Hierarchical Methods for N-body Simulations

Jimmy Chen

27 March 2024

Abstract

Two classic algorithms for force calculations in N-body simulations with pairwise interactions, the Barnes-Hut (BH) algorithm and the Fast Multipole Method (FMM), along with a brute-force method have been implemented in `C++`. The time complexity and error of the algorithms and their scaling with different parameters have been studied. At large particle numbers, FMM was found to be the most efficient of the three, followed by BH, both of which showed close-to-linear scaling with particle number. The three methods were then used with a leapfrog integrator to simulate a number of physical scenarios, and their results were compared, showing good agreement in their qualitative behaviours. Excluding this abstract and the appendix, the word count comes just below 3000.

1 Background

N-body simulations have been a great tool to astrophysicists for the study of galactic and celestial dynamics. As the problems of interest grow increasingly large in scale, the calculations are also becoming more computationally demanding. In an age when Millennium Runs regularly simulate over 10 billion particles [1], efficient execution of the simulation is essential, and the naive approach of complexity $\mathcal{O}\left(n^2\right)$ proves completely untenable. Indeed, much work has been done to improve the efficiency of force calculation, the typical bottleneck in the simulation. Many such algorithms have been proposed, typically trading small amounts of error to significantly boost performance.

One category of such algorithms is known as hierarchical methods, which involves recursively subdividing space into boxes of octants, and applying special treatment to particles in well-separated boxes. In this paper, I present implementation of two well-known hierarchical approaches, the Barnes-Hut (BH) method and Fast Multipole Method (FMM), which differ in their treatment of distant boxes.

In the next section, I will discuss mathematical prerequisites, after which the algorithms themselves are presented. Section 4 provides detailed complexity analyses and simulation results, and I discuss optimization efforts in Section 5. Finally a summary is given in Section 6.

2 Mathematical background

Before we can approach the algorithms, we must look at multipole expansions, which lies at the heart of our implementation. For lack of space, the treatment is necessarily very brief, and I refer interested readers to Appendix A of [2] for detailed mathematical derivations.

Our aim is to find for an inverse-square law force of coupling constant G, expressions for the potential ψ and force $\boldsymbol{g}$ at one cluster of charges b, with centre of charge (COC) r_b, due to a far-away, well separated cluster a with COC r_a. For a particle j in b at $\boldsymbol{r}' = \boldsymbol{r}_b + \boldsymbol{r}_j$ with charge q_j, the potential due to a particle i in a at $\boldsymbol{r} = \boldsymbol{r}_a + \boldsymbol{r}_i$ with charge q_i is written

$$\psi(\boldsymbol{r}') = G\frac{q_i}{|(\boldsymbol{r}_j + \boldsymbol{r}_b) - (\boldsymbol{r}_i + \boldsymbol{r}_a)|} \tag{1}$$

Note that the denominator is in fact $(\boldsymbol{r}_b - \boldsymbol{r}_a) + (\boldsymbol{r}_j - \boldsymbol{r}_i)$, where, since the clusters are well separated, the first term in brackets dominates. This invites us to perform a Taylor expansion in the small parameter (second term) and truncate at the p-th order, the result of which is a *multipole expansion of order p*. The force can then be found with $\boldsymbol{g} = -\boldsymbol{\nabla}\psi$. So far we have worked free of a basis: The resulting expression can be written in Cartesian form, or in a more convenient form involving solid spherical harmonics, which I adopt in my code.

It turns out that in spherical harmonics expansions, we only need two sets of coefficients M_n^m and F_n^m $(0 \leq n \leq p,\ -n \leq m \leq n)$ localised at the COCs of clusters a and b to completely describe an expansion of order p. Convenient expressions, known as *kernels*, enable us to calculate M_n^m from a's charge distribution, F_n^m from M_n^m and forces on b particles from F_n^m. Some additional kernels exist. They have all been given convenient names as shown in Table 1. Because most of them take a form similar to matrix multiplications between one set of coefficients and the set of solid harmonics of order p, the operations are generally of complexity up to $\mathcal{O}\left(p^4\right)$.

Table 1: Kernels in the multipole expansion.

Name	Purpose
P2M	Find M_n^m from local charge distribution
M2M	Shift M_n^m to new expansion centre
M2L	Find F_n^m from distant M_n^m
M2P	Find ψ, $\boldsymbol{g}$ on particle from distant M_n^m
L2L	Shift F_n^m to new expansion centre
L2P	Find ψ, $\boldsymbol{g}$ on particle from local F_n^m

3 Algorithmic approach

In this section, I will outline the working of the Barnes-Hut and Fast Multipole methods. A brute-force algorithm of complexity $\mathcal{O}\left(n^2\right)$ naively enumerating each pairwise interaction has also been implemented as a reference but will not be discussed further here.

3.1 Barnes-Hut method

The Barnes-Hut approach starts by constructing an *octree* data structure, with which it is easy to determine when a particle is well-separated from a region in space. This allows us to make approximations by replacing the charges in that space with a multipole expansion from which forces are derived efficiently.

3.1.1 Octree construction

The octree is a tree structure representing 3D space. Each node represents a cuboid in space and can have up to 8 children corresponding to the 8 cuboid octants of the parent node. The root node represents the whole space of interest.

We can store on each node the particles in the represented region, total mass, COC and multipole coefficients M_n^m, F_n^m at the COC up to order p [1]. The storage of pre-processed information is key to our speed-ups. A general application of the octree requires that each leaf node holds no more than $n_{\max}$ particles; For Barnes-Hut, $n_{\max} = 1$.

Such a tree can be constructed with the recursive algorithm in Listing 1, starting from root. This can be done in time $\mathcal{O}(n \log n)$, since the expected mean tree height is of order $\log n$, and this is repeated for each particle. Similarly the space complexity is $\mathcal{O}(n \log n)$. Nodes are created whenever we visit a previously non-existing one.

Listing 1 Octree construction.

```
def AddParticle(NODE, PARTICLE):
    if NODE is a leaf:
        add PARTICLE into NODE
        if NODE has > n_max particles:
            split NODE into smaller octants
            for particle in NODE:
                AddParticle(correct child_node, particle)

    else:
        AddParticle(correct child_node, PARTICLE)

    update COC, total mass on NODE
```

To initialise the M_n^m coefficients, we then perform a depth-first search (DFS), which calculates M_n^m at the leaf nodes first with the P2M kernel, and then on returning to the parent level, calculate the parent M_n^m by summing the M2M-shifted child node M_n^m coefficients.

3.1.2 Barnes-Hut force calculation

With the octree ready, we are now able to calculate the forces. Introducing the parameter θ, the *opening angle*, we can define a *multipole acceptance criteria*(MAC):

[1] The original Barnes-Hut paper stores only the total mass and so is in fact the zeroth order approximation. I extended this to work for higher order expansions.

Multipole Acceptance Criteria We are allowed to multipole-expand a source box if $d/x < \theta$, where d is the length of the longest side of the box, and x is the distance from the particle of interest to the box COC.

The Barnes-Hut calculation on one particle is then shown in Listing 2. We simply repeat this procedure on each particle to get all the forces.

Listing 2 Barnes-Hut force calculation.

```
def Interact(NODE, PARTICLE):
    if NODE is a leaf:
        add force exerted by single particle in NODE to PARTICLE
    elif NODE, PARTICLE satisfy MAC:
        calculate force exerted by NODE on PARTICLE via M2P kernel
    else:
        Interact(child_nodes, PARTICLE)
```

The algorithm essentially simplifies calculation of force from distant boxes by replacing all those pairwise interactions with multipole expansion results. it can be shown that this is also of complexity $\mathcal{O}(n \log n)$ [3], leading to an overall complexity of $\mathcal{O}(n \log n)$. There is an additional dependence on expansion order p, bounded above by $\mathcal{O}(p^4)$, since not all interactions are calculated from the multipole expansion.

3.2 Fast multipole method

We again build the octree at the start of FMM, this time setting $n_{\max} \sim \mathcal{O}(1)$, which modifies the complexity of each step in Listing 1, and therefore the overall complexity, only by a constant.

At the heart of FMM is the approximation of well-separated box-to-box interactions with multipole expansions. This is made possible by using the M2L kernel, an operation independent of particle numbers. We again need a multipole acceptance criteria, this time between two boxes:

Multipole Acceptance Criteria Two boxes are considered well-separated if the sum of the lengths of the maximum sides of the boxes l and the distance between box COCs d satisfy $l/d < \theta$ where θ is again the opening angle.

We can then make use of the M2L kernel with the algorithm in Listing 3. To understand its effect, consider the chain of nodes leading from the root to an arbitrary leaf node, $\{a_0, a_1, ..., a_n\}$. As `Interact` descend to node a_i, boxes satisfying the MAC $\{b_i\}$ are expanded, and their influence is recorded in the F_n^m of a_i. For descendants of a_i then, the children of b_i are now irrelevant. Moving down to the next level a_{i+1}, yet some more nodes now satisfy the MAC and will be multipole-expanded. The number of such nodes however should be *roughly constant* on each level: only boxes sandwiched between nearby boxes that do not satisfy the MAC and those already expanded on the last level ($\{b_i\}$) can be considered. We continue descending, until it becomes necessary (both nodes are leaves) or more economical (number of direct interactions below n_{direct}) to perform brute-force directly.

Listing 3 Dual-tree traversal.

```
def Interact(NODE1, NODE2):
    if (NODE1, NODE2 are both leaf nodes
        or have fewer than n_direct interaction pairs):
        Calculate pairwise forces directly
    elif NODE1, NODE2 satisfy MAC:
        Accumulate F coefficients on both nodes via M2L kernels
    else:
        if NODE1 is NODE2:
            for all pairs child_node_1, child_node_2 in NODE1:
                # including child_node_1 == child_node_2
                Interact(child_node_1, child_node_2)
        else:
            for child_node in larger NODE:
                Interact(child_node, smaller NODE)
```

All the necessary interactions to a_n are therefore imprinted in its ancestors' F_n^m coefficients. To calculate the total force on each particle of a_n, we descend from a_0 to a_n, shifting F_n^m to the child node COC and adding it to the child node F_n^m at each step with the L2L kernel, and finally apply the L2P kernel on particles in a_n. This can be done for all leaf nodes in one go with a simple push-down-then-descend DFS after `Interact` has finished.

In building the tree, the P2M kernel calculates M_n^m coefficients on all leaf nodes in time $\mathcal{O}(n)$. It is then passed upward in another $\mathcal{O}(n)$ operations. The same is true the downward pass of F_n^m via L2L and the conversion of F_n^m to forces. The intermediate `Interact` function is also expected to be of order $\mathcal{O}(n)$ or even less [4]. `Interact` will however still dominate the execution time, for it calls the time-consuming M2L kernel much more than the other parts call their kernels.

4 Tests and simulations

Having presented the algorithms themselves, I evaluate their performance and robustness now through various tests and simulations.

All tests and simulations were conducted on a MacBook Pro early 2015 with an Intel i7-5557U@3.1GHz and 16GB RAM. The system is a fully upgraded Arch Linux as of 26th March with Linux kernel 6.8.2. All code has been compiled with `clang++` version 17.0.6 and the following optimization flags:

```
-std=c++17 -O3 -DNDEBUG -ftree-vectorize -flto -finline-small-functions
-march=native
```

For best stability, Turbo Boost had been disabled, and CPU governor had been set to `performance`, which disables dynamic scaling and fix the CPU frequency to its highest (3.1GHz). The laptop was charged throughout, and all foreground processes were closed. No throttling was observed during the runs.

4.1 Complexity and error tests

Various tests have been conducted where I change one parameter shared between Barnes-Hut and FMM, while other parameters are fixed with values specified in Table 2. For each parameter, 10 random uniform distributions of charges are generated from different (but known) seeds as repeats, then fed to brute-force, Barnes-Hut and FMM. They are then timed by `std::chrono` calls which sandwich the computational code. Finally the timing and computation results are saved to files, to be processed by `Python` scripts. We define the error in a hierarchical method to be the relative error against brute-force results, and use the error averaged over all components of the force and over the 10 repeats in our figures below.

Table 2: Default parameter values.

Variable	Default
n	1000
θ	0.5
p	3
n_{direct} (FMM)	3
n_{max} (FMM)	5

4.1.1 Effect of changing n

As expected, both Barnes-Hut and FMM show close-to-linear scaling against number of masses, demonstrating their superiority at large n. Code profiling with `valgrind` demonstrates that both code spends the most time in their `Interact` routines (97% for Barnes-Hut, 82% for FMM).

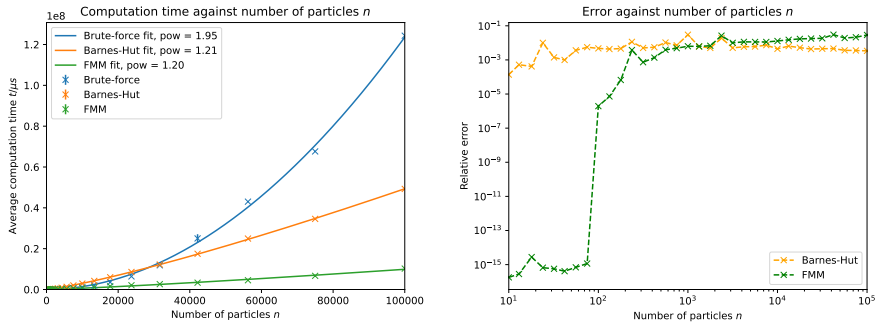
The Barnes-Hut scaling has a much larger constant of proportionality - This is due to the need to compute multipole expansions for each sink particle, which is very demanding. FMM does not face the same problem, due to its use of box-to-box approximations.

At large n, the mean error is largely constant. The very small error of FMM at small n is likely because most interactions are allowed to proceed through direct brute-force calculations.

4.1.2 Effect of changing θ

θ sets what we consider to be well-separated in Barnes-Hut and FMM. As we reduce θ from 1 in Figure 2a, there is first a rise in computation time, caused by FMM and BH having to do more multipole expansions at deeper levels of the octree, followed then by a drop, corresponding to the reduction to brute-force (brute-force is still quick at $n = 1000$, therefore the *speed-up*.) At the small θ limit, both methods have a constant overhead above brute-force. Barnes-Hut is significantly slower, since it needs to traverse the entire octree for each particle, whereas FMM benefits from the one-shot dual-tree traversal and a shallower tree thanks to $n_{\text{max}} \geq 1$.

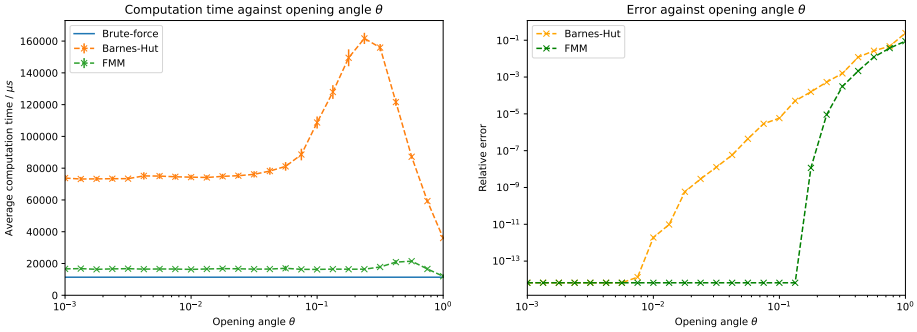
Barnes-Hut displays a slower drop in error: This is likely because its MAC, which only concerns one box, can allow multipole expansions at smaller θ. The low-θ limits in error is



(a) Time against n (b) Error against n

Figure 1: Response to variation in n.

likely floating point errors in the computation due to different calculation orders between BH, FMM and brute-force.



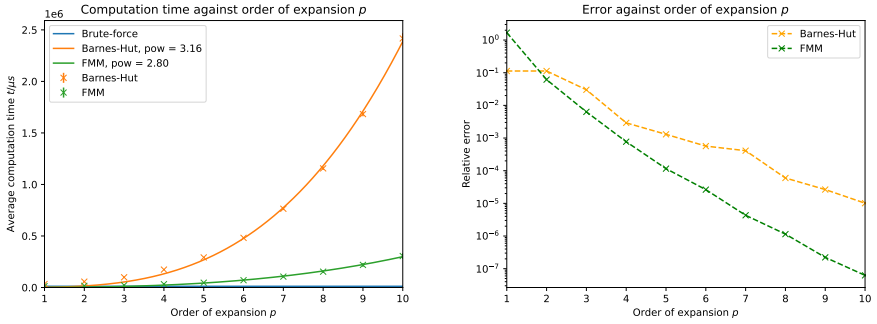
(a) Time against θ (b) Error against θ

Figure 2: Response to variation in θ.

4.1.3 Effect of changing p

As explained in Section 3.1.2, we expect the complexity dependence p to be bounded by $\mathcal{O}\left(p^4\right)$, but lower in practice. This is indeed seen in the timing results of Figure 3. FMM has a smaller power law exponent, likely because its conditions for direct interaction is more permissive.

As for the error, multipole expansions should converge quickly with reasonable MACs. This is the case for both, but FMM is more sensitive to the value of p, because it employs five multipole kernels in one calculation compared to three in BH. Another interesting feature is that error distributions in BH for $p = 1$ and $p = 2$ are almost identical. This is not fully understood, but inspection shows second order terms have been suppressed, possibly due to properties of the uniform distribution.



(a) Time against p

(b) Error against p

Figure 3: Response to variation in p.

4.1.4 Error distributions

For each combination of parameters, we have taken their error distributions. We only show six distributions from BH with varying θ (Figure 4), as most distributions share similar features. On a logarithmic scale, tightening the constraint (reducing θ here) causes the distribution to uniformly shift to the left, but the same shape is maintained, until θ is so small the MAC simply cannot be met. This is when BH transitions to the brute-force regime (Figures 4d to fig. 4f).

Another feature is the long tail towards the right: in each case a small portion of particles suffer from significant error. This can be remediated by using a more sophisticated MAC [2].

4.2 Simulations

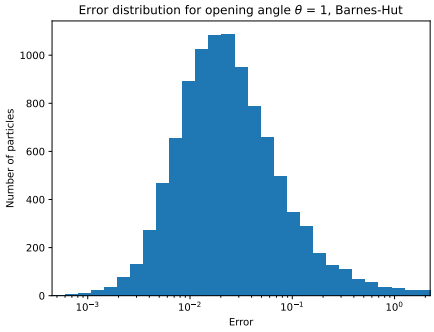
To verify the reliability of my algorithms, I have run simulations under three initial conditions and compared brute-force, BH and FMM results. After each round of force calculation, a leap frog integrator evolves the system forward with step size $\delta t = 0.01$, for a total time of $\Delta t = 10$. The default parameters mirror those in Table 2. The three initial conditions are:

1. Cold start

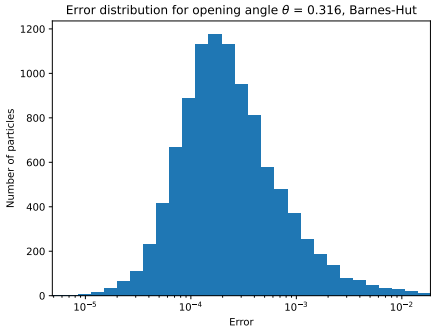
   Generate a random uniform distribution of stationary charges in a sphere, and assign masses (and gravitational charge) to each with a normal distribution.

2. Single "galaxy"

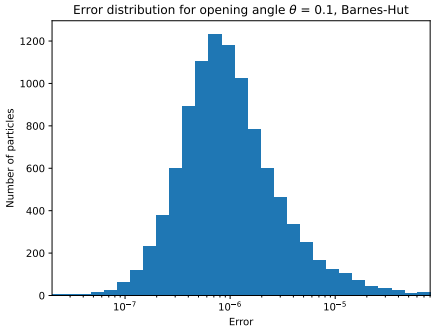
   Generate a random uniform distribution of charges on a thin disk about set axis, excluding a small central region. Add a "supermassive black hole" (SMBH) - particle of very high mass at the centre, and set all particle velocities to that of circular motion around the SMBH.



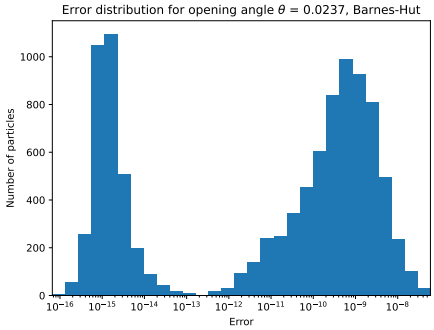
(a) $\theta = 1.0$



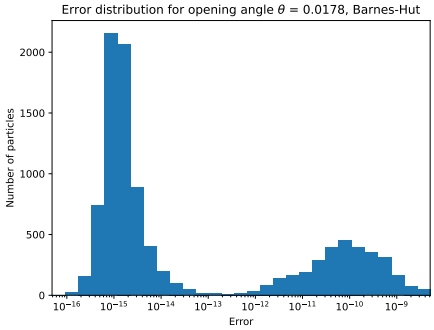
(b) $\theta = 0.316$



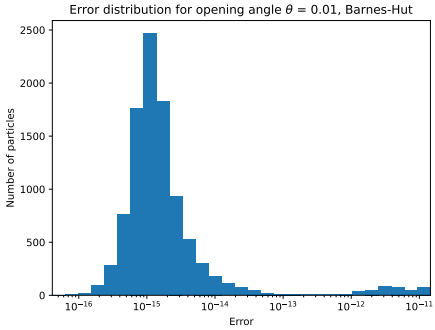
(c) $\theta = 0.1$



(d) $\theta = 0.0237$



(e) $\theta = 0.0178$



(f) $\theta = 0.01$

Figure 4: Distribution of errors as θ is reduced.

3. Two “galaxies”

   Generate two galaxies as above, tilted at different angles, then boost them with velocities required for the SMBHs at the centre to perform circular motion about their centre of mass.

Due to lack of space we present and discuss the most interesting third case only here, while some results from the other two can be found in the Appendix. Videos of full simulations are also available.

Snapshots of the third system, shown in Figure 5, display very good qualitative agreement across the three methods. As the galaxies rotate around each other, many outer stars do not feel enough attraction to keep following the galaxies' orbit, and are scattered outwards. This is not seen in the simulation of a single galaxy, and is thus a unique feature of this model. The stars at the centre on the other hand feel similar attractions from both galaxies, and can be exchanged between the them. Interestingly, the galactic centres appear further apart at half-period, likely because the attraction is reduced when the galaxies are facing away from each other. At a full period the centres are observed to return to almost the same initial locations.

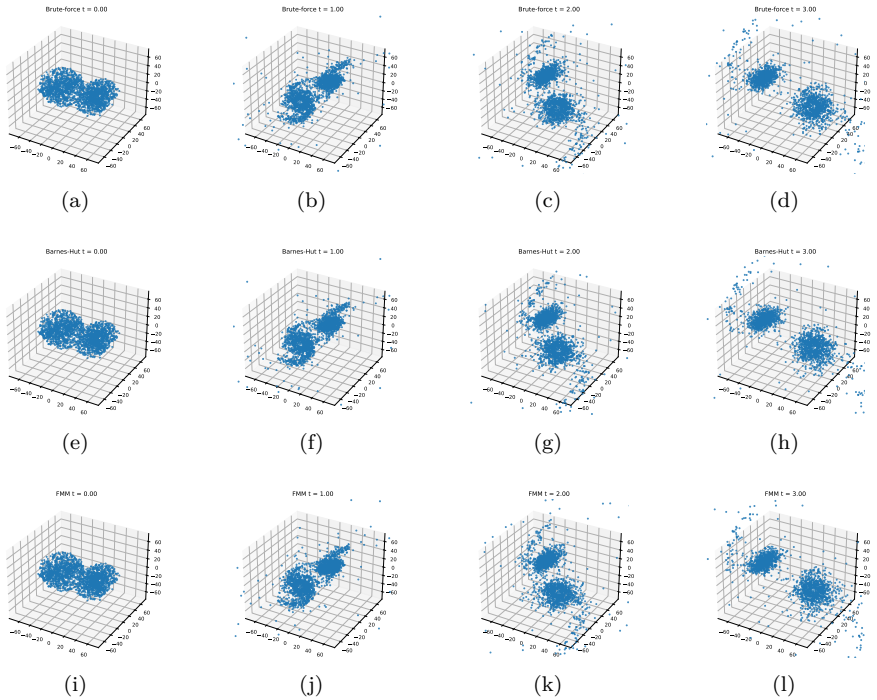


Figure 5: Simulation of two galaxies with total $n = 2000$. Top to bottom: brute-force, BH, FMM. Left to right: $t = 0, 1, 2, 3$.

We can further inspect the change in energy and angular momentum against time (Figure 6). Several “steps” are seen as energy conservation is severely violated, but these are

likely caused by the leap frog integrator evolving a particle with very large acceleration. In fact, in between every jump, the energy has been reasonably well conserved for the two hierarchical methods, demonstrating their sufficiency. In order to prevent the sharp "steps" however, gravitational softening will be needed, which demands modification of the algorithms.

On the other hand, angular momentum is much better conserved. Note an interesting "synchronised", "smooth" pattern in FMM's angular momentum components, which is likely correlated with the relative positions of the galactic centres, and which might even be understood with a perturbative treatment.

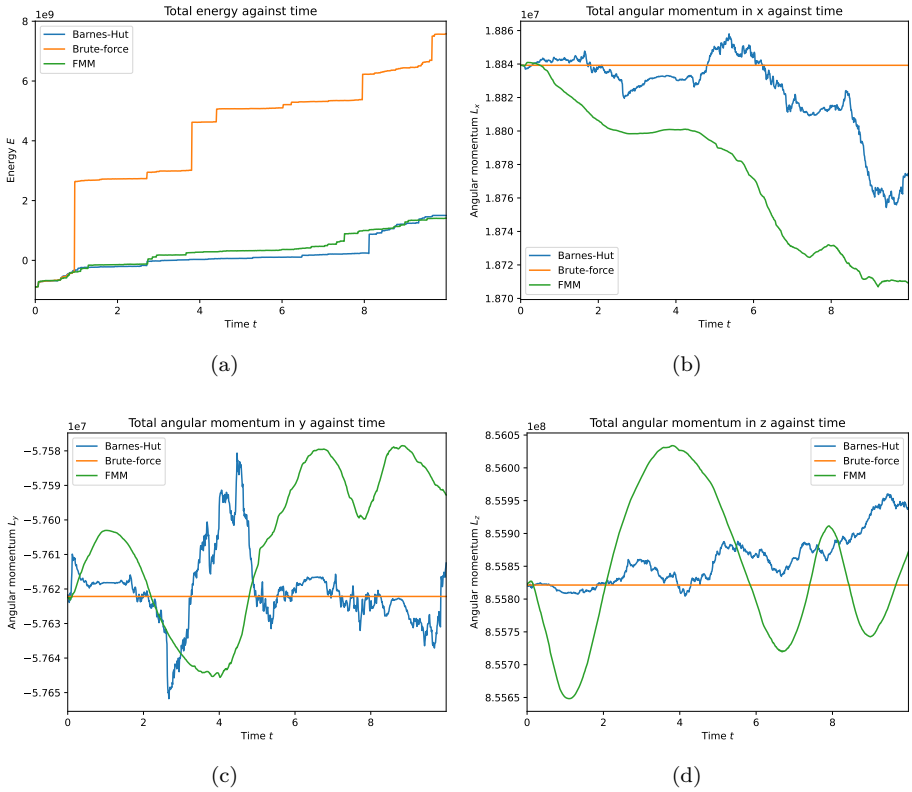


Figure 6: Change of energy and angular momentum as simulation progresses.

5 Optimizations

Much effort has been made to speed up the running of the code, with the aid of profiling tools such as `valgrind`. Here I briefly outline some techniques used, and note some improvements that can be made later on.

`-O3` and `-ftree-vectorize`. The two compiler flags turn on the most aggressive optimizations and auto-vectorization of our program, which greatly helps with loops encountered

in vector operations. The combined effect is a near two-fold increase compared to `-O2` optimizations.

Memory allocation. Because of stack limitations, the `grid` carrying a list of all particles (containing many 3D vectors) has to allocate its particles on the heap. However, all other particle instances are stored on the stack, reducing the allocation and access times. This leads to a 10-fold speed-up compared to when all particles are dynamically allocated. After each time evolution, the previous `grid` is written to disk and deallocated to save memory.

`soul` labels. To track the particles within an octree node, instead of storing copies of the particle we merely store a label with which we can find the particle in a separate list. This can significantly reduce the memory requirements.

Recurrence relations. We can pre-process all solid harmonics of order p in time $\mathcal{O}\left(p^2\right)$ by harnessing their recurrence relations with simple math operations, instead of directly evaluating their computationally costly expressions based on Legendre polynomials in the 4th order loops of the kernels. This change brought a 10x speed-up to BH and FMM.

Multipole kernels. My spherical harmonics implementation of complexity $\mathcal{O}\left(p^4\right)$ already beats the naive Cartesian expansion of order $\mathcal{O}\left(p^6\right)$. However, the complexity can be further reduced to $\mathcal{O}\left(p^3\right)$ if we rotate the interactions onto the z-axis [2]. This might not be better for computations of my size however, because of the extra overhead of rotation operations.

6 Conclusions

I presented implementations of two hierarchical algorithms, Barnes-Hut (BH) and Fast Multipole Method (FMM) for the accelerated calculation of N-body inverse-square law interactions. Their time complexities, in particular, the close-to-linear scaling with the number of particles, were shown to be consistent with theoretical descriptions. With reasonable parameters, FMM was found to be superior in nearly every way to BH, and both consistently outperform brute-force in a large system. The program can calculate forces on 10^5 particles in 10 s with FMM, where it would have taken brute-force 120 s.

The algorithms have additionally been used to perform simulations on simplified models such as collision of two galaxies, and good agreements were seen between the hierarchical approaches and a brute-force calculation.

As a next step, energy conservation can be improved by using higher order integrators and incorporating softened gravity, and we can extend the range of applicability by building in e.g. the Stokes and Helmholtz kernels. It will also be possible to introduce parallelism such as `OpenMP` or `MPI` so the code can handle much larger systems on a computing cluster.

A Appendix

A.1 Code submission

The submitted code should contain all the information required to build the project. `README.md` in the base folder contains a short introduction, as well as build instructions. Some build targets, such as this report and documentation have been generated and included with the submission for the convenience of the reader.

The full codebase has been documented and a symlink to the generated documentation can be found at `refman.pdf` in the project base. A list of source files can be found in the *File Index* section of the documentation.

A.2 Simulation results

Below we list partial results to the cold start and single galaxy initial conditions. Side-by-side animated comparisons of results from the three algorithms, for all three initial conditions, are included with the submission.

A.2.1 Cold start

The initial condition for the cold start has been described in Section 4.2. In this run, $n = 500$. Snapshots are shown in Figure 7; energy and angular momentum change, in Figure 8.

A.2.2 Single galaxy

For this we use $n = 1000$. Snapshots are shown in Figure 9. Most of the masses away from the centre are in fact circling the centre. The details of the evolution are not at all obvious from snapshots alone, and the reader is therefore encouraged to check out the actual animations.

For energy and angular momentum change, see Figure 10.

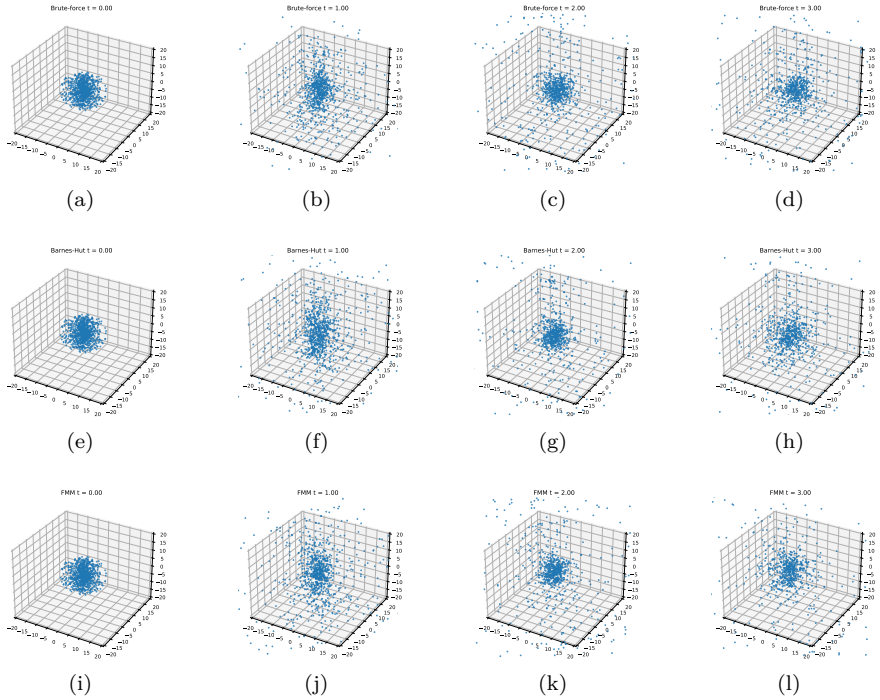


Figure 7: Simulation of cold start with total $n = 500$. Top to bottom: brute-force, BH, FMM. Left to right: $t = 0, 1, 2, 3$.

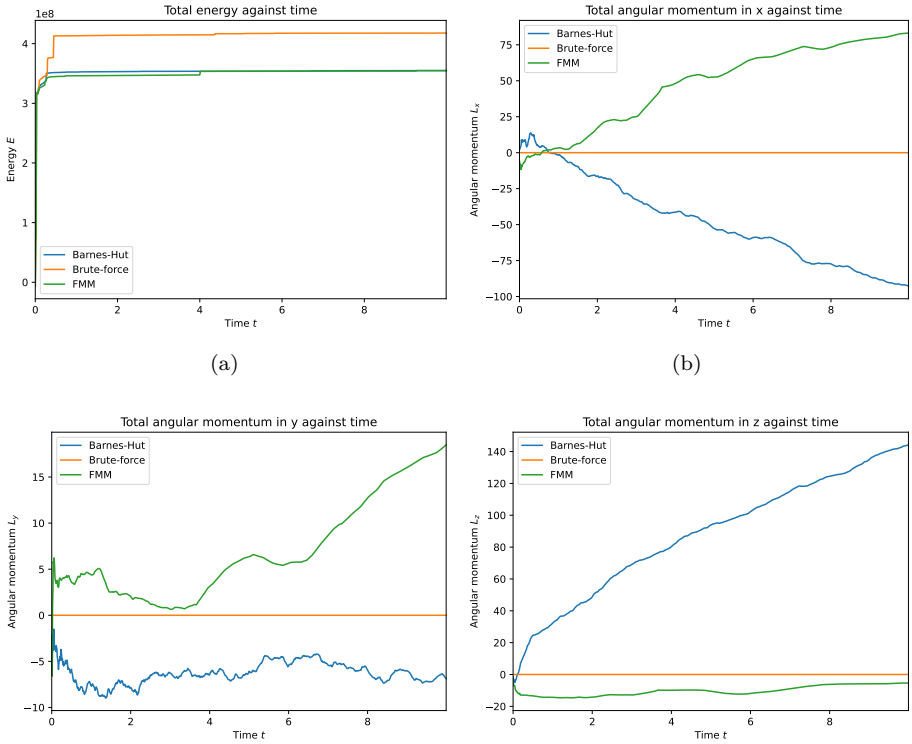


Figure 8: Change of energy and angular momentum in cold start.

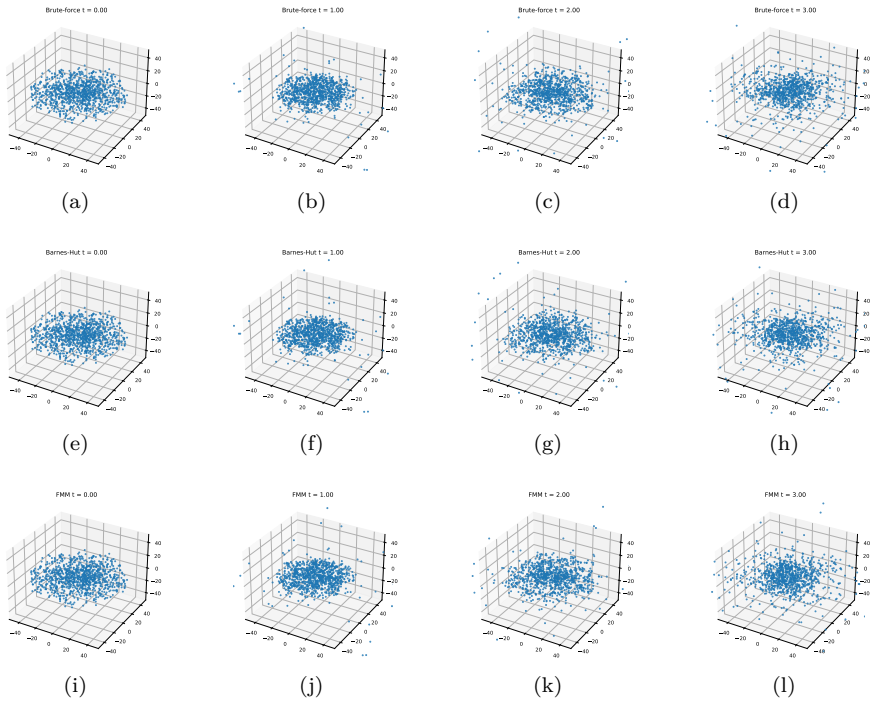


Figure 9: Simulation of a single galaxy with total $n = 1000$. Top to bottom: brute-force, BH, FMM. Left to right: $t = 0, 1, 2, 3$.

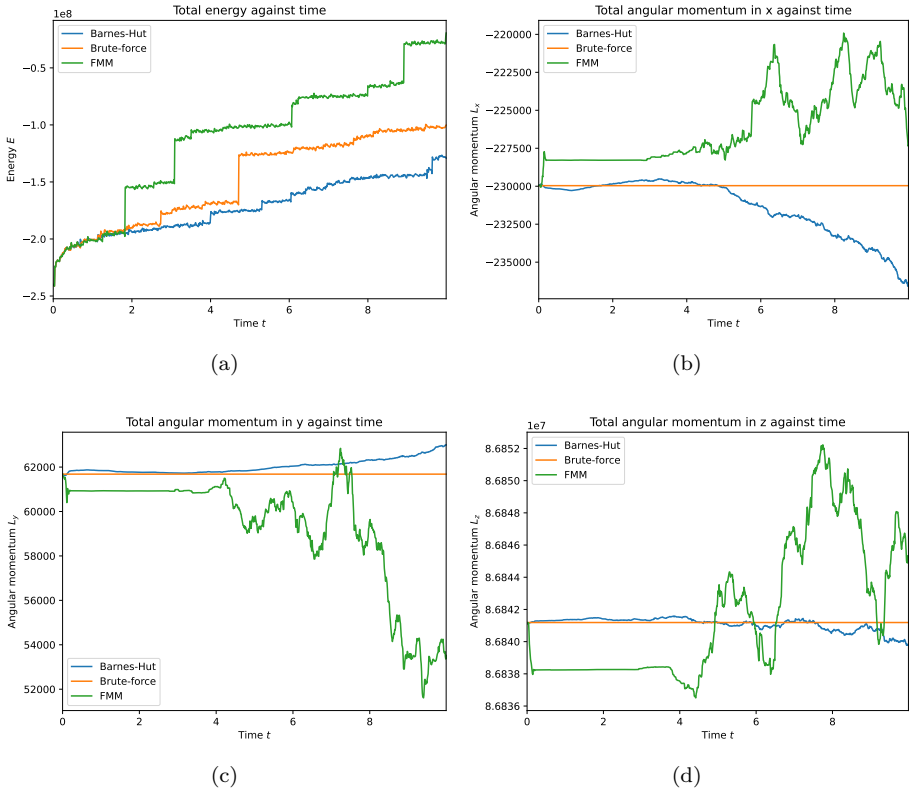


Figure 10: Change of energy and angular momentum in a single galaxy.

References

[1] V. Springel et al., *Simulations of the Formation, Evolution and Clustering of Galaxies and Quasars*, Nature **435**, 629 (2005).

[2] W. Dehnen, *A Fast Multipole Method for Stellar Dynamics*, Computational Astrophysics and Cosmology **1**, 1 (2014).

[3] J. H. Barnes and P. Hut, *A Hierarchical o(n Log n) Force-Calculation Algorithm*, Nature **324**, 446 (1986).

[4] W. Dehnen, *A Hierarchical o(n) Force Calculation Algorithm*, Journal of Computational Physics **179**, 27 (2002).