete introduction and description of the Gravitational N-Body Simulation topic. The dynamics of the Gravitational N-Body System is described by the equation 1 where the force affecting every particle have contributions from all the other particles.

$$\vec{F_i} = -\sum_{i \neq j}^{n} G \frac{m_i m_j (\vec{r_i} - \vec{r_j})}{|\vec{r_i} - \vec{r_j}|^3} - \vec{\nabla} \cdot \phi_{ext}(\vec{r_i}) \tag{1}$$

with $G = 6.67300 \cdot 10^{-11}$ the universal gravitational constant, ϕ_{ext} a possible external potential.
The dynamic evolution of each particle is then described by Newton's equation 2:

$$m_i \frac{\partial^2 \vec{r_i}}{\partial t^2} = \vec{F_i} \tag{2}$$

Equations 1 and 2 describe entirely the N-Body gravitational dynamics. The complexity of the problem goes with $O(N^2)$ as we need to go through all the particles to calculate the net force of one particle. The force calculation is therefore going to be the main aspect to optimize in this work.

II. TEMPORAL EVOLUTION

We use a Leap-Frog symplectic integrator (also Verlet integrator) with:

1) Advance of 1/2 Δt for the velocities (kick)
2) Advance of 1 Δt for the positions (drift)
3) Advance of 1/2 Δt for the velocities (kick)

It is of special interest for us since it conserves the global Hamiltonian (slightly modified) of the system [2]. The temporal resolution can be increased with diminishing Δt. Nevertheless this comes with the cost of more timesteps (though force calculations) for the same simulation time.

[1]https://c4science.ch/source/phpc_SDHC_nbody/

III. NAIVE ALGORITHM - DIRECT SUM $O(N^2)$

We first approach the N-Body simulation through the particle-to-particle direct method, the naive brute force algorithm of the direct sum. The force calculation is described by:

$$\vec{F_i} = -\sum_{i \neq j}^{n} \frac{G m_i m_j (\vec{r_i} - \vec{r_j})}{(|\vec{r_i} - \vec{r_j}|^2 + \varepsilon^2)^{3/2}} \tag{3}$$

with ε is a smoothing parameter to avoid the force to diverge when two bodies are very close. It also defines the minimum time for the simulation as a dynamic time of a sphere of radius ε. This calculation is exact (it follows exactly the physics) but requires much computation ($O(N^2)$).

We implemented this brute force solution to have a baseline of the improvements we build on top of it.

IV. BARNES-HUT TREE ALGORITHM $O(Nlog(N))$

The Barnes-Hut Tree algorithm is based on the idea of approximating groups of particles by their respective center of mass particle [1]. This is done through the construction of a tree structure where all the particles will be contain into nodes of the tree.

A. Tree Construction

The construction of the tree is done recursively, where at each level, the algorithm perform the following steps:

1) Divide the available space into nodes, 4 equal squares (for the 2-Dimensional version) or 8 equal cubes (for the 3-Dimensional version).
2) If a nodes contains only one particle it is considered a leaf and the tree construction algorithm stops.
3) If there are more than one particle in the nodes, a new depth level is necessary and the tree construction algorithm continues.

The algorithm stops when each particle is contained in a nodes of the tree. We therefore have a tree composed of nodes with different levels of depth. Each final node contains one particle or zero particle.

B. Force Calculation

An important step before calculating the approximation of the force is to compute the center of mass and the total mass of each node. This information will be crucial for force computation step, the core of the algorithm.

- Center of mass and total mass

For each sub-division of the tree, we compute the coordinates of the center of mass and the total mass of this subdivision. We do this at each level of depth, until arriving to the leafs of the tree (the subdivision which have only one particle). This can be done recursively and therefore has a complexity of $O(N)$.

- Force Computation

We introduce the following ratio:

$$\frac{D}{r} \tag{4}$$

with D the size of the node (the square in 2-D or the cube in 3-D) and r the distance of the particle with the center of mass of the node.

By comparing the ratio 4 with a user-specified ϑ threshold (usually around 1) comes the criterion if, instead of computing the force by taking each particle individually we use the center of mass and the total mass instead. This radically reduces the number of force calculations to perform.

```
if $\frac{D}{r} < \vartheta$ then
    $\vec{F_i} = -\frac{G m_i m_{cm} (\vec{r_i} - \vec{r_{cm}})}{(|\vec{r_i} - \vec{r_{cm}}|^2)^{3/2}}$
else                                                                  (5)
    $\vec{F_i} = -\frac{G m_i m_j (\vec{r_i} - \vec{r_j})}{(|\vec{r_i} - \vec{r_j}|^2 + \varepsilon^2)^{3/2}}$
end
```

We can see that as we decrease ϑ we approach the brute-force $O(N^2)$ algorithm and as we increase ϑ we decrease the number of particles to take into account in a force calculation, thus increasing the performance of the algorithm (bounded by $O(Nlog(N))$) but at the cost of diminishing the precision of the approximation.

V. Parallelism and Optimization Strategy

It is clear that before introducing parallelism into our implementation we need to be sure that our sequential implementation is as optimized as it can be.

A. Preliminary Profiling

As of *May 14, 2018* only the brute-force implementation has been 100% completed. It carries an additional Energy Calculation function to have to possibility to compute how much of the energy is conserved.

Time [%]	Function
≈ 100	forceCalculation
0.05	energyCalculation
0.02	kick
0.02	drift
< 0.01	createVector
< 0.01	saveState (I/O)

TABLE I: Profiling of a 1000 bodies, 5000 steps ($\varepsilon = 0.1$) simulation with the direct sum algorithm ($O(N^2)$). The simulation took $41.24s$ to run on a 8 core Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz with 16GB of RAM memory.

In the profiling summarized in table I we observe how indeed the almost all of the computation time goes is in the *forceCalculation* function. This is the main argument to implement the Barnes-Hut approximation for this problem.

B. Parallelism

The N-Body problem is one of the problems which is considered to be *embarrassingly* parallel. So, theoretically by introducing parallel threads T to our problem we would expect decrease of our simulation time directly proportional with the number of threads, so for our $O(N^2)$ version we expect to get to $O(\frac{N^2}{T})$ and for our $O(Nlog(N))$ version to $O(\frac{Nlog(N)}{T})$.

- Direct Sum $O(N^2)$

For the naive particle to particle implementation, the force calculation is the main bottleneck. A shared memory parallelism is intuitive here, since every particle needs the information of all the other particles. OpenMP is definitively a good and quick choice. Nevertheless we focus our effort in implementing parallelism in our $O(Nlog(N))$ version.

- Barnes-Hut $O(Nlog(N))$

We identified to options to implement parallelism in our algorithm. For the moment the improvements thought are shared-memory implementation (using more OpenMP than MPI). Nevertheless, it is expected to increase our thread number with distributed memory parallelism through MPI. Distributed memory parallelism is more complicated to implement.

1) Space Partition
   In order to avoid having idle threads we need to think in a heuristic algorithm to subdivide the space depending on the particle density until achieving as many subdivisions as we have threads available. The Gordon Bell Prize at Supercomputing 92 was won with this algorithm [4].
2) Tree Partition
   After constructing the tree for the first time, an estimation of the workload of each subdivision is to be chosen. In a shared memory context (using OpenMP) each thread takes care of a part of the tree with equivalent workloads. In the distributed memory (Using MPI + OpenMP) context another structure is necessary to minimize the communications between the nodes. This would be a hashed tree. Warren et al. first used this kind of structure in a large scale astrophysical simulation in 1993 [5]. Each direction of subdivision of the tree is hashed to a table through a hash function (each subdivision obtains a unique identifier in the table). The table is communicated between the nodes and therefore each core can then access directly the subdivision of the tree without excess node-to-node communication.

VI. Current Conclusions and Steps to Follow

- The Gravitational N-Body problem is relatively simple at a conceptional level. Nevertheless, it represents a challenge computationally as our physical systems often have millions, billions of particles.
- We have implemented a simple and naive $O(N^2)$ implementation of the simulation. It is quite clear how it can be improved using approximations. We have directed our attention to the Barnes-Hut algorithm since it gives good results $O(Nlog(N))$.
- As the N-Body is in theory *embarrassingly* parallel we aim to insert parallelism our implementation of the Barnes-Hut approximation. We have several options available but will begin with the Tree Partitioning strategy.

References

[1] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324(6096):446–449, Dec. 1986.

[2] E. Forest and R. D. Ruth. Fourth-order symplectic integration. *Physica D: Nonlinear Phenomena*, 43(1):105–117, May 1990.

[3] M. Trenti and P. Hut. N-body simulations (gravitational). *Scholarpedia*, 3(5):3930, 2008.

[4] M. Warren and J. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. pages 570–576. IEEE Comput. Soc. Press, 1992.

[5] M. S. Warren and J. K. Salmon. A parallel hashed Oct-Tree N-body algorithm. pages 12–21. ACM Press, 1993.