An Object-Oriented Parallel Algorithm for the Newtonian N-Body Problem

Masahiro Yasugi Akinori Yonezawa

Abstract

A fast parallel object-oriented numerical algorithm which computes the positions and velocities of a large number of particles (masses) interacting via a long range force is presented. The time-complexity of the algorithm is less than $O(\log N)$ using $O(N)$ concurrent objects. The algorithm is a parallel extension of the Barnes and Hut algorithm[3].

1 Introduction

The Newtonian N-body problem is to find the behavior of N particles interacting with each other through a long range force such as gravity or Coulombic force. Efficient methods for solving this problem provide powerful research tools in physics and astronomy. Since a naive (exact) simulation needs to compute the interactions of all the pairs of particles, its time complexity, which is $O(N^2)$, becomes prohibitive as N increases. Thus, methods have been developed to reduce the computational cost by approximation. One such method was developed by J. Barnes and P. Hut[3]. The time complexity of their method is $O(N \log N)$ and its rigorous upper bound of errors can be estimated. Another approach has been taken by L. Greengard [4], who used the multipole expansion method. This paper presents a new parallel algorithm for the Newtonian N body problem, which is inspired by the method of Barnes and Hut. Our algorithm is based on the use of a large number of concurrent objects[8, 1, 2, 9]. The time complexity of this algorithm is $O(\log N)$ using $O(N)$ concurrent objects. This algorithm exploits the characteristic nature of object-oriented concurrency[8, 1, 2, 9] using dynamic concurrent object creation and the dynamic reconfiguration of the message passing topologies.

2 Serial Algorithm (Barnes and Hut)

The essence of Barnes and Hut's algorithm[3] is the use of a tree structure, which corresponds to the spatial distribution of particles. Nodes in the tree structure correspond to cells (or subspaces) which are obtained by dividing the entire space containing all the particles in the following manner. The entire space is recursively divided into cells with child cells having exactly half the length, breadth and width of their parent. Cells containing just one particle (or none at all) are not further subdivided. Figure 1(a) shows such a division of space in the two dimensional case. Dots denote particles. A cell containing

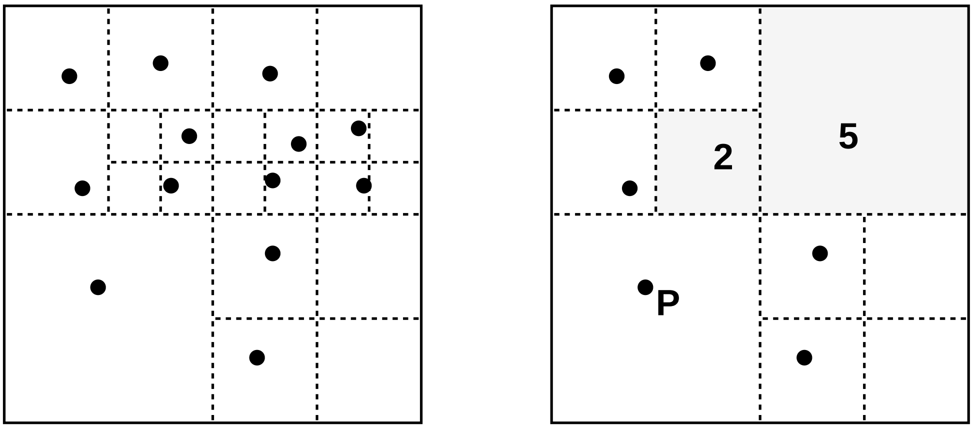


Figure 1: (a) Entire Space is Divided Recursively. (b) Virtual Particles(gray zones)

no particles has no corresponding node in the tree structure. A cell containing just one particle has a single corresponding node which is a terminal node in the tree structure. The root node corresponds to the entire space and non-terminal nodes correspond to cells containing at least two particles.

Having such a tree structure, we assume that each node in the tree structure keeps the total mass and the center-of-gravity of the particles contained in the corresponding cell. Thus, a non-terminal node can be viewed as a terminal node for a cell which contains a single *virtual* particle. The virtual particle's mass and position are the total mass and center-of-gravity, respectively, of the particles actually contained in the cell. In Figure 1(b), a number in a cell stands for a virtual particle and indicates the number of actual particles represented.

If a particle P whose distance from the center-of-gravity is sufficiently larger than the diameter of the cell (i.e., the longest distance between any pair of particles in the cell), the total force exerted on P by the particles in the cell does not have to be computed as the sum of the individual forces exerted by the particles in the cell. The total force can be estimated within a required error bound as the force exerted by the virtual particle. Whether or not the approximation technique is employed is determined by a threshold value θ derived from the specified error bound. Letting the diameter of a cell be s, and the distance between P and the position of the virtual particle be d, if the inequality

$$s/d < \theta$$

holds, approximation is permissible. Figure 1(b) shows how this approximation is done. The virtual particles gray-zoned in Figure 1(b) are used when the force exerted on P is estimated. The analysis by Barnes and Hut[3] shows that the time required to compute the total force exerted on a single particle by all other particles in the space is $O(\log N)$. Thus, to compute the force exerted on N particles requires $O(N \log N)$ time.

3 Overview of Our Parallel Algorithm

An overview of our parallel algorithm is given as follows:

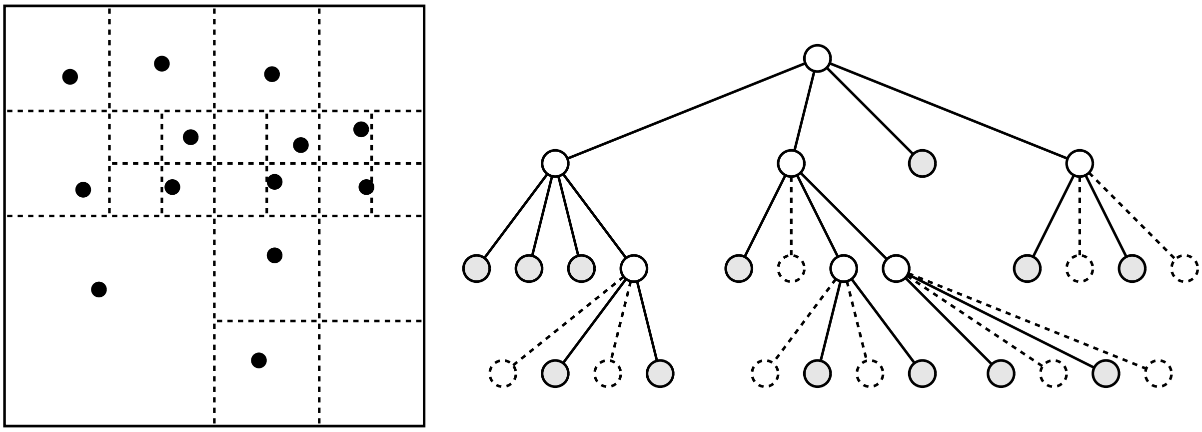


Figure 2: Tree Structure of Cell Objects

1. For each node in the tree structure of Barnes and Hut's algorithm, there is a concurrent object, called a *cell object*, which keeps the physical information such as the number of the particles in the corresponding cell, their total mass, and their center-of-gravity.

2. Cell objects form a tree structure isomorphic to that of Barnes and Hut's algorithm. (See Figure 2.) Thus each cell object also keeps the structural information (of the corresponding tree structure) such as the names of its parent cell objects, child cell objects (and sibling cell objects if necessary). This tree structure is constructed by performing parallel *summation* operations on tree structures, which will be explained in Section 5

3. The *terminal* cell object residing at a terminal node of the tree structure represents a single particle in the system. The non-terminal cell object residing at a non-terminal node of the tree structure represents a virtual particle or a cell containing more than one particle.

4. The total force exerted on a particle is calculated by its corresponding terminal cell object by summing up forces. Each force is derived by using the mass and distance of a non-negligible (virtual) particle which are contained in a message sent to the terminal cell object from the cell object representing a (virtual) particle. The range of the message distribution and how messages are distributed will be described in Section 4.

5. Such messages are sent concurrently to different terminal cell objects. Thus a very high degree of concurrency is obtained.

6. At the beginning of each time interval, a new tree structure composed of cell objects which reflect the updated values of particle's velocity and position is constructed.

Keeping this general picture of our parallel algorithm in mind, let us go into details.

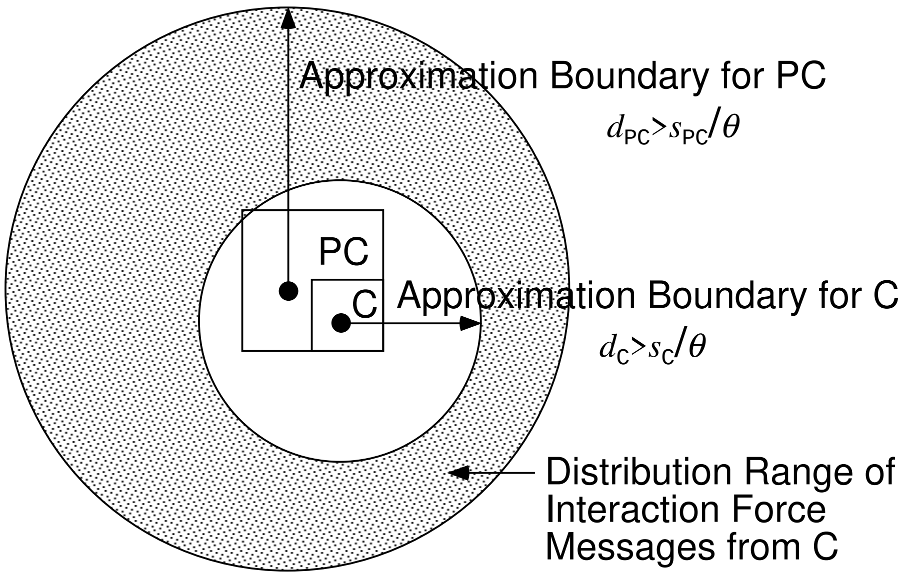


Figure 3: The range of the distribution of interaction-force messages (2-D case)

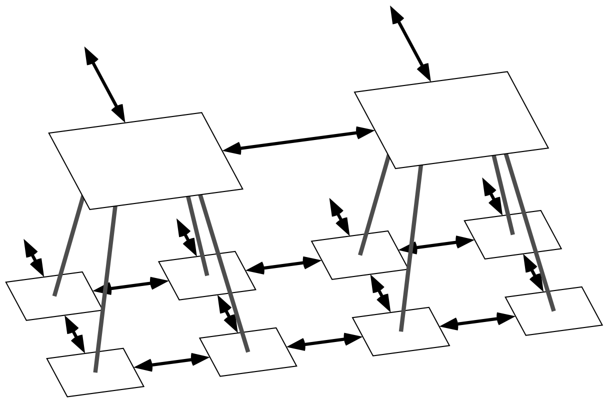
4 Parallel Calculation of Forces

We assume that each cell object of the tree keeps the following data about the cell (subspace) that it represents:

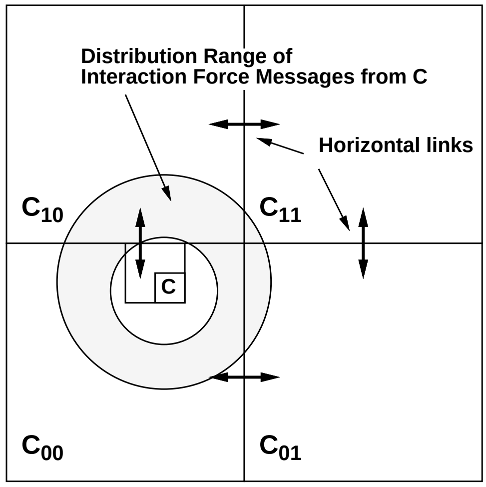
1. the size of the cell,
2. the position of the center of the cell,
3. the center-of-gravity of the particles contained in the cell,
4. the number of the particles contained in the cell,
5. the total mass of the particles contained in the cell,
6. the structural information such as who are its parent, children, etc.

As described in the previous section, to calculate the total force exerted on a particle P within an error bound, the terminal cell object TC representing P must know the mass, and position of all non-negligible (virtual) particles. Messages containing this information, which we call *interaction-force messages* or IFMs, are transmitted from the cell objects representing the non-negligible (virtual) particles. Upon receiving a relevant IFM, the terminal cell object TC calculates the force exerted by the corresponding (virtual) particle. This calculation continues until no more relevant IFMs are received. Exhaustion of the relevant IFMs can be easily detected by counting the particles represented in the IFMs.

What needs to be more precise about is the range of the distribution of IFMs and how they are distributed. First, let us consider the range of the distribution. The set



(a) Horizontal Links



(b) Use of the horizontal links

Figure 4: Horizontal links between adjacent cell objects of the same level (2-D case)

of terminal cell objects that need to be sent an IFM from a cell object C is the set of terminal cell objects whose particles are "in range" of the (virtual) particle represented by C. Note that two IFMs, one sent by a cell object and the other sent by the parent of the cell object, may reach the same terminal cell object. In such cases, the terminal cell object may include the force of the same particle more than once. This must be avoided. The distribution area of an IFM from C is the set of points for which approximation relative to C is possible and for which approximation relative to PC, the parent of C, is not possible. This area is illustrated as the gray zone in Figure 3.

How is the distribution of an IFM done? An IFM transmitted from C first travels upward to its PC. Then, the PC adds information about its approximation boundary to the IFM and launches it into the network of cell objects. Seemingly, the routing of IFM can be performed efficiently by cell objects that have links to their parent and children. However, it should be noted that all the non-terminal cell objects concurrently transmit IFMs. Thus a single cell object may simultaneously receive many IFMs originated from different cell objects causing a message processing bottleneck, especially near the root cell object. To alleviate this problem, we put *horizontal* links between adjacent cell objects of the same level (i.e., which are also of the same size) if adjacent ones exist. See Figure 4(a).

Such horizontal links allow interaction-force messages to bypass parent cell objects, reducing the congestion on vertical links. We can distribute an IFM in the following manner:

1. The IFM is repeatedly forwarded upward to the first cell object whose cell size is larger than the size of the distribution range of the IFM (C_{00} in Figure 4(b)).

2. Along the horizontal links, the IFM is distributed among the cell objects whose corresponding cells cover the distribution range (C_{00}, C_{01}, C_{10} and C_{11} in Figure 4(b)).

3. Each distributed IFM is recursively forwarded downward to the cell objects whose corresponding cells intersect the range of the distribution.

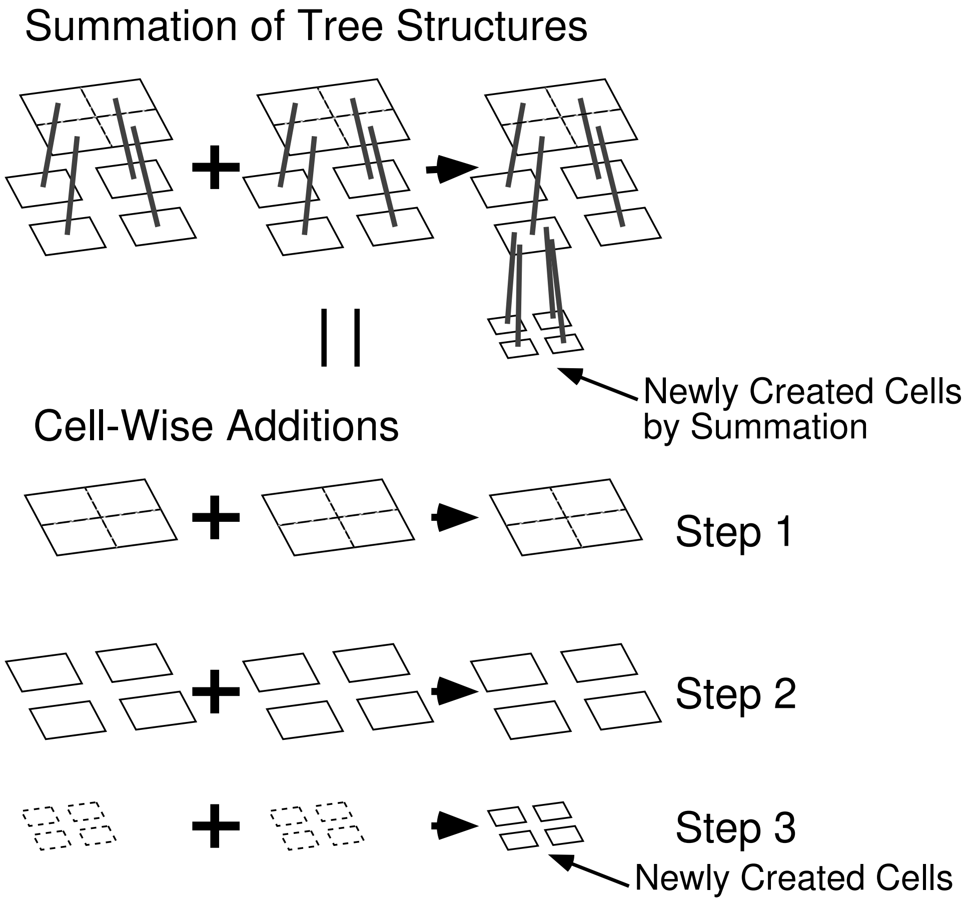


Figure 5: Summation of tree structures

Our analysis shows that this eliminates the bottlenecks[1]. Having no bottlenecks allows us to estimate the upper bound of the time complexity of the parallel force calculation of this section as the total length of path along which an IFM travels. This is at most $O(\log N)$. Furthermore, it can be easily shown the construction of such links requires only $O(\log N)$ time complexity, assuming that the tree structure already exists.

5 Parallel Tree Construction

In order to achieve $O(\log N)$ time complexity for our entire algorithm, we must be able to construct the tree structure in $O(\log N)$ time.

5.1 Summation of Tree Structures

We devised a commutative operation called *summation* for tree structures, which is used in our parallel tree construction algorithm. Let A and B be tree structures of cell objects

[1]By a bottleneck, we mean that more than constant time is required for a message to be processed after arriving at a concurrent object.

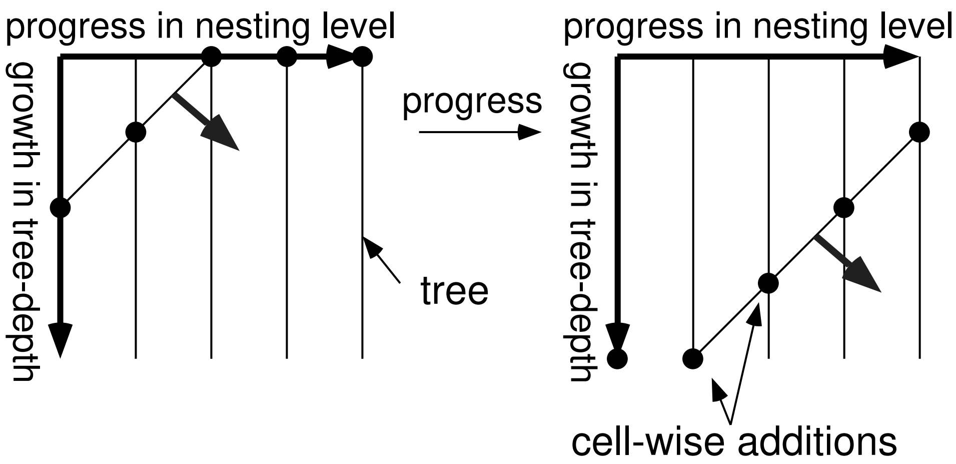


Figure 6: Pipelined Parallel Summation of tree structures

whose roots represent the same space. We denote the tree structure obtained by the summation operation of A and B by

$$A + B$$

The root of $A+B$ represents the same space represented by the root of A and/or B and $A+B$ contains the union of the sets of particles contained in A and B. Of course, $A+B$ has the same structural property explained in Section 2.

Given two tree structures A and B which represent the same space, how do we obtain $A+B$? First, the data to be kept in the root cell object of $A+B$ is computed by using the total masses, the centers-of-gravity and the total numbers of particles kept in the root cell objects of A and B. Then, the data kept in each child of $A+B$ is computed by using the data kept in the corresponding children of A and B. (See Figure 5.) This process of *cell-wise additions* is repeated until all the terminal nodes of $A+B$ are reached. Note that the total number of cell objects in $A+B$ may be greater than that of A and B. Furthermore, the depth of $A+B$ may be greater than that of A and B. (Note Step 3 in Figure 5.)

Since the summation operation on tree structures is performed from the root towards their leaves sequentially, its time complexity is the depth of the resulting tree structure. In the case where the depth of tree structure is $O(\log N)$, the time complexity is $O(\log N)$,

5.2 Tree Construction by Pipelined Parallel Summations

Now we consider how a tree structure is created from the initial set of data about the particles. Suppose we have particles a, b, c, d, e, f, g and h. Correspondingly, we create cell objects A, B, C, D, E, F, G, and H, all of which we regard as (singleton root-only) tree structures. Instead of adding each tree structure one by one, the following nested summation is performed to obtain a tree structure T containing all the particles a through h.

$$T = (((A + B) + (C + D)) + ((E + F) + (G + H)))$$

Note that the summation operations at a same level of nesting are performed in parallel. Thus, $(A+B)$, $(C+D)$, $(E+F)$, and $(G+H)$ are performed in parallel. Since the number of nesting levels is bounded by $O(\log N)$ and the each individual summation operations requires $O(\log N)$ time at most (as noted at the end of the previous subsection), the total construction of T requires $O((\log N)^2)$ time at worst. Unfortunately, our goal is $O(\log N)$ not $O((\log N)^2)$.

The reason the construction of T requires $O((\log N)^2)$ is the fact that a summation operation of two trees at a given nesting level is performed after completing the construction of the two trees. To achieve our goal of attaining $O(\log N)$ time complexity, we can make the summation operation of two trees at a given nesting level and the construction of the two trees overlap in time. In other words, the summations are pipelined over nesting levels. This pipelining is illustrated in Figure 6 where the vertical coordinate represents the growth (construction) of individual trees and the horizontal coordinate represents the progress in the nested summation operation. The circles on a diagonal line represent the cell-wise additions in the summation operations being performed in parallel. For example, the cell-wise addition of the third level (from the root) cell objects of A and B, the cell-wise addition of the second level cell objects of $A+B$ and $C+D$, and the cell-wise addition of the root cell objects of $(A+B)+(C+D)$ and $(E+F)+(G+H)$ are performed in parallel. Now it is easily seen that it takes $O(\log N)$ time for the diagonal line to finish sweeping the whole area.

6 Conclusion

We have presented a $O(\log N)$ parallel algorithm for the Newtonian N-body problem using $O(N)$ concurrent objects. The main techniques devised are: (1) elimination of message bottlenecks by originating *interaction-force messages* from non-terminal cell objects instead of terminal cell objects, (2) elimination of message bottlenecks by routing through *horizontal* links connecting adjacent cell objects of the same level, (3) employing *summation* operations for tree structures, (4) tree construction by *pipelined* parallel summations.

We are currently conducting performance evaluation of this algorithm using two implementations of our language ABCL/1 on EM-4[6] and AP1000[5]

Acknowledgements

The work was initiated by Prof. P. S. Krishnaprasad who hosted the second author's visit to Systems Research Center, University of Maryland during the summer of 1988. We would like to express our deep appreciation of Krishna's pioneering sense in realizing the significance of object-based concurrent computation in the N-body problem. We would like to thank Jeff McAffer for his comments which substantially improved the presentation of this paper.

References

[1] G. Agha: *Actors: A Model of Concurrent Computation in Distributed Systems*, The MIT Press, 1986.

[2] G. Agha: *Concurrent Object-Oriented Programming*, CACM, October, 1990.

[3] J. Barnes and P. Hut: "A Hierarchical $O(N \log N)$ Force-Calculating Algorithm," Nature, 324:446-449, 1986.

[4] L. Greengard: *The Rapid Evaluation of Potential Fields in Particle Systems*, ACM Press, 1987.

[5] K. Taura, S. Matsuoka, and A. Yonezawa: "An Efficient Implementation of Object-Oriented Concurrent Language on Multicomputer," Proc. Joint Symposium on Parallel Processing, 1992

[6] M. Yasugi, S. Matsuoka, and A. Yonezawa: "ABCL/onEM-4: A New Software/Hardware Architecture for Object-Oriented Concurrent Computing on an Extended Dataflow Supercomputer," Proc. 6th ACM International Conference on Supercomputing, 1992.

[7] M. Yasugi, and A. Yonezawa: "An Object-Oriented Concurrent Algorithm for N-Body Problem," Proc. 8th Annual Convention of Japanese Society of Software Science and Technology, 1991.

[8] A. Yonezawa (Ed.): *ABCL: An Object-Oriented Concurrent System* The MIT Press, 1990.

[9] A. Yonezawa, and M. Tokoro (EDs.): *Object-Oriented Concurrent Programming*, The MIT Press, 1987.