Introducing Nested Data Parallel in Barnes Hut Algorithm

Mónica Fuentes, Fabiana Piccoli, Marcela Printista *
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950
++54-2652-430224
5700 - San Luis
Argentina
e-mail: {gfuentes, mpiccoli, mprinti}@unsl.edu.ar

Abstract

The N-body problem is often characterized by the necessity to interact each of the N bodies with every other one. Its principal problem is the time spent in force computation.

Computing the force among a set of N bodies can be done in a straightforward way by computing all N^2 pair wise interactions. However, a number of more efficient algorithms have been proposed, these can approximate the forces among N bodies in close to linear time. The Barnes-Hut algorithm is one these.

Barnes-Hut algorithm is suitable to be resolve in parallel. Several parallel challenges have been done, most of them for shared memory machine.

In this paper, we describe the design of portable and efficient parallel implementation of adaptive N-body method: Barnes-Hut algorithm. Our propose is based on a regular communication pattern and work partitioning scheme that allows to apply nested data parallelism and to obtain a portable solution.

Finally, our aim is not simply to develop an efficient implementation of one algorithm, but show how a programming model can be applied in problems not suitable to it, in first instance.

Keywords: N-body problems, parallel programming models, communication pattern, data partition, balanced workload.

*Group supported by the UNSL and ANPCYT (Agencia Nacional para la Promoción de la Ciencia y Tecnología)

1 Introduction

The N-body problem is the problem of simulating the movement of a set of bodies (or particles) under the influence of gravitational, electrostatic, or other type of force. N-body simulations have a number of important applications in fields such as astrophysics, molecular dynamics, fluid dynamics, and even computer graphics. Its objective is to find the positions and movements of the bodies in space that are subject to gravitational forces from others bodies using Newtonian laws of physics:

$$\frac{\delta^2 \vec{x}_1}{\delta t^2} = \sum_{j \neq i}^{N} \vec{a}_{ij} = \sum_{j \neq i} -\frac{G m_j \vec{d}_{ij}}{\mid d_{ij} \mid^3}$$

where

$$\vec{d}_{ij} \equiv \vec{x}_i - \vec{x}_j$$

A large number of algorithms for N-body simulations have been proposed; a basic approach taken by most of these algorithms is to simulate the system by advancing the bodies in discrete time steps. In each time step, the algorithm computes (or approximates) the force exerted on each body due to all other bodies; this determines the acceleration and speed of that body during the next time step. Computing the forces among a set of N bodies can be done in a straightforward way by computing all N^2 pairwise interactions. This solution implies an $O(N^2)$ set of operation. Its direct implementation is a trivial programming exercise. It is simply a double loop which vectorizes very well. At discrete time intervals, the algorithm computes the forces on the bodies and adjust their velocities and positions:

```
While time≤ t_end do
   Accumulate forces by finding the force f(i,j) of particle pair i,j
   Integrate the equations of motion  by updating the position
   Update time counter
```

Such as we say above, unfortunately it has an asymptotic time complexity. When there are large number of bodies, the evaluation of the force function can consume too much time. If we have N bodies, then there will be N evaluations of the force function, each of which will have $N - 1$ terms that involve expensive operations.

In ours experiments with a system of 256 bodies, the evaluation of the force function will account for well over 95% of the CPU time used. The integration required for to compute new_position and all the other housekeeping taking up around 5%. For other N-body programs that have to evaluate thousands or even millions of bodies, this method of evaluating the force function is almost completely useless.

Because the efficient evaluation of the force function is so critical for most N-body programs, this has been a very active area of research in the last years. However, a number of more efficient algorithms have been proposed that can approximate the forces among N bodies in close to linear time.

The remainder of this paper is organized as follows. The next section gives a description of the Barnes-Hut algorithm used in our implementations. Section 3 explains the parallel programming model: nested data parallellism. Section 4 discusses some issues related to this work. Section 5 contains a description of our implementation. Finally, Section 6 offers the conclusions and future works.

2 Barnes-Hut algorithm

The trivial N-body cost, $O(N^2)$, can decrease by taking advantage of locality in the space-partitioning: *cluster of bodies*, that is, groups of bodies that are fairly close to each other [13]. This is possible by the principle: *"The effect of a cluster of bodies at a distant point can be approximated by a small number of initial terms of an appropriate power series"*, see Figure 1.

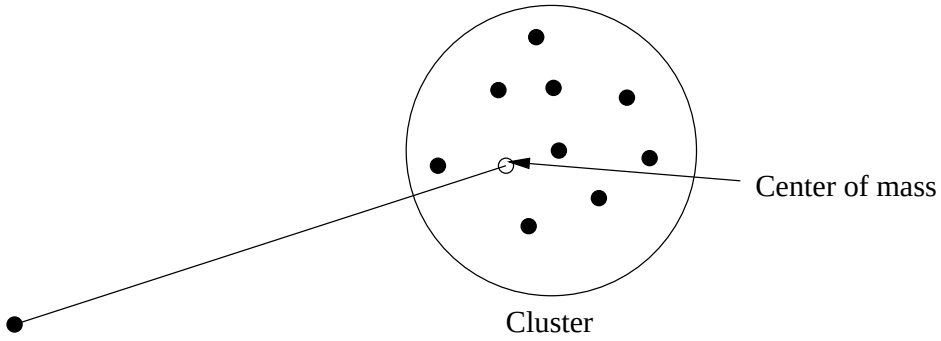

Figure 1: Cluster and Center of Mass

An important class of N-body algorithms applies this idea and uses a tree data structure to hierarchically group the bodies into cluster. These methods can calculate the forces among N bodies in to approximately $O(N \log N)$ or even $O(N)$ [1]. They are classified in adaptive or non-adaptive algorithm. The difference is in how they divide the space, according to the input distribution (adaptive) or fixed size (non adaptive).

The Barnes-Hut algorithm is adaptive tree-based method whit cost $O(N \log N)$.

The next psudecode shows the Burnes-Hut algorithm. The algorithm has four basic steps in each time step.

```
for each time step:
   1. Build the BH-tree
   2. Compute centers-of-mass bottom-up
   3. For each body
          Start a depth-first traversal of the tree,
          Truncating the search at internal nodes where the
          approximation is applicable;
          Update the contribution of the node to the acceleration of the body
   4. Update the velocity and position of each body
```

Figure 2: The Barnes-Hut algorithm

The first step is a tree-build phase where a tree(BH-tree) is created from the bodies in the space. The next phase is to calculate the centers-of-mass and total mass of each internal node of the tree. The third phase is the force computation phase, where some body-to-body interaction rule is used to calculate the effects of each particle on the others. And finally, an updating phase where the effects of interactions are propagated to every bodies.

The Barnes-Hut algorithm organizes the set of bodies into a hierarchy of clusters. To minimize the number of operations, each body computes interactions with the largest clusters

for which the approximation can be applied. As shown in Figure 2, the algorithm first computes an tree partition of the region of space enclosing the set of bodies. The partition is computed recursively by dividing the original box into eight or four sub-box of equal volume until each undivided box contains exactly one body. The Figure 3 shows the partition of space (four sub-box) and the correspond BH-tree (quad-tree).

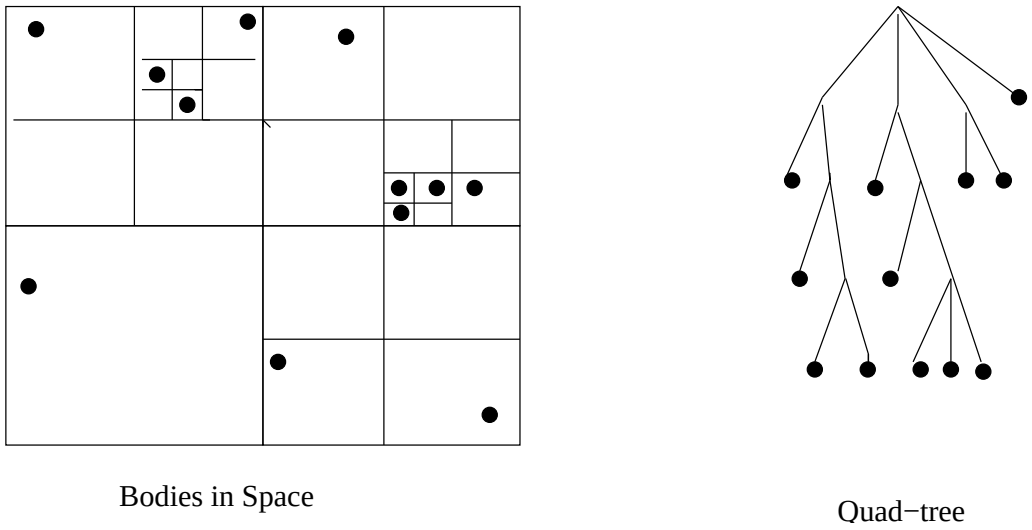

Figure 3: Plane Division and its Quad-BH-tree

The sequential Barnes-Hut algorithm constructs the BH-tree by inserting bodies into the cluster hierarchy one at a time. The i^{th} body is added into the BH-tree consisting of the first $i-1$ bodies. The body descends down the BH-tree until it is alone in its box.

Each internal node of the BH-tree represents a cluster. Once the BH-tree has been built, Step 1, the centers-of-mass of the internal nodes are computed in one phase up the tree, starting at the leaves, Step 2. Step 3 computes accelerations; each body traverses the tree in depth-first manner starting at the root. For any internal node sufficiently far away, the effect of the subtree on the body is approximated by a body-body interaction between the body and a point mass located at the center-of-mass of the tree node. The tree traversal continues, but the subtree is bypassed.

Applying center-of-mass approximation in a top down tree traversal reduces the amount of computation in Barnes-Hut algorithm. First, for a body far away from a cluster, the effect of the cluster can be approximated by its center of mass, see Figure 1. As a result a simple two bodies interaction suffices to update the position of the body without calculating individual effects exerted by bodies within the cluster. Secondly, top-down traversal ensures that each body interacts only with the largest clusters for which the approximation is valid. Calculate the total mass and center of mass at each node could be done recursively[15]. The total mass, M, is given by the simply sum of the total masses of the children:

$$M = \sum_{i=0}^{k} m_i$$

where k is the arity of tree and m_i is the total mass of the ith child. The center of mass, C, is given by

$$C = \frac{1}{M} \sum_{i=0}^{k} (m_i * c_i)$$

where position of the centers of mass have many components how dimension have.

Once the accelerations on all the bodies are known, the new positions and velocities are computed in Step 4.

The entire process, starting with the construction of the BH-tree, is repeated for the desired number of time steps.

3 Nested Data Parallelism

Data parallelism, [5][9], is one of the more successful efforts to introduce explicit parallelism to high level programming languages. Data parallel programming is particularly convenient for two reasons. The first, is its ease of programming. The second is that it can scale easily to larger problem sizes. Several data parallel language implementations are available now. However, a perceived disadvantage of data parallelism is that it is only applicable to problems where a large set of data has to be uniformly operated, it is to say, monolithic problems. Hence, a set of independent sub-computations is strongly associated to a subset of these data. Such computations are inherently parallelizable, but each computation itself must be sequential.

The task parallel model, [5][9], achieves parallelism by using multiply threads of control, each getting part of the problem. Although the multiples threads of control have the disadvantage of to be more difficult to understand and use, task parallel approach allows efficient implementations of irregular algorithms.

Nested Parallel model is an extension of standard data parallel model which includes the capability of nested parallel invocations.

The nested data parallelism approach is characterized by dividing the problems into subproblems that are of the same structure as the larger problem [6]. Further divisions into still smaller sub-problems are usually done by recursion. The recursive method will continually divide a problem until the problem cannot be broken down into smaller parts. Then the very simple tasks are performed. The tasks' results are combined with the others tasks' results in the same level. Nested parallelism accomplishes the ability to take a parallel function and apply it over multiple instances in parallel. The significantly enhanced expressiveness of this paradigm originates greater demands upon the computational model that implements it.

To solve Barnes-Hut algorithm applying nested data parallelism is a good challenge, because to Barnes-Hut's characteristics: irregular data structure and communication pattern, and characteristic of nested data parallelism: divide-and-conquer problem whit vectorize data structure.

4 Issues of Parallel Barnes-Hut Implementation

The Barnes-Hut algorithm provides sufficient parallelism [2][8][11]; It has many suitable point to apply parallelism, but a good implementation must consider a number of issues: Data structures, Computation and Communication. The next subsections describe each of them.

Two clear aspect of Barnes-Hut can be parallelized: building of the BH-tree and the force computation. The principal problem of parallel solution is in the communication overhead and

computational throughput: if the bodies partitioned among the processors, the costs of building and traversing the BH-tree can increase significantly. In contrast, the time for arithmetic operations will, essentially, decrease linearly as the number of processors increases.

4.1 Data Structures

If you want to parallelize the building of the BH-tree, you have to take into account that Barnes-Hut has dynamic and irregular data structures and, in consequence they have many difficulties to represent it in distributed memory. So you have to consider:

1. The large number of Barnes-Hut tree nodes must be equitably distributed among processors.

2. Data partitioning must preserve data locality. An inappropriate data mapping which does not preserve locality increases communication overhead and decreases overall performance. The partitioning must assign data to processors carefully to keep data locality and data distribution properties at the same time.

3. The mapping must be dynamically updated so that it can adapt to the evolving data structures.

4. The distributed data structure must be consistent with a sequential implementation.

But, if you want to parallelize the force computation, many of before issues have solutions if global BH-tree representation is present in every processors.

4.2 Computation

The workload of computing new positions must be evenly distributed. The number of floating point operations required to compute the acceleration varies from one body to another. Therefore, it is not sufficient to map the same number of bodies to each processor. Instead the same amount of calculation should be assigned to each processor for fast parallel execution.

4.3 Communication

As Barnes-Hut algorithm has different point to resolve in parallel, the communication pattern depends of the distribution of bodies and data distribution among processors. If the BH-tree is distributed, each processor must request some essential data from other processors. The distribution of bodies can be highly irregular, so the communication cannot be described by a uniform pattern. Moreover, the irregular communication pattern can change from iteration to iteration because of continuous movement of bodies.

Besides, in sequential Barnes-Hut algorithm, more than 90% of the time is devoted to arithmetic operations involved in computing accelerations, and less than 1% of the time is devoted to building the tree, so parallelize the computation of acceleration and speed seem a good idea. In this case, the communication pattern will be regular and possible optimization methods can be apply.

5 Design and Implementation

There have been a large number of parallel implementation of Burnes-Hut algorithm[4][10][14]. In this case we want to make a parallel algorithm where are present paradigm of nested data parallelism and the principle of bulk computation[12] and communication. To apply nested data parallelism implies take advantages to *flat* parallelism [7], and stablish a communication pattern among every processors. To process data in bulk, we separate control into a sequence of alternating computation and communication phases. The communication phase fetches all the remote data essential to the following computation phase. Thus a computation phase starts with all the necessary data available in local memory, and is not slowed down by communication. Alternating computation and communication also results in simple control structure with cleanly defined functionality for each phase.

The figure 4 gives a high-level description of the code structure. Note that the local trees are built at the start of each time step. Step 1 builds the global Barnes-Hut tree from the local bodies. Step 2, in parallel, makes Step 3. The processors are divided in *Ntask* group. The group size is determined by the workload. Each group of processor works on its own sub-set of bodies. Step 3 imply to calculate the accelerations to each body in the processor and update its position depending on the computed acceleration. This parallel process has not combination face.

To finalize this process, every processors have conformed to the new distribution of bodies and can compute the new position of bodies.

```
0.    for every time step do{
1.       build BH-trees
2.       forall  Ntask {
3.            force computation
4.       }
6.    }
```

Figure 4: Outline of code structure

We decided parallelize the force computation phase because to it is the most heavy. The Figure 5 shows the time spent to the tree building and calculation of center of mass, the force computation and new position. You can see the why of this decision.

In the next subsection, we explain with more detail each step.

5.1 Tree Building and The Center of Mass

The tree construction is very simple, each processor builds the tree on every bodies, starting out with the entire extension of plane (two dimensions) or space (three dimensions) in the root node. The number of dimensions determines the tree arity, if the problem is the space, then the arity is eight, oct-tree. Therefore, if it is the plane, the arity is four, quad-tree. The tree has two kind of node, *cell* and *particle*. A cell represents a cluster, it has four or eight children, depend if a quad or oct tree. The root and every inner nodes of the tree are *cells*. A internal *cell* represents a fixed region of plane or space. The leaves of the tree, *particle*, are the bodies, there are exactly as many leaves as bodies there.

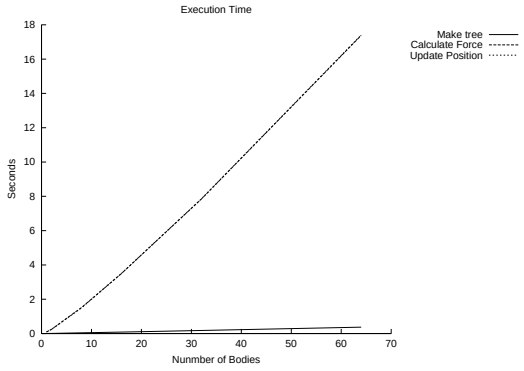

Figure 5: Time of each Barnes-Hut phase

The cost of inserting a body is proportional to distance from the root to leaf in which the particle resides. The close bodies belong to a same cluster, so that they lie in a single tiny sub tree. If the bodies are uniformly distributed, the leaves are all on same level.

Once the BH-tree has been constructed it is possible to start to calculate the total mass and the center of mass of each cluster. These computations are made through a post order traversal of the BH-tree, and the result is stored in each *cell*.

5.2 Force Computation

The Figure 6 shows the structure of parallel force computation process. It is recursive process. The recursive invocation are made in parallel, line 6 and 7. $Ntask$ indicates how many sub groups the current processors set will be partitioned. Initially every processors belong to same group. If there is only one processor, no partition takes place and the acceleration is calculated for each own local bodies and their positions are updated.

```
0.      if (NP==1){
1.          calculate accelerations
2.          update velocities and positions of bodies
3.      }
4.      else {
5.          data partitioning
6.          forall  Ntask
7.              force computation
8.          communicate the new position of bodies
9.      }
```

Figure 6: Structure of Parallel Force Computation Process

At all moment the of parallel computation phase, every processor contains all necessary data that it has to access and the computation is completely local and can be done by the same code as in the sequential case.

In following sections, we describe them with more detail.

5.2.1 Data Partitioning

The force calculation phase dominates simulation time; it is essential that the work during this phase be evenly distributed among processors. The workload of a body is the number of floating point operations required to update its position. The workload of a body depends on its surrounding and varies from one to another. As a result, equal amount of workload, not number of bodies, should be assigned to each processor.

In an N-body problem, the system of particles can be enclosed by a rectangular box for isolated systems like individual galaxies or an exact cube in cosmological systems with periodic boundaries. Salmon[10] developed a parallel algorithm for decomposing this volume into rectangular sub-volumes the method of orthogonal recursive bisection (ORB)[3]. We use ORB to distribute the same load among processors.

In ORB, a volume is represented as a hierarchy of rectangular boxes somewhat like the BH-tree. The main volume is first cut along an arbitrary dimension at an arbitrary position. For a balanced tree, the position is chosen so there are equal numbers of particles on each side of the cut. The resulting two volumes are cut again along the same or different dimension again at an arbitrary position. The slicing of each sub-volume continues for as many levels required. The resulting process can be represented by a binary tree structure since each node points to two children. An ORB tree of n levels therefore results in 2^n sub-volumes. This is suitable to most massively parallel machine, because of they are organized into partitions of 2^n processors.

At each level in the construction of the ORB tree, we have the freedom to choose both the dimension and position of the volume subdivision. With load balancing in mind, we wish to position our slice of sub-volume so that there are equal amounts of *computational work* on each side of slice. We can keep track of the computational work for an N-body simulation by simply knowing, for each body, the number of cell-particle interaction.

The choice of dimension for cutting the sub-volumes in the ORB tree construction is generally arbitrary but it is best to select to longest dimension for slicing.

The ORB tree decomposition is similar to divide-and-conquer technique, it can be represented by a binary tree. For all these, we can apply nested data parallel.

We chose ORB decomposition for several reasons. It provides a simple way to decompose space among processors, and a way to quickly map points in space to processors. Furthermore, ORB preserves data locality reasonably well and permits simple load-balancing. Finally, many searches have demonstrated that the ORB partitioning results in excellent speed-up for the Barnes-Hut algorithm.

5.2.2 Calculating Accelerations and New Positions

In this proposal, these two phase, as in parallel such as in sequential, are the same. In other word, these two phase in sequential Barnes-Hut algorithm are the same that in the parallel Barnes-Hut algorithm, the only difference is in the set of bodies.

Each processor calculates the new position of its own bodies. To compute accelerations and new positions, we do not need any data from other processor, and in consequence no communication is required.

The acceleration is computed to every bodies by traversing the path from BH-tree's root to particular body.

The parallel implementation can speed up the tree traversal by ignoring some tree nodes, just it visits the nodes on the way to body.

5.2.3 Communication

The ORB data decomposition determines an hypercube model communication. An ORB decomposition on a parallel machine can be carried out by assuming the processors are laid out in a hypercube communication, although they may not be physically connected in this way. A machine containing 2^n processors can be arranged using the communication model of an n-dimensional hypercube. In this arrangement, each processor resides at the vertex of this hypercube and can only communicate to n neighbors along the lines on edge hypercube.

The data division establishes a relation among processors in the different sets produced by the division. Each processor in a processors set settles a partnership relation with one or more processors in the other subsets. The partnership relation determine the communication of the results produced by the parallel task. The structure of division generated and partnership relation among processors give place to communication patterns among processors that are topologically similar to a hypercube. The number of divisions determines the dimension of hypercube and the number of parallel task is the degree of the hypercube.

The communications are necessary at end of position updating phase. In each time, every processor has to have the current space, all bodies. When each processor finalizes the computation of new positions of its bodies, it has to communicate them to other processors. Only, at this moment is necessary the communication.

6 Conclusions and Future Works

The design of a portable and efficient parallel implementations of adaptive N-body methods such as Barnes-Hut algorithm is critical. Thus, simulating such system means advancing the bodies in discrete time steps. In each time step, the algorithm computes the force exerted on each body due to all other bodies; this determine the acceleration and speed of that body during the next time.

In this paper, we analyze a parallel Barnes-Hut algorithm, discuss its critical point to implement it in distributed memory machine, and propose a possible solutions applying nested data parallelism.

This propose is the first step. Here, we decided to implement a parallel Barnes-Hut algorithm, applying parallelism in those phases that spent the most computation time: calculate force and the new position of bodies.

Our final aim is to implement a portable Barnes-Hut algorithm, where each of its phase will be resolved in parallel. This task requires careful data structure design because of implementing the Barnes-Hut tree in distributed memory is so difficult. Moreover, the parallel BH-tree structure implies a dynamic and irregular distribution of bodies. The irregular distribution makes data mapping difficult to compute, and the movement of bodies mandates the data mapping to be dynamic.

An earlier implementation and its executions show us the advantages of this propose. Finally, we argue that if a good speed up is obtain to our first model, then the final full parallel Barnes-Hut algorithm will work very well on distributed memory machine.

7 Acknowledgments

We acknowledge the co-operation of the project group for providing new ideas and constructive criticisms. Also to the Universidad Nacional de San Luis and the ANPCYT from which we receive continuous support.

References

[1] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. Nature, 324, 1986.

[2] S. Bhatt, M. Chen, C. Lin, and P. Liu. Abstractions for parallel N-body simulation. In Scalable High Performance Computing Conference SHPCC-92, 1992.

[3] D. Blackston and T. Suel. Highly Portable and Efficient Implementations of Parallel Adaptive N-Body Methods. In ACM. 1997.

[4] G. Blelloch and G. Narlikar. A practical comparison of N-body algorithms. In Dimacs Implementation Challenge Workshop, October 1994.

[5] G. Blelloch. Programming parallel algorithms. Communications of the ACM, 39(3):85-97, March 1996.

[6] J.A. González, C. León, F. Piccoli, M. Printista, J.L. Roda, C. Rodriguez, F. Sande. Groups in Bulk Synchronous Parallel Computing. Proc. 8th Euromicro Parallel and Distributed Processing. Pp 246-253. January 2000.

[7] F. Almeida, V. Blanco, I. Dorta, F. Garcia, J.A. González, C. González, C. León, F. Piccoli, M. Printista, J.L. Roda, C. Rodriguez, F. Sande. Nested Parallelism as a Mechanism to Integrate Passing and Shared mamory Programming. XII Jorandas de Paralelismos. Pp, 173-178. Universidad Politécnica de Valencia. España. Setiembre de 2001.

[8] Y. Hu, S. L. Johnsson, and S.H. Teng. A data parallel adaptive N-body method. In Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing. SIAM Press, 1997.

[9] M. Quinn. Parallel Computing. Theory and Practice. Second Edition. McGraw-Hill, Inc.

[10] J. Salmon. Parallel Hierarchical N-body Methods. PhD thesis, Caltech, 1990.

[11] J. Singh. Parallel Hierarchical N-body Methods and their Implications for Multiprocessors. PhD thesis, Stanford University, 1993.

[12] L. Valiant. A bridging model for parallel computation. Communications of the ACM, 33(8):103–111, 1990.

[13] M. Warren and J. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In Proceedings of Supercomputing, 1992.

[14] M. Warren and J. Salmon. A portable parallel particle program. Computer Physics Communications, 87, 1995.

[15] B. Wilkinson & M. Allen. Parallel programming: Techniques and Application using Networked Workstations and Parallel Computers, (Prentice-Hall, 1999).