THE FAST MULTIPOLE METHOD FOR THE N-BODY PROBLEM

PHILIP MURZYNOWSKI

Introduction. Though the fast multipole method finds use across many disciplines, it can often most easily be described in the context of the n-body problem, which is the context in which it will be pursued in this project.

Briefly, the n-body problem is the problem of predicting the motion of particles due to inter-particle forces generalized to some n number of particles.

In the cases of $n = 1$ or $n = 2$ closed form solutions are widely known with solutions occasionally present in special conditions for slightly more bodies such as $n = 3$. [5] However, general solutions for any number n bodies are not known and computation of this problem has relied on simulation and development of numerical algorithms, as naively computing the interactions between all n particles by summing the contributions from all $n - 1$ remaining particles for each particle results in a complexity of $\mathcal{O}(n^2)$. [5]

The Fast Multipole Method is one such algorithm, famous for its asymptotic complexity and guarantees with respect to accuracy. The fast multipole method can perform the computation in linear complexity, $\mathcal{O}(n)$, multiplied by a constant term dependent on the desired error, which in practice often works out to a small term on the order of 24 and 53 to achieve accuracy within machine epsilon for Float32 and Float64, respectively. [14] It achieves this in four main ideas: approximating the contributions between well separated regions, defining separate regions in a hierarchical decomposition, reusing computation when traversing the hierarchical decomposition, and representing regions with a series expansion to obtain greater accuracy. [20] [16]

Specifically for this project, to keep the scope appropriate for a one semester, the focus is on gravitational forces between particles in two dimensions, combined with a basic time integration scheme such as Verlet integration which is symplectic and provides good numerical stability at a small computational cost, allowing for the simulation of the movement of the n bodies and animation.[6]

The implementation is validated by generating a number of particle configurations in two dimensions with approximately uniformly distributed particles, specifying the number of expansion coefficients needed for a specified error tolerance, and then verifying that the Fast Multipole Method implementation provides force values for each particle within the specified error tolerance as compared to the naive $\mathcal{O}(n^2)$ algorithm.

The algorithm is written in Julia due to its capability for high performance and ability to provide high level operators and abstraction. Libraries of particular usefulness included BenchmarkTools.jl for accurate runtime and memory benchmarking, GFlops.jl for flop counts, the Plots.jl ecosystem due to its easy plotting and animation environment. Continued future work may additionally include the use of StaticArrays.jl which for arrays with a small number of elements can provide significant speedup due to the size being known to the compiler allowing for stack allocation and other potential algorithmic improvements, and LoopVectorization.jl to provide performance enhancements such as vectorization and loop unrolling wherever appropriate as long as it has been ensured that this will not result in consequences for stability and accuracy, but was not yet employed due to time constraints and compatability issues.

The algorithm is additionally compared to the Barnes Hut algorithm, an alternative well-known algorithm for computation of the n-body problem. The most important numbers in this comparison are of course accuracy and runtime, though as runtime requires careful optimization of both algorithms for a fair comparison, the runtime complexity is analyzed from a perspective of flopcount using GFlops.jl on a ported implementation oh the Barnes-Hut Algorithm based on ParallelBarnesHut.jl.

1. Historical Context. Prior to creation of the Fast Multipole Method, a major development in n-body simulation took place in 1986, with publication of "A Hierarchical $\mathcal{O}(NlogN)$ Force-calculation Algorithm" by Barnes and Hut, today commonly referred to as the Barnes-Hut algorithm. Before the Barnes-Hut algorithm, there had been two main approaches to n-body simulation. [10] The first was the obvious, direct $\mathcal{O}(N^2)$ approach which clearly suffered from much poorer asymptotic complexity, and the second involved forming and fitting a potential-model to the n-body simulation which was in fact also $\mathcal{O}(NlogN)$ in complexity. However, though the asymptotic complexity of the second method was favorable, it came with substantial drawbacks in terms of accuracy as well as the need to specialize the algorithm for the geometry of each particular simulation.[10] Just before the publication of Barnes-Hut algorithm, research into tree based codes was developing but introduced additional error which was difficult to analyze. [10] [2] Therefore when the Barnes-Hut algorithm was developed, its simplicity of implementation and generality with reasonable error characteristics despite being an $\mathcal{O}(NlogN)$ approach marked a significant development in the history of n-body simulation. [10]

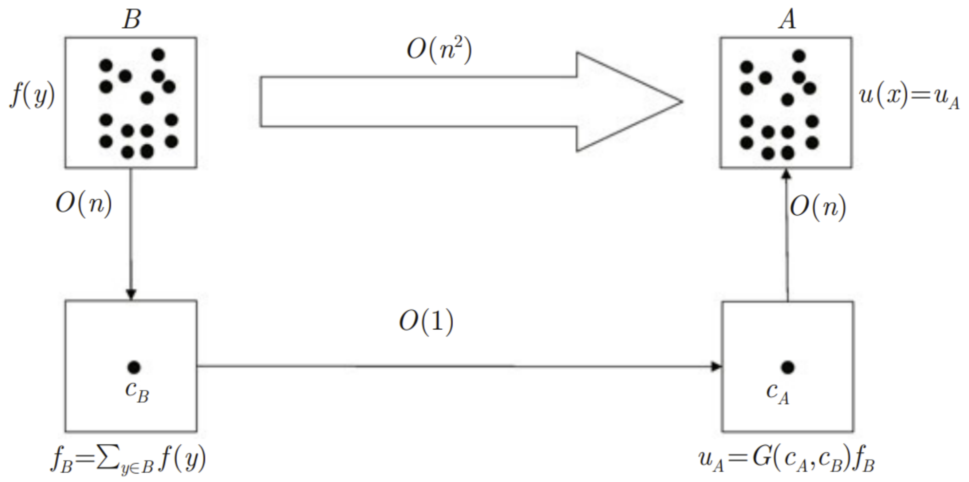
The next major milestone in n-body simulation came in 1987, when the initial Fast Multipole Method itself was developed for n-body simulation by Rokhlin and Greengard, presenting the famous algorithm of $\mathcal{O}(N)$ complexity multiplied by a constant factor for desired error tolerance [14]. Interestingly, the initial inspiration for the algorithm largely derived from the results of work by Rokhlin in 1985, who did not yet use the term Fast Multipole Method but introduced a highly similar solution in the context of Laplace's equation, specifically for two dimensional integral equations. [23]. In Greengard and Rokhlin's 1987 publication, "A Fast Algorithm for Particle Simulations," the notion of hierarchical decomposition was introduced along with generalization for application beyond the problem of n-body computation. [14] [11]. Since then, numerous variants and extensions of the Fast Multipole Method have been created, denoting the effect of the FMM on subsequent technological advancements as extremely influential due to its broad applicability across multiple disciplines, most easily seen in its listing as one of the ten most important algorithms developed in the 20th century. [3] To quickly name a few other fields which enjoy the application of the Fast Multipole Method, these include molecular dynamics, boundary integral equation methods, elastostatics, elastodynamics, fluid dynamics, wave problems, and acoustical and electromagnetic scattering problems. [11][18] Some examples of notable variants proposed after the initial paper include the Adaptive Fast Multipole Method, a method not dependent on the distribution of particles developed to compensate for the shortcomings associated with the restraint of assuming a uniform distribution of particles and also of complexity $\mathcal{O}(N)$, [8], extension to oscillatory kernel functions with $\mathcal{O}(NlogN)$ complexity [16], acceleration of the Fast Multipole Method for the Hemholtz equation, [13], and for the three dimensional Laplace equation the conversion of the multipole expansion into plane-wave expansions to obtain a method of lower complexity. [12] [1] Focusing somewhat more on performant implementation of the Fast Multipole Method rather than later numerical variants, many variants with parellelized computation have also since been introduced. [15] [17] [19]. However, turning the scope back to the problem of n-body simulation, competing algorithms to the Fast Multipole Method include Mesh-Code methods often used for high density particle simulations, Panel Clustering, Symplectic methods, Self-Consistent Field methods, and Barnes Hut, the last of which a description of and comparison to is provided towards the end of this paper. [22] [11]

The Four Main Ideas Of the Fast Multipole Method. The fast multipole method can be outlined by four central ideas: first, the computation between well distanced regions by means of a low rank approximation, second, hierarchically decomposing the space spanned by the N bodies, third, reusing computation when propagating information between levels of the hierarchical decomposition, and fourth, representation of boxes with multipole expansions and associated manipulations of these expansions for increased accuracy. [20][16] The ideas will be outlined in the context of computing potential of bodies, as the expansions for potential can be promptly turned into forces by taking the derivative analytically, and most importantly due to the potential functions analytical nature, the error bounds provided follow the same form as derived for potential. [14]

The first idea, approximating the computation between well distanced regions, is a central aspect of the motivation for the fast multipole method, as it offers insight into the potentially sizable performance gains available. To start with an example,[16] if we would like to compute the potentials of stars in two unique galaxies which are separated by distances numerous orders of magnitudes greater than the distances between stars within a given galaxy, then we can break down the computation of the potential into two components, potential contributions from stars within the same galaxy, and potential contributions from stars from the other galaxy. First, for a given star in the first galaxy, we can directly compute the potential due to each other star in the same galaxy for a precise measurement of the potential contribution from the the starts within the same galaxy. If we have N stars total and assume that stars are roughly equally distributed across galaxies, then this would be an $\mathcal{O}(N^2)$ computation for all stars. However, instead of also directly evaluating the potential contribution from each other star in the other galaxy for another $\mathcal{O}(N^2)$ computation for all stars, by recognizing that the differences in distances between a star in the first galaxy and all the stars in the other galaxy is negligible compared to the distances between the two galaxies, we can approximate the stars in the distant galaxy as a single star with mass equal to the sum of the masses of all stars in the distant galaxy, with distance equal to distance from the new single star approximation to the center of all of the stats in the near galaxy, which results in an $\mathcal{O}(N)$ computation for all stars. This is equivalent to a rank one approximation as illustrated in the diagram below.

The second main idea, hierarchical decomposition, provides structure for the space encapsulating all the particles in the simulation such that there is a well defined notion of well-distanced regions and method for applying the associated rank 1 approximation. [14] A fully expanded tree is constructed such that each leaf box in the tree has $\mathcal{O}(1)$ points, meaning the height of the tree is $\mathcal{O}(logN)$ and the number of boxes in the tree is $\mathcal{O}(N)$, and then two boxes are considered well distanced, meaning the rank 1 approximation of the first idea can be applied, if the boxes are at the same depth of the tree and are not neighbors. [14] At the root level, using the notation of depth 0 in this paper, the space has not yet been divided into sub boxes, and at depth 1, with the space divided into 4 boxes, all boxes are neighbors, so in both situations this concept is not yet applicable. However, at depth 2, with 16 boxes, there are box pairs which are not neighbors of each other, so for each box we and all boxes which are not

Fig. 1.1. ***Rank 1 Approximation of well-distanced groups of bodies***



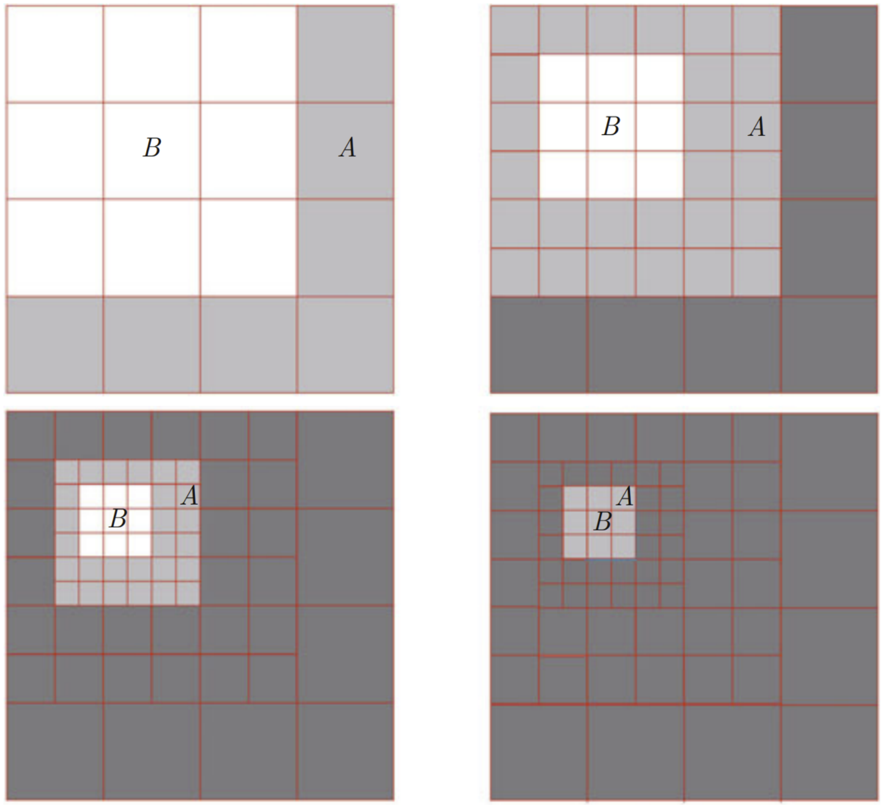
Two groups of bodies whose separation is far greater than the distances between bodies in individual groups can be a potential source of savings in computation with the use of a rank 1 approximation. Reprinted from [16].

its immediate neighbors we can apply the rank 1 approximation.

However, after performing this computation for all distant boxes, there still remains the contribution of the near boxes which needs to be taken into account. These can be handled be traversing a down the tree to the next depth, where the box of interest and its nearest neighbors are subdivided into four smaller boxes each, and as a result there are again boxes among these smaller boxes which are not neighbors, so the rank 1 approximation can be again applied to these smaller boxes. The contribution from small neighbor boxes is again handled by this subdivision and the tree is recursively traversed down in this fashion. Notably, in this manner a box which is not in the leaf depth and at depth of at least 2 will ever need to compute is interactions with at most 27 other boxes, known as the interacting boxes, as boxes far away will have been handled by its predecessors and immediate neighbors by its descendants. [20] At some depth the leaf boxes will be reached at which point the boxes can no longer be subdivided and computation of neighbors offloaded to descendants, but in this scenario a direct all pairs computation can be carried out among the neighbor boxes as in the naive $\mathcal{O}(N^2)$ algorithm, as by construction all boxes at leaf level have $\mathcal{O}(1)$ particles and each box has a constant number of neighbors, so the direct computation for each box is $\mathcal{O}(1)$ as well. [14]

The third main idea is to reuse computation when traversing the tree. Putting aside error analysis momentarily as that will be the major point of discussion in the fourth main idea, from the first two ideas an $\mathcal{O}(NlogN)$ algorithm can be put together in four steps. [14] First, for each box at each depth of the tree, the sum of the masses of all contained points in the box can be computed and stored as the total mass of the box. As there are $\mathcal{O}(N)$ points in the tree which has a depth of $\mathcal{O}(logN)$, meaning each point is contained in $\mathcal{O}(NlogN)$ boxes, this will take $\mathcal{O}(NlogN)$. Second, for each box at every depth, the rank 1 approximate potential contribution from the interacting boxes can be evaluated by using the the total mass of a given interacting box and the

Fig. 1.2. ***Hierarchical Decomposition***



Top left: At depth 2 for an example box B, the well distanced boxes are shaded in gray, of which box A is one of. Top right and bottom left: near neighbors can be further subdivided and computation deferred to descendants, also demonstrating the maximum of 27 interacting boxes. Bottom right: At the leaf level a direct computation must be performed. Reprinted from [16].

centers of the given box and interacting box. This will take $\mathcal{O}(N)$ as there are $\mathcal{O}(N)$ boxes in the tree each having no more than 27 interacting boxes as previously noted. Third, for each particle in the tree, we can sum the evaluated potential contributions from each of the boxes it belongs to to obtain the potential contribution from all boxes in the tree that are not leaf box neighbors of the leaf box that the particle belongs to. As there are $\mathcal{O}(N)$ with each particle belonging to $\mathcal{O}(NlogN)$ boxes, this will also take $\mathcal{O}(NlogN)$. Fourth and finally, the potential contributions of from within a leaf box and its immediate neighbors must be factored in, so for each box at the leaf level, the potential of all contained points is updated with the contributions from all other points in the box and the points in the leaf box's neighboring boxes by direct computation of the $\mathcal{O}(N^2)$ method. However, as outlined before this direct computation is in fact $O(1)$ due to having a constant number of immediate neighbors and particles within each box, resulting in a total runtime of $\mathcal{O}(N)$ for this step. The key observation at this point is that the two $\mathcal{O}(NlogN)$ steps, steps 1 and 3, can in fact be converted to $\mathcal{O}(N)$ by cleverly reusing computation. Instead of performing the computations by examining all of the boxes that every given point is contained in, the computation can be recursively passed either up or down the tree, respectively. [14] Step 1, storing the total mass of the particles in each box at every level, can be done in$\mathcal{O}(N)$ by computing the direct summation of a constant number of particles at the leaf level, and computing the total mass of particles in boxes a level above, the parent boxes, by summing the total mass values of the four child boxes contained by the parent box. Step 3, factoring the potential contributions from boxes at every level besides immediate leaf neighbors to particles in the leaf boxes, can also be accomplished in $\mathcal{O}(N)$ by accumulating and propagating down the tree computed potentials from coarser depths. In this step, each child box simply adds to its computed potential the computed potential of its parent. Thus, as every step in the algorithm is now $\mathcal{O}(N)$, the total runtime of computing potential contributions from all particles in the tree is $\mathcal{O}(N)$.

Finally, the fourth idea, representation of boxes with multipole expansions and associated expansion manipulation, builds upon the framework provided by the first three ideas that when combined together as a whole presents a method which has complexity that is linear in N scaled by a constant factor proportional to desired accuracy, and most importantly can be used to achieve potentially arbitrary accuracy, certainly within machine epsilon. [14] The expansions can be classified into two main categories: outer expansions, representing potential outside of an area as a result of the particles within an area, and inner expansions, the the potential within an area due to the particles outside of it. [21] First, in two dimensions, the coordinates of a point or box center can be described as a complex number, so then outer expansion of potential at a point z outside of the smallest circle encapsulating a given box with center z_0 and n particles of masses $m_1...m_n$ and locations $z_1...z_n$ within can be written as

$$\phi(z) = a_o log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_0)^k} \tag{1.1}$$

$$\text{where } a_0 = \sum_{i=1}^{n} m_i \text{ and } a_k = \sum_{i=1}^{n} \frac{-m_i(z_i - z_0)^k}{k} \tag{1.2}$$

[14] [17] [7] which extends from the observation that a single mass m with location $|z| > |z_0|$ gives a potential function described by

$$\phi_{z_0} = mlog(z - z_0) = m\Big(log(z) - \sum_{k=1}^{\infty} \frac{1}{k}\Big(\frac{z_0}{z}\Big)^k\Big) \tag{1.3}$$

where the second equality is obtainable by expanding the logarithm, and can then can be extended into the general case by performing a summation over all points. The error bound can then be derived by noticing that if we use only $p \geq 1$ terms in the summation, then

$$\Big|\phi(z) - a_0 log(z - z_0) - \sum_{k=1}^{p} \frac{a_k}{z^k}\Big| \leq \Big|\sum_{k=p+1}^{\infty} \frac{a_k}{z^k}\Big|$$

[20] which with the definition of well-distanced boxes of being on the same depth but not neighbors and therefore having at least a box span between them can be further upper bounded by

$$\Big|\phi(z) - a_0 log(z - z_0) - \sum_{k=1}^{p} \frac{a_k}{z^k}\Big| \leq a_0\Big(\frac{1}{2}\Big)^p \tag{1.4}$$

[20] assuming strictly positive masses m_i. This is an extremely significant result, as p terms can be kept at each box to represent the potential contribution to boxes well distanced from it, and the error is halved for ever additional coefficient, meaning that relative error within machine epsilon for Float32 and Float64 can be achieved by approximately 24 and 53 coefficients, respectively. [21] [16]

Manipulating Multipole Expansions. The last main idea in the previous section introduced the representation of a potential outside of a box with an outer expansion and the associated error characteristics, though in fact to make the multipole framework fully compatible with the previous ideas a few more key manipulations of the expansions must be shown, including the method of using a outer expansions to form an inner expansion, the function of the potential inside of a box due to the particles outside of it. [21]

First, for the upward pass, instead of simply summing the total masses in the child boxes to obtain the total mass in the parent box, a method must be defined for translating outer expansions to be written in terms of the center of the parent box prior to summing. Using z_c as the center of a child box and z_p as the center of the parent box, this can be achieved with

$$\phi(z) = b_0 log(z - z_p) + \sum_{l=1}^{\infty} \frac{b_l}{(z - z_p)^l} \tag{1.5}$$

$$\text{where } b_0 = a_0 \text{ and } b_l = -\frac{a(z_c - z_p)^l}{l} + \sum_{k=1}^{l} a_k (z_c - z_p)^{l-k} \binom{l-1}{k-1} \tag{1.6}$$

[17] [20] This can be shown by again using the log expansion from 1.3 and the additional expansion also valid for $|z| > |z_0|$:

$$(z - z_0)^{-k} = \sum_{l=k}^{\infty} \binom{l-1}{k-1} \frac{z_0^{l-k}}{z^l}$$

Most importantly, as a result of the uniqueness of the multipole expansion, using a finite p term expansion of the translated outer expansion allows for the same error bound as presented in 1.4. [20]

After the outer expansions have been propagated from the leaves up the tree, the inner expansions for each box describing the potentials due to particles outside of the box can begin to be formed. As stated in the second main idea, hierarchical decomposition, the computation of the potential within a box can be done in a manner such that at any given level a box will only need to compute the contributions from at most 27 well-distanced boxes, the interacting boxes, and then as shown in the third main idea, the reuse of computation, potential computed at higher levels is recursively passed down to the lower levels until reaching the leaves. This requires two more additional mechanisms, first, a method to convert outer expansions of interacting boxes to a inner expansion for a given box, and second, a method to translate inner expansions from a parent box to child boxes. For the first mechanism, for an outer expansion of an interacting box with center z_d described by coefficients $a_0 ... a_k$, the inner expansion for a given box with center z_L can be written as

$$\phi(z) = \sum_{l=0}^{\infty} b_l (z - z_L)^l \tag{1.7}$$

$$\text{where } b_0 = a_0 log(z_L - z_d) + \sum_{k=1}^{\infty} \frac{(-1)^k a_k}{(z_d - z_L)^k} \tag{1.8}$$

$$\text{and } b_l = -\frac{a_0}{l(z_d - z_L)^l} + \frac{1}{(z_d - z_L)^l} \sum_{k=1}^{\infty} \frac{(-1)^k a_k}{(z_d - z_L)^k} \binom{l+k-1}{k-1} \tag{1.9}$$

[20] with for $p \geq 1$ an associated error bound of

$$\Big|\phi(z) - \sum_{l=0}^{p} b_l (z - z_L)^l\Big| \leq a_0(6e(p+2)+2)\Big(\frac{1}{2}\Big)^{p+1} \tag{1.10}$$

[20] [14] again assuming strictly positive masses and well-distanced boxes, where the form of the function largely stems from a similar log expansion to 1.3 again as well as the fact that

$$\begin{aligned}(z - z_0)^{-k} &= \Big(\frac{1}{-z_0}\Big)^k \Big(\frac{1}{1 - \frac{z}{z_0}}\Big)^k \\ &= \Big(\frac{1}{-z_0}\Big)^k \sum_{l=0}^{\infty} \binom{l+k-1}{k-1} \Big(\frac{z}{z_0}\Big)^l\end{aligned}$$

[20] though the somewhat more involved proof of the bound is deferred to the original paper by Greengard and Rokhlin with the ratio c as defined in their paper greater than or equal to 2. A very high level takeaway is the fact that the error once again approximately halves for every additional expansion coefficient. The second mechanism, translation of inner expansion coefficients $a_0 ... a_l$ from a parent box with center z_p to a child box with center z_c, is possible using

$$\phi(z) = \sum_{l=0}^{p} b_l (z - z_c)^l \tag{1.11}$$

$$\text{where } b_l = \sum_{k=1}^{p} \binom{k}{l} (z_c - z_p)^{k-l} \tag{1.12}$$

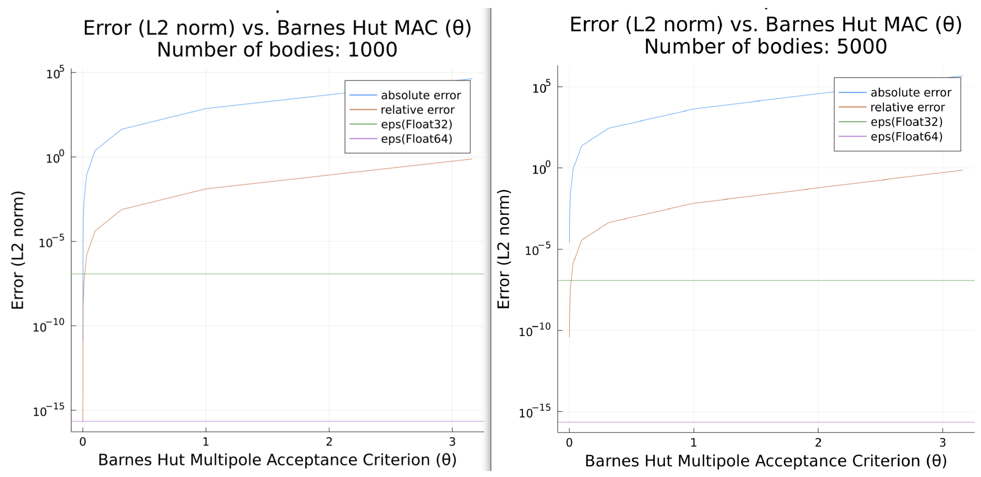
[7] [17] which does not necessitate an additional error bound as it is an exact consequence of Maclaurin's theorem. [14]

The last item that must be commented on for these manipulations is the induced additional complexity on the Fast Multipole Method by their use. Formation of the initial multipole expansions at the leaves is $\mathcal{O}(Np)$ as there are $\mathcal{O}(N)$ particles which each contribution to one p-term expansion, and propagation of the multipole expansion up the $\mathcal{O}(N)$ boxes in the tree is $\mathcal{O}(Np^2)$ as the coefficients in each box are shifted by $\mathcal{O}(p^2)$ operations. [14] For the $\mathcal{O}(N)$ boxes in the tree evaluating the potential experienced with a box is also $\mathcal{O}(Np^2)$ as either there are a constant number of interacting boxes or the potential is passed down from a parent and conversion of the multipole expansion is of order $\mathcal{O}(p^2)$. Finally, the direct computation at leaf level for the nearest remains unchanged and of $\mathcal{O}(N)$ as there a constant number of particles in each box, and actual computation of the potential for each particle from the coefficients is $\mathcal{O}(p)$. [14] This results in a overall complexity of $\mathcal{O}(Np^2)$, however by choosing the number of bodies in the leaf boxes to be proportional to the number of terms in the expansion, the expression for the operation count can additionally be simplified to approximately $40Np$.[20]

Comparison to A Competing Algorithm: Barnes Hut. The barnes hut algorithm also make use of the notion of well distanced particles, and subsequent approximation of the contributions of the force contributions of said particles, as well as hierarchical tree decomposition of particles, as central mechanisms leading to its improved asymptotic complexity and runtime for large scale systems. However, the first noticeable difference from the Fast Multipole Method is in the construction of the tree. As stated previously, and again in the setting of a roughly uniform particle distribution, in the fast Multipole Method a tree of specified fixed depth is constructed prior to any computation and that same tree is normally used for subsequent timesteps, with leaves of the tree being properly updated with particles at each timestep. However, in the Barnes-Hut Algorithm, the tree is typically reconstructed at each timestep, with the divisions in the tree being created with the insertion of each particle. Namely, each box in the tree can have at most one particle, so if a particle is loaded into a box which has no further children and already contains an existing particle, then the box is recursively split into four children (considering a two-dimensional example, eight children in three dimensions), until the existing particle and newly introduced particle are placed into separate boxes. Thus the distribution of the particles much more directly influences the tree decomposition, rarely leading to a fully expanded tree in practice, though with a uniform distribution the height can be bounded as O(log(N)), and resulting in a total tree construction time of O(Nlog(N)). [10] Next, in contrast to the expansion used in the Fast Multipole Method, the boxes are associated with information relating to the total mass and center of mass of all of the particles within them, effectively creating pseudo-particles. This is accomplished by propagating up the tree from child boxes total and center of mass information to the parents, which assuming a tree height of $\mathcal{O}(log(N))$ is $\mathcal{O}(Nlog(N))$ in complexity. [10] After the total mass and center of mass for each nonempty box has been determined, the approximation based on well-distanced particles comes into play, traversing the tree once for each particle to determine all of the force contributions from other particles. [4] A particle is considered well-distanced from a box if the ratio of the box width over the distance from the particle to the box's center of mess is less than a fixed parameter, normally denoted as θ. [10] Then beginning from the root node, dependent on this parameter θ and the mentioned ratio determined for each individual particle and a given box, either the center of mass and total mass are used as a pseudo particle approximation, or each of the children of the box are recursively visited, dependent on whether the ratio is less than or greater than θ, respectively. [4] Lastly, examining the complexity of determining the force contribution of all other particles for each given particle, for a θ value nominally set close to 1, each particle effectively interacts with $\mathcal{O}(log(N))$ other real or pseudo-particles, resulting in a total asymptotic complexity of $\mathcal{O}(Nlog(N))$. However, a major disadvantage in comparison to the Fast Multipole Method is the inability to approach machine precision while retaining a runtime of $\mathcal{O}(Nlog(N))$. It has been empirically found that the accuracy is often within one percent, which often also dictates its use case when such a condition is sufficient, however the accuracy has also been shown to be potentially unbounded for $\theta > \frac{1}{\sqrt{3}}$, and in fact the worst case relative errors are only guaranteed to be within 5 percent of the accurate solution for $\theta < 0.25$, which ends up drastically increasing the constant factor multiplying the number of computations as the number of computed interactions is proportional to θ^{-3}. [10][9] More expansion terms are often use to help improve this error in addition to the total mass and center of mass in later variants, though this is not a component of the original Barnes-Hut algorithm.

Implementation Discussion. The multipole computation after the quadtree has been filled with points was optimized to run serially in just about 3.5 ms for 1000 particles of varying mass with p = 33 on a Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz processor allowing for very responsive real time simulation of 1000 moving bodies. Though the precise error and performance characteristics are shown shortly below, a simulation file is additionally provided in the code base to intuitively and visually demonstrate scalability to large numbers of particles achievable with the Fast Multipole Method, and that for evenly high p values such as 33 the large constant factor necessary for good error bounds certainly does not immediately throw the FMM out of the running for smaller to mid-sized problems. P = 33 is chosen as an example here because larger numbers of coefficients led to larger binomial coefficients in the formation of the expansions which occupied more bits and were not yet completely optimized for, leading to an unfortunate noticeable constant multiplier degradation in performance, around 2-3x. The majority of the speedups within the multipole code were achieved by taking advantage of cache locality as much as possible and reusing the same arrays as much as possible, with many portions also preallocating large portions of temporary space and obtaining views into the arrays. The second most important factor was the precomputation and storage of binomial coefficients which after profiling revealed led to even double digit multipliers in terms of speedups on portions of the code. Finally, the third most important factor, and as hinted in the complexity of the shift operations for the expansions, there was a noticeable difference in performance when retaining a temporary for a term raised to the kth power when it is known that the $k+1$th power must be computed next, as the power operation is more expensive than a simple addition or multiplication.

On a separate note, a substantial portion of the Barnes Hut algorithm in ParallelBarnesHut.jl had to be rewritten as well before performing comparisons as it was by default implemented in three dimensions and as a result using octtrees instead of quadtrees which would extremely likely skew the flop count when performing a comparison. Unfortunately, the motivation for change also partially came from a point of correctness, as it was discovered that the default package introduced significant errors on every simulation by forgetting to handle boxes that were within the threshold ratio θ but only had one particle and as such required direct computation. Furthermore, before proceeding the numerical results it should be stated that although no altercations were made to the numerically sensitive portions of the code as that is essential to the guarantees and accuracy of the method, for simplicity of simulation a small region of space was densely packed with particles in order to better guarantee uniform distribution within the region, along with the liberty of disallowing particles from exiting the prescribed region by means of threshold position at the boundaries. This is of course damages the accuracy of the simulation as particles may normally exit the region, but is acceptable for the purposes of this paper which is to demonstrate the potential performance gains and retained accuracy when using the hierarchical decompositions and expansions of the fast multiple method for high numbers of particles. Additionally, when filling the quadtree with points (a separate component from the expansion computation and manipulation), after noticing that on the previously described computer realistically only low depth trees of 3 or 4 were performant for medium scale simulations, the quadtree point filling algorithm was experimentally modified to an algorithm based on the Dutch National Flag problem of color sorting with runtime $\mathcal{O}(NlogN)$, as it is an in place algorithm and therefore enjoys better locality. However, the O(N) method should ideally be re implemented for massive scale systems, and which can easily done by diving a point in space by the leaf box spacing and finding the nearest leaf box center. However, emphasis should again be made that this was largely explored for simplicity and potential performance benefits for mid scale problems that could be seen on a personal laptop computer, and that this is a minor portion of the full algorithm especially in comparison to the numerically oriented components.

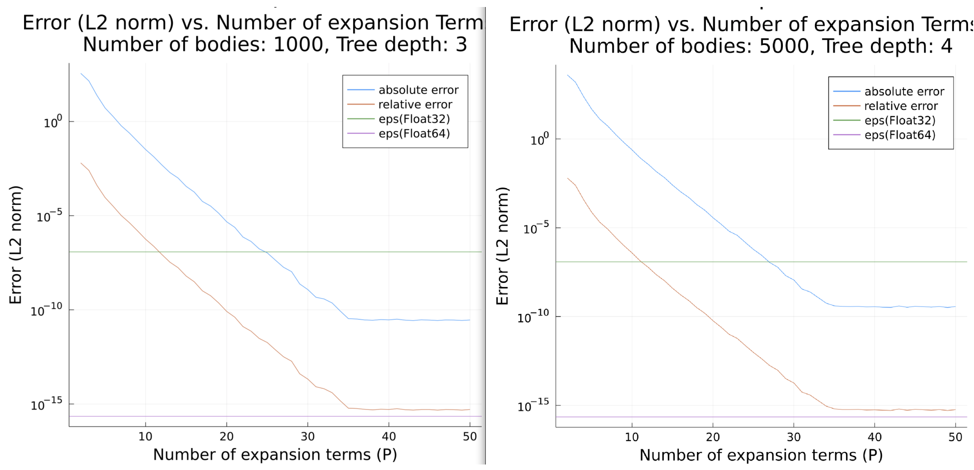


Plots of error for a force calculation spanning a single timestep using the Barnes Hut algorithm while varying θ. On the left, 1000 randomly initialized bodies are used, while on the right 5000 are used.

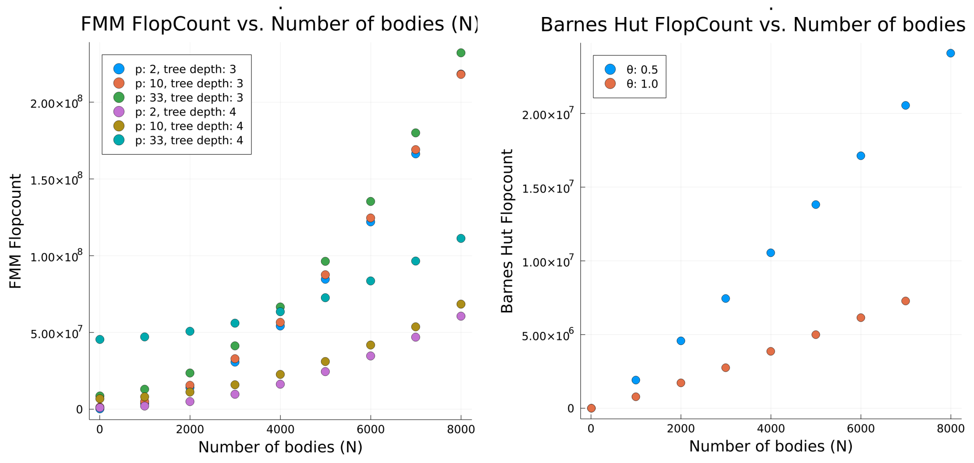
Results. Error in the Barnes Hut algorithm is highly dependent on the MAC criterion, θ, where low values of θ achieve better accuracy but by construction approach simply doing direct $\mathcal{O}(N^2)$ computation. This demonstrates that in the Barnes Hut algorithm there is often an inherent trade off of complexity against accuracy, meaning that for a fast and performant $\mathcal{O}(NlogN)$ computation, significant detriment to accuracy cannot be necessarily avoided.

On the other hand, error when calculating the forces acting on particles when using the Fast Multipole Method can be observed to be decreasing with increasing number of coefficients following the theoretical derivation. Machine epsilon for Float32 and Float64 are in fact reached before the predicted coefficient numbers of 24 and 53, respectively, though that is potentially a consequence of small floating point error in the benchmark computation. From this perspective, the Fast Multipole Method is far superior to Barnes Hut and the Fast Multipole Method should definitely be preferred when accuracy is of paramount importance.

When looking at next criterion, performance, the Fast Multipole Method implementation has been optimized considerably but the Barnes Hut implementation has not, so a proxy for runtime in the form flop counts is used. This is not entirely ideal as memory access patterns of a program are often critical to its performance, but without prior in depth knowledge of the extent to which the memory access in both programs can be optimized flop counts is a suitable indicator for overall performance.



Plots of error for a force calculation spanning a single timestep using the Fast Multipole Method while increasing number of expansion terms p. It can be observed that the error still approaches machine epsilon independently of the increased tree depth or number of bodies.



Left figure: Flop count with increasing number of bodies for different numbers of expansion coefficients p and tree depth. Right figure: Flop count with increasing number of bodies for two different θ values.

In the final figures, presenting flop counts for varying parametrized FMM executions and different parameters θ in the Barnes Hut algorithm, it can be seen that for low numbers of bodies N, higher tree depths, and greater number of coefficients p, the added constant complexity of the FMM appears to potentially greatly increase the overall execution time of the FMM program leading to potentially much slower execution then as seen in Barnes Hut. The exact intersect point was not located due to the long runtime of the unoptimized Barnes Hut code for large N, but a trend can be observed with the decreasing slopes of FMM executions with greater depth, which suggests that the crossover with a slighly more conservative value of θ such as 0.5 is extremely likely to take place within an order of magnitude, though the exact points strongly depend on the exact values p, and the tree depth.

REFERENCES

[1] Alexander T. Ihler, *An Overview of Fast Multipole Methods*, 2004, https://www.ics.uci.edu/~ihler/papers/ihler_area.pdf (accessed 2020/05/19).

[2] Andrew W. Appel, *An Efficient Program for Many-Body Simulation*, SIAM Journal on Scientific and Statistical Computing, 6 (1985), pp. 85–103, https://doi.org/10.1137/0906008.

[3] Barry A. Cipra, *The Best of the 20th Century: Editors Name Top 10 Algorithms*, SIAM News, 33 (2000).

[4] G. Blelloch and G. Narlikar, *A practical comparison of n-body algorithms*, 1997.

[5] Douglas C. Heggie, *The Classical Gravitational N-Body Problem*, 26 (2005), https://doi.org/arXiv:astro-ph/0503600.

[6] Ernst Hairer, Christian Lubich, Gerhard Wanner, *Geometric numerical integration illustrated by the St¨ormer–Verlet method*, Acta Numerica, (2003), pp. 399–450, https://doi.org/10.1017/S0962492902000144.

[7] Hanliang Guo, *A Crash Course on Fast Multipole Method (FMM)*, 2018, http://www-personal.umich.edu/~hanliang/publications/FMM_Tutorial_Hanliang.pdf (accessed 2021/05/01).

[8] J. Carrier, L. Greengard, and V. Rokhlin, *A Fast Adaptive Multipole Algorithm for Particle Simulations*, SIAM Journal on Scientific and Statistical Computing, 6 (1988), pp. 669–686, https://doi.org/10.1137/0909044.

[9] John K.Salmon, Michael S.Warren, *Skeletons from the Treecode Closet*, Journal of Computational Physics, 111 (1994), pp. 136–155, https://doi.org/10.1006/jcph.1994.1050.

[10] Josh Barnes Piet Hut , *A hierarchical O(N log N) force-calculation algorithm*, Nature, 324 (1986), p. 446–449, https://doi.org/10.1038/324446a0.

[11] Ken-ichi Yoshida, *Applications of Fast Multipole Method to Boundary Integral Equation Method*, 2001, http://gspsun1.gee.kyoto-u.ac.jp/yoshida/doctoral_thesis/dthesis.pdf (accessed 2021/05/10).

[12] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the laplace equation in three dimensions*, Acta Numerica, 6 (1997), pp. 229–269.

[13] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura, *Accelerating fast multipole methods for the Helmholtz equation at low frequencies*, IEEE Comp. Sci. Eng., 5 (1988), pp. 32–38, https://doi.org/10.1109/99.714591.

[14] L Greengard, V Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, (1987), pp. 325–348, https://doi.org/10.1016/0021-9991(87)90140-9.

[15] L. Greengard, W.D.Gropp, *A parallel version of the fast multipole method*, Computers Mathematics with Applications, 20 (1990), pp. 63–71, https://doi.org/10.1016/0898-1221(90)90349-O.

[16] Lexing Ying, *A pedestrian introduction to fast multipole methods*, Science China Mathematics, 55 (2012), p. 1043–1051, https://doi.org/10.1007/s11425-012-4392-0.

[17] Lexing Ying, George Biros, Denis Zorin, Harper Langston, *A New Parallel Kernel-Independent Fast Multipole Method*, Proceedings of the 2003 ACM/IEEE conference on Supercomputing, (2003), https://doi.org/10.1145/1048935.1050165.

[18] Nader Engheta, William D. Murphy, Vladimir Rokhlin, and Marius Vassiliou, *The Fast Multipole Method for Electromagnetic Scattering Computation*, IEEE Transactions on Antennas and Propagation, 40 (1992), pp. 634–641.

[19] R. Yokota, L. A. Barba, T. Narumi, K. Yasuoka, *Petascale turbulence simulation using a highly parallel fast multipole method on GPUs*, Computer Physics Communications, 184 (2013), pp. 445–455, https://doi.org/10.1016/j.cpc.2012.09.011.

[20] Rick Beatson, Leslie Greengard, *A short course on fast multipole methods*, 1997, https://math.nyu.edu/~greengar/shortcourse_fmm.pdf (accessed 2021/5/01).

[21] Stanford CME342 Lecture, *Fast methods for N-body problems*, 2014, http://adl.stanford.edu/cme342/Lecture_Notes_files/lecture13-14_1.pdf (accessed 2020/05/11).

[22] Tancred Lindholm, *Seminar presentation N-body algorithms*, 1999, http://www.cs.hut.fi/~ctl/NBody.pdf (accessed 2020/05/18).

[23] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, Comp. Phys., 60 (1985), pp. 187–207, https://doi.org/10.1016/0021-9991(85)90002-6.