Computer Science Problems in Astrophysical Simulation

Richard Anderson

Abstract

This paper presents a survey of current work on N-body simulation in astrophysics. The goals of the paper are to present several computer science problems that arise in N-body simulation, and to show how cross disciplinary collaboration can enrich computer science.

1 INTRODUCTION

A major challenge to Computer Science is to successfully collaborate with other science and engineering disciplines. In the last three decades, Computer Science has built its foundations, with dramatic advances in fields as diverse as Theory, Artificial Intelligence, Computer Architecture, and Programming Languages. However, during this period of growth, the emphasis has been inward, to build a discipline, as opposed to looking outward, to apply techniques to problems encountered by other scientists and engineers.

In this paper, I discuss first hand experience from a collaborative project between Astronomers and Computer Scientists at University of Washington. The project is to study astrophysical simulation algorithms and to implement them on high performance parallel computers. The basic goals of this paper are to present the Computer Science problems that arise out of this work, and to argue that this type of collaboration yields many benefits to Computer Science. (It is hoped that the Astronomers will also argue that they benefit from this type of collaboration.)

2 Collaboration

Many forces are pushing collaboration between Computer Science and other disciplines. First and foremost there is the intellectual argument that this collaboration will be mutually beneficial and will advance all of the involved disciplines. This argument is being formally made by organizations such as the National Research Council [1]. Currently, at least in the United States, funding opportunities such as the HPCC Initiative are promoting collaborative ventures. Collaboration with an application discipline is often necessary if one want to have access to state of the art high performance computers since the machines are now way to expensive for a single academic unit to afford.

In spite of the forces encouraging collaboration, there are many difficulties to overcome to engage in a successful collaboration. First of all, there is a very large overhead in starting work in a different discipline. The overhead of starting work includes both learning the basics of the science, as well as overcoming a language and culture gap between fields. Another difficulty in starting a collaboration is to have accurate expectations about the role of the collaborators, and to understand how everyone involved can contribute to the goals of the project. A final difficulty is to make sure that the project is one in which collaboration is possible. There are many computational problems which are already well solved, so that improved performed is only a question of accessing greater resources. In this case, there may not be computer science problems to solve, so the collaboration may be unwarranted. Another issue in collaboration is to ensure that all members have an opportunity to gain credit for the work. In the standard academic currency, this means that results of the work have to be publishable. The key issue is where these results will be published, since it is critical to be able to publish in ones own field.

This work is part of the University of Washington project on Astrophysical Simulation. The project includes eight faculty members from Astronomy, Physics, Applied Mathematics and Computer Science, as well as four post doctoral research associates. The project is currently funded as a NASA HPCC grand challenge project. Although

the project is only one year old, it has been very successful in building ties between fields and in establishing a high level of interaction. The project centers around the development of several different simulation codes, with a strong emphasis on parallel implementation. This paper will stress the computer science side of the project and will not discuss the astrophysical problems that will be studied through simulation.

3 Astrophysical Simulation

Particle simulation is widely used in many scientific and engineering disciplines, including chemistry, material science, bioengineering, and mechanical engineering. The technique is used to complement experiment and observation. Astrophysical simulation generally involves following the time evolution of a set of particles under gravitational force. The particles often correspond to the aggregation of a large number of stars. Simulation is particularly important to astrophysics because the time and distance scales involved make many observations difficult and experiment impossible. The difficulties in observation are that it is difficult to determine the distance of far away objects such as galaxies, so that we essentially have a two-dimensional view, and the time scales involved are so vast, that we essentially only have a snap shot of the universe, and cannot view objects moving through time.

The basic algorithm is to compute the force on each particle and then advance the particles assuming constant force for a small amount of time. The problem of computing the force on each particle is generally referred to as the N-body problem.[1] In many applications of simulation, it is necessary to use a very large number of particles to accurately reflect the physics of the problem. The current state of the art is to perform simulations involving about twenty million particles using the largest parallel computers. Astrophysicists argue that qualitatively different problems will be able to be studied if the number of particles can be increased by one or two orders of magnitude.

[1] However, in the case of small N, notably two and three, the N-body problem may refer to the problem of computing the equations of motions for the bodies. In the case of two, Newton showed that the motion is an ellipse. The problem of computing the equations of motion for three bodies appears to be intractable.

3.1 N-body Problem

The N-body problem is: given a set of particles with masses and positions, determine the force upon each particle, assuming gravitational interaction between particles. Since gravitational force obeys an inverse square law, the problem is to compute the sum:

$$F_i = \sum_{j \neq i} \frac{g m_i m_j}{\|x_i - x_j\|^2} \widehat{x_j i}$$

where $\widehat{x_j i}$ denotes the unit vector from x_i to x_j. The force on a single particle can be computed in $O(n)$ operations,[2] so computing the force on all particles can be done by a straightforward $O(n^2)$ time algorithm.

The improvement over an $O(n^2)$ algorithm comes from computing approximate forces instead of exact forces. As we discuss later, the basic method of approximation is to aggregate particles and to approximate the effect of a collection of particles by a single computation. Although it is important to pay attention to the accuracy of the approximation, it is not necessary to evaluate the forces to a very high level of accuracy. The reason for this is that there are several other sources of error in the simulation process, so that a highly accurate evaluation of forces does not increase the overall accuracy of the simulation. The integration process is one source of error, since forces are assumed constant during a time step. The other major source of error is that a physical system is often represented by a much smaller number of particles than it actually contains, so each particle in a simulation corresponds to a large number of real particles. This modeling error is often the factor the limits the utility of simulation, and it also motivates the use of larger numbers of particles.

3.2 Application of Simulations

Many different problems in astrophysics are addressed by simulations. These simulations can classified roughly into two categories, large scale and small scale. The distinction is that a large scale simulation is aimed at understanding the structure of the Universe as a whole, while a small scale simulation studies the behavior of a specific system. The type of problem that is studied by large scale simulation is one such as galactic clustering, where the

[2] We are assuming infinite precision operations to claim an $O(n)$ bound. Since the simulation algorithms are generally done at a low precision, the issue of solving the N-body problem to a high level of accuracy does not arise.

major questions include understanding the overall distribution of galaxies in the universe, and relating internal structure of galaxies to the large scale structure. Examples of smaller systems which are studied by simulation include the collision of galaxies and the mass exchange between orbiting white dwarf stars[2].

One of the key technical challenges in simulation arises from the necessity of using a large number of particles. The largest reported simulations (as of June, 1992), involve following the evolution of 17 million particles over 600 time steps [3]. The simulations were performed by Salmon and Warren using the 512 processor Intel Touchstone Delta (i860). The simulations ran in roughly 24 hours. These simulations were very large scale simulations where each particle represented over 10^{10} solar masses, and each time step corresponded to over 10 million years. Smaller "routine" simulations run by astrophysicists may involve one million particles and run for one month on a fast workstation [4].

The reason that astrophysical simulations require such a large number of particles to achieve accurate results is that the problems involved have a very large dynamic range. This means that interesting events simultaneously occur on many different length scales. The type of problem where different length scales occur is in looking at galactic clustering. The galaxies are very widely distributed, and it is of interest to be able to see the internal structure as well. This means that there must be a sufficient number of particles in each galaxy to allow its features to emerge, and enough galaxies to model the large scale distributional effects. It is generally believed that increasing the number of particles by one or two orders of magnitude will allow qualitatively different physical problems to be studied.

4 N-body Algorithms

A rich set of methods have been developed which improve on the basic n^2 algorithm for the N-body problem. Currently, a set of methods, referred to as *tree-codes* are the most popular for large scale simulations. These were independently discovered and developed by a number of researchers.

4.1 Analog algorithm

The history of N-body algorithms dates back to a truly remarkable paper published in 1941[5]. In this paper, Erik Holmberg, a Swedish physicist described an analog method for N-body simulation. His idea was to replace gravitational force with light, since both light and gravity obey an inverse square law.

Holmberg was interested in studying tidal disturbances caused by nebulae passing in close proximity to one another. He dismissed numerical integration as being computationally infeasible. The specific system that he considered was two nebulae, consisting of 37 bodies each, so a single time step would have required computing 2,701 square roots, which would have been a tremendous amount of work without a calculator. The experimental system that Holmberg used was a collection of 74 light bulbs set up on a board. The light bulbs were replaced one at a time by a light meter to determine the forces. By measuring the light intensity in the $+x, -x, +y$ and $-y$ directions the force could be determined. Particular care was taken with the experimental set up, for example, special light bulbs were manufactured with a vertical spiral filament to ensure that the light was uniform in all directions. The paper also discussed how errors such as reflected light from the table surface were measured and dealt with.

4.2 Mesh Algorithms

The first computational method which gave a substantial improvement over direct computation was to use a mesh[6]. In this method, each particle is moved to the closest grid point. The force computation becomes a convolution which can be done with a FFT. This reduces the complexity of the computation to $M \log M$ where M is the number of grid points used. The main drawback to this method is the grid, which determines the resolution of the simulation. One of the main characteristics of astrophysical simulations is that they involve a very non-uniform distribution of mass so a large range of resolution is important. A number of approaches have been taken to improve upon the mesh algorithms, including hybrid methods which use direct computation for close interactions, and the mesh computation for long range interaction, and multigrid techniques which allow varying the mesh size.

4.3 Tree-codes

The current method of choice for astrophysical simulation is to approximate the force on each particle

with the aid of a geometric data structure. Implementations of this method are often referred to as *tree-codes*. The method was independently discovered by several researchers. The particular version that is most commonly used is the Barnes-Hut algorithm [7].

The starting point for the tree codes is to use a natural approximation in computing the force. Suppose that we want to compute the force exserted by a set of particles $S = \{p_1, \ldots, p_k\}$ on a particle x. If all of the particles in S are far away from x, then it is natural to replace the set of particles S by a single particle located at the center of mass, and assign all of the mass to this particle. A more accurate approximation can be achieved by using a multipole expansion around the center of mass instead of just the single term. One of the important details in the algorithm is to decide when the particle is sufficiently far away so that the approximation can be used.

The spatial data structure that is used is a tree, where a set of particles is associated with each of the tree nodes. The data structure obeys the following three properties:

1. The entire set of particles is associated with the root.
2. The children of a node T represent a partition of the particles associated with T.
3. Each leaf has at most one particle associated with it.

In principle, any tree that satisfies the properties could be used. However, it is important that the tree decomposition reflects the spatial distribution of particles. Many of the methods that are used are based upon recursively decomposing space, and assigning all of the particles in a spatial region to a tree node.

The basic algorithm to compute the force on a particle p can be expressed as the following recursive procedure. We assume that there are subroutines for computing the force and testing if the approximation is valid. For clarity of expression the tree is assumed to be binary.

```
Evaluate(p : ParticleType, T : TreeNode)
  if IsLeaf(T)
    return ExactForce(p, T);
  if FarAway(p, T)
    return ApproxForce(p, T);
  return Evaluate(p, T.left) + Evaluate(p, T.right)
```

The simulation algorithm is to compute the force upon each particle using the routine Evaluate. The run time to compute the force on a particle is proportional to the number of nodes that are expanded. An important aspect of a tree-code is that it relies on performing particle-cluster computations. We discuss methods which allow cluster-cluster computations below.

Opening criterion One of the important subroutines of the algorithm is the test as to whether or not the approximation by a region is sufficiently accurate. This is often called the *opening criterion*, since it is used to decide if a node should be expanded. The basic condition for using the approximation for the set of particles S to compute the force on x is that all points in S are far away from x. The standard way to implement this is to look at the ratio between the size of S and the distance from x to S. In the Barnes-Hut algorithm, each point set is enclosed in a cube. If s is the length of the cube side, and r is the distance from x to the center of mass of S, the approximation is used if $\frac{s}{r} < \theta$, where θ is an input parameter that controls the accuracy. Many simple variants of this could be used, for example, the size of the point set could be measured by its diameter (in either the L_2 or L_∞ norms), and the distance to the point set could be measured by the distance from x to the boundary of S or to the closest point of S instead of to the center of mass. Salmon and Warren [8] consider other choices including methods which take into account the magnitude of the errors in using various approximations.

Data Structures There is a tremendous flexibility in the choice of spatial data structure that could be used in the algorithm. To enhance the accuracy of the computation, it is important the the regions have roughly the same size in all directions. It is also desirable that the regions chosen reflect the geometry of the point set. The methods that are in current use can be divided into top down structures, which recursively divide the space into regions using planes parallel to the coordinate axes and bottom up methods which recursively combine close together particles to form clusters.

The Barnes-Hut algorithm uses the *oct-tree* data structure. The point set is assumed to lie inside a cube. The oct-tree is constructed by recursively subdividing the cubes into eight subcubes, splitting at the geometrically central point. The subdivi-

sion continues until cubes contain fewer than two particles. Other data structures, including $k-d$ trees, used by Appel [9], and fair-split trees, proposed by Callahan and Kosaraju [10], choose separating planes based upon the point set, where the subdivision does not necessarily create equal sized regions.

The bottom up up approach aims at grouping together points in a way that reflects the geometry of the particles. Independently, Benz et al. [2] and Jernigan and Porter [11] gave schemes where close together points are combined to form clusters. Although these data structures are much less understood than the top down approaches, they appear to perform well in practice [12].

Performance Although the performance of the particle-cluster algorithms is generally characterized as $O(n \log n)$, the actual run time does depend upon the distribution of the points. Since the force on a particle is computed by traversing a tree from the top, down to some of the leaves, the tree height appears in the run time. The height of an oct-tree can be unbounded in terms of the number of particles, since the division into regions does not necessarily subdivide the points[3].

We prove that the run time of the Barnes-Hut algorithm is related to the average depth of leaf nodes. If the tree is balanced, the average depth is $O(\log n)$, which gives the $O(n \log n)$ bound which is generally claimed for the algorithm. The proof is to bound the amount of work done in terms of the cells that are examined. The total leaf depth of a tree is the sum over all of the leaves of their depths.

Lemma 1 *Let x be a particle. The number of cells of size D encountered when evaluating x is bounded by a constant.*

Proof: The number of cells of size D encountered is at most the number of cells which can be direct descendants of cells of size $2D$ which fail the accuracy test. Let C be a cell of size $2D$, and suppose the distance of the center of mass of C to x is r. C fails the accuracy test if $\frac{2D}{r} > \theta$. The number of cells which fail the accuracy test is bounded by the number of disjoint cells of size $2D$ that can be placed so that their centers of mass are within distance $\frac{2D}{\theta}$ of x. This is bounded by the number of cells of size $2D$ which can be placed inside a sphere of radius $\frac{2D}{\theta} + 2D\sqrt{3}$ which is constant. ∎

We also need a related lemma, which bounds how many times a particular cell can be examined by particles in larger cells.

Lemma 2 *Let C be a cell of size D. The number of particles x located in leaf cells of size $D' > D$ which encounter C is bounded by a constant.*

Proof: A particle can only encounter C if C's parent fails the accuracy test. This means that x must be within distance $\frac{2D}{\theta}$ of C. The leaf cells are disjoint. By a straightforward packing argument, the number of cells of size at least $2D$ within distance $\frac{2D}{\theta}$ is bounded by a constant. ∎

Theorem 1 *Let T be an oct-tree total leaf depth L. The Barnes-Hut algorithm takes time $O(L)$ on input T to compute the force on all of the particles.*

Proof: The two lemmas allow us to account for all of the work. Let x be a particle that is in a leaf cell of size D. We account separately for the work that x does in looking at cells of size D or greater, and the work done at looking at cells of size less than D. Suppose that x is at depth k in the tree. The first lemma says that $O(k)$ work is done looking at cells of size at least D. The second lemma says that each cell is evaluated at most a constant number of times by particles in larger cells, so this gives a linear amount of work summed over all cells. The total amount of work is thus proportional to the total leaf depth. ∎

It is of interest to look more carefully at the constants. The proof of the first lemma gives a constant of

$$\frac{32}{3}\pi\left[\frac{1}{\theta^3} + \frac{3\sqrt{3}}{\theta^2} + \frac{9}{\theta} + 3\sqrt{3}\right].$$

For large values of θ, when θ is relatively close to one, the bound is pessimistic. For example, when $\theta = 1$, the bound is 683, while a direct bound for $\theta = 1$ gives a constant of 216. The bounds make a worst case assumption about the location of the center of mass in each cell. If the center of mass is in the center of each cell, then the bound improves to 64 when $\theta = 1$. The large branching factor of the tree and the cubic dependence on θ cause the constant to be so large.

[3] However, in actual simulations the problem of unbalanced trees is not overwhelming. One reason is that the pathological cases require tremendous precision, so they cannot occur when fixed precision arithmetic is used.

4.4 Fast Multipole

It is possible to gain a theoretical improvement in N-body algorithms by allowing cluster-cluster approximations instead of just particle-cluster operations. Appel [9] introduced the use of cluster-cluster operations, and Greengard and Rokhlin [13] showed how the operations could be used to achieve high accuracy in linear time.

The idea for cluster-cluster evaluations is that if there are point sets A and B, which are *well-separated*, then a single force computation can be done between the sets. Appel's approach was to compute the acceleration assuming all mass was concentrated at the center of mass, and then use that acceleration for all of the points in each set. The Greengard-Rokhlin method is far more sophisticated. In their method they translate the force field of B so that it applies to the domain of A. This is done by using several Taylor series expansions. Once the acceleration is computed for a region, it is passed down the tree towards the leaves which correspond to the particles. In the Greengard-Rokhlin algorithm, the values which are passed down the tree are actually functions, which are evaluated at the particle's position.

The run time for the cluster-cluster algorithms is generally considered to be $O(n)$, however, this is under some assumptions about the distributions of the particles. In particular, if the particles are very highly concentrated, the trees can have a large depth, which adds additional cost to the algorithm. The basic argument used to establish linear time is to argue that each cell can be involved in only a constant number of evaluations. This is done with a packing argument that is similar to the lemmas established above. If the Greengard-Rokhlin algorithm is also parameterized by the output precision, then it is possible to achieve linear time.

Although the Greengard-Rokhlin algorithm is theoretically superior to the Barnes-Hut algorithm and the other tree-codes, it is not widely used for astrophysical simulation. The main reason for this is that the constant factors are large in the algorithm, and the equations used in the evaluation are very complicated. In order for the force approximation to be valid, the clusters must satisfy a well-separated property, which is that the ratio of the separation and the cell radii must exceed a constant. In the three dimensional algorithm, to evaluate a cell of size D, all D cells in a cube of diameter $9D$ must be examined, which is 729 cells [14]. The force evaluation is also very expensive. In two dimensions, the coordinates can be represented with complex numbers, which greatly simplifies the computation of the functions. In three dimensions, a different technique, such as spherical harmonics is necessary [15].

The fast multipole method is well suited for the case where high accuracy is required. Although it has large constant factors, additional accuracy is available at relatively low cost. However, astrophysical simulation generally requires only low accuracy, so the overhead of the fast multipole algorithm limits its utility.

4.5 High performance computing

Since large N-body simulations have such high computational demands, they have been natural applications for the most powerful computers. The initial implementation on supercomputers were on vector machines. The control structure of the N-body algorithms, made it a substantial challenge to get good performance on machines such as Crays, but eventually vectorized codes were developed. Currently, the main interest in implementation is achieving high performance on large parallel machines. The largest simulations that have been run [3] utilized the 512 processors Intel Touchstone Delta.

The N-body problem potentially has a large amount of work that can be done independently and in parallel, especially when the number of particles is much larger than the number of available processors. However, there are a number of problems that must be solved to achieve high efficiency. The problem that has been the most challenging has been to implement the algorithm on a distributed memory machine. It is necessary to have methods which associate the data with specific memories and allow convenient access by other processors. Solving this problem means not only solving the efficiency problem on a particular machine, but also addressing implementation and portability issues. A secondary problem to solve is the load balancing problem, which is to make sure that all processors remain active. Both of these problems have been addressed in a number of implementations, and good performance has been achieved on several different machines. Machine utilization of over 80% has been reported on hypercubes [16], and on the Stanford Dash [17].

4.6 What is left to be done?

Physicists desire to perform simulations which are orders of magnitude larger than what is feasible today. Today's simulations are limited both in terms of total processing power (the speed and number of processors), as well as by the available memory. Larger simulations will be enabled by access to larger and faster machines. However, there is also a major role that computer scientists can play in making these large simulations possible. Improved understanding of simulation algorithms will increase confidence in results, and will also allow resources to be concentrated upon the parts of the simulation which are most sensitive in errors. There are opportunities to improve the performance of the algorithms. Even constant factor improvements in the algorithms will be important in increasing the size of problems which can be addressed. Finally, there are many open computer science problems which relate to high performance computing in general. These include problems such as developing methods for portable parallel programs and developing support software for scientific applications.

5 Errors in Simulation

The fundamental concern when working with simulation is the degree of correspondence between the simulation and a real system. There are certain constraints, such as energy conservation which are easily checked. These either give confidence in the accuracy of a simulation, or establish that the simulation is not correct. It is generally easier to show that a simulation is incorrect than to establish its correctness. If the simulation exhibits any nonphysical behavior, then it is necessarily suspect. However, the fact that a simulation gives the expected results on test cases, does not guarantee that it will be correct when applied to new situations.

Errors in simulation can be either caused by having too large a granularity, or can be systematic errors in the algorithm. Granularity errors include having too few particles in the system, having a timestep that is too big, or using an approximation threshold that is too weak. Systematic errors are a very big concern, since they indicate that the method is flawed. In this section three problems are mentioned which have come up in simulation algorithms. The reason for discussing these cases is to show the importance of understanding what the simulations are computing, as opposed to just concentrating on the performance of the simulation.

5.1 Disk heating

One problem that was observed in early simulations was that galaxies gradually disappeared.[4] This has been termed the "Disk heating problem", since is is caused by particles being accelerated and ejected. The problem arises from performing integration with respect to the $\frac{1}{r^2}$ gravitational force. Numerical integration is performed by treating a force as constant throughout a timestep. If a particle is passing close to another one, and if the force is measured when the particles are very close together, then there is a large acceleration, while if the force is measured at a different point in the trajectory, the acceleration would be much slower. The problem comes from the accidental measurement of very close encounters causing large accelerations. To fix this, the $\frac{1}{r^2}$ gravitational force is "softened", and replace by another force law, such as $\frac{1}{r^2+\epsilon}$ for some $\epsilon > 0$. This solution appears to be ad hoc, although it does give more accurate simulations.

5.2 Error bounds

The error bounds chosen in simulations are surprisingly weak. Generally, the parameter θ in the Barnes-Hut algorithm is chosen in the range 0.7 to 1.0. The accuracy of the force is only guaranteed to be within about 30% of the real force. The force computation is the sum of a number of terms, so if errors were uniform and independent, then the accuracy would increase because of a cancellation of errors. The errors are not independent, and there is also conditioning between time steps, so a central limit theorem approximation is not valid. It is probably not possible to formally derive a distribution on the errors, although they have been observed to satisfy reasonable statistical properties [12]. A worst case analysis, which assumes a conspiracy of errors, yields an overly pessimistic bounded. Motivated by the needs to run large simulations, physicists will continue to run simulations outside the range where error analysis gives reasonable bounds on the accuracy.

[4] Galaxies would disappear in under one billion years of simulated time, while real galaxies have existed for ten billion years.

5.3 Exploding galaxies

Salmon and Warren [8, 16] have documented a systematic error arising in the Barnes-Hut algorithm. The problem arises when a small galaxy collides with a larger galaxy. As the smaller galaxy approaches, it loses self gravitation and falls apart. What is especially peculiar is that this occurs when the small galaxy approaches at 45 degrees (on the line $y = x$), but does not occur when it approaches along the x-axis. This problem was discovered in actual simulation, as opposed to being contrived to show fault with the algorithm.

The problem is a "corner effect" that occurs when the bulk of the mass is located at the corner of a cell, and it depends upon the specific opening criterion used in the Barnes-Hut algorithm. There are many ways to solve this problem, such as requiring a lower error threshold ($\theta < \frac{1}{\sqrt{3}}$) or by modifying the opening criterion. The unsettling aspect of this problem is not that it is difficult to deal with, but that the problem had been in codes that had long been used as production codes for astrophysical research. There is no guarantee that there are no other pathologies in the Barnes-Hut or other algorithms which will cause physically incorrect results. It clearly is not possible to develop a set of benchmark simulations which are sufficient to validate simulation algorithms.

6 High Performance Computing

A major challenge is to make effective use of high performance computers in executing the existing algorithms for N-body algorithms. The word effective is used to indicate a wide range of issues; performance is of critical importance, while issues such as ease of implementation and portability between machines are also very important. Researchers are seeking general solutions to many problems in high performance computing. The N-body problem has become a popular application to study [18] since it has a less regular structure than many other scientific applications, and hence requires more general solution techniques. (One concern about the computer science work that uses the N-body problem as a benchmark is that it is concentrating on the Barnes-Hut algorithm. N-body algorithms are evolving to have less regularity than Barnes-Hut, so solutions proposed might not be sufficiently general. It is important for Computer Scientists to keep abreast of the advances of simulation algorithms.)

6.1 Parallelization

There are currently a range of different architectures used for high performance computers. Some of the issues in parallelization differ between classes of machine. Researchers are seeking solutions to these problems that apply to as wide a range of machines as possible.

The instances of the N-body problem that arise in astrophysics generally provide a large amount of work that can be done in parallel, so the problem is to take advantage of existing parallel work was opposed to finding a new algorithm with enough parallelism. In current algorithms, almost all of the work is done in actual force computation with construction of the tree data structure being only a small amount of the work. (Salmon [16] gives the cost of tree construction as only 2% of the sequential run time.) The load balancing problem is to assign the particles to the processors, so that each processor performs approximately the same amount of work. This can be done in either a static or a dynamic manner, where a static method divides the work before the computation and a dynamic method can reallocate work to keep all processors busy. Static schemes that have been used for the N-body problem include a top down method, *Orthogonal Recursive Bisection* [16] and a bottom up method, *Costzones* [19]. One idea that works well in load balancing is to keep track of the amount of work per particle done in each time step, and use it as an estimate of the work when load balancing is done. The load balancing problem is relatively well solved for current N-body algorithms, so the current challenges are to develop general methods which apply to large classes of applications.

A second problem in parallelization is to assign the data structure to memory. The characteristics of this problem depend strongly upon the type of machine that is being considered. If the machine is a distributed memory machine, then the problem is to choose where the data goes to minimize communication costs. On the other hand if the machine is a shared memory machine, where memory management is not under user control, the problem is to ensure that the memory system can take advantage of data locality to achieve reasonable performance. In the N-body problem, the key is to map the spatial decomposition tree to memory. The problems

include making sure that each particle has access to all relevant portions of the tree, and determining which processor is responsible for updating each of the tree nodes.

One direction of research is to develop system level solutions to problems such as load balancing and memory mapping instead of user level solutions. Reasons for doing this include a desire to avoid solving the problem for each application, a system level solution could interact with other system level functions, such as job scheduling, and a system level solutions could use facilities not normally available at user level.

6.2 Abstraction

The sequential version of the Barnes-Hut algorithm is elegantly expressed in terms of tree traversal. This leads to convenient implementation in languages which support pointer manipulation.[5] However the sequential abstractions are not sufficient for parallel implementation, especially when a message passing machine is in use. The most difficult problem is to ensure that each processor has access to all necessary parts of the tree. This is a very difficult implementation problem. For example, the solution to this problem was the most complicated part of the Salmon implementation [16]. The problem is easily solved on a shared memory machine [17]. This specific example is used by proponents of shared memory in the on going discussion of the relative merits of shared memory versus message passing.

A major step in developing abstractions for tree codes on message passing machines was taken by Bhatt et al. [20]. They propose a pair of abstractions, *Traverse* and *Deliver*, which lead to a concise implementation of the Barnes-Hut algorithm. The key mechanism is to define a tree traversal which prompts the delivery of messages to the originating processors. This type of implementation requires library routines to be written which support message passing and parallelism. The user is not exposed to any of the details of the machine. There are very strong arguments for this style of implementation based upon software engineering considerations such as portability, extensibility, and ease of maintenance. The one weakness of the Bhatt abstraction is that it was designed strictly for the Barnes-Hut algorithm . It is not clear if it is sufficient for newer algorithms which have different strategies for node expansion, or allow different time scales.

[5] Astrophysical simulation is a domain where scientific programmers have abandoned FORTRAN with a preference for C.

6.3 Portability

A very general and difficult problem is to develop methodologies and support software which allows programs to be moved between different parallel machines. It is essential that methods for porting programs pay very close attention to the efficiency of the resulting program. There are several ongoing efforts, such as the Orca Project at University of Washington [21, 22] which are developing systems which apply to various classes of scientific applications. Again, the control structure of the N-body algorithms make them a very interesting type of application to study.

7 Algorithms

The most significant improvements in the performance of N-body algorithms have come through algorithmic innovations, and there are still many sources of potential improvement. Algorithmic research in the N-body problem aims at both finding faster algorithms as well as gaining a better understanding of existing algorithms.

The key algorithmic problems turn out to be in computational geometry and data structures as opposed to being in parallel computation. The reason for this is that N-body algorithms generally have enough inherent parallelism that it is easy to find enough parallel work to achieve high utilization of today's parallel machines. For example, solving a problem with $N = 1,000,000$ on a one thousand processor machine gives an adequate grain size for current machines. If machine sizes were to increase substantially, and problem size were not to increase, then parallel algorithms issues would become more important.

7.1 Spatial Data Structures

The central part of a particle-cluster tree code is the geometric data structure which drives the flow of control and determines the approximation. These algorithms must take $\Omega(n \log n)$ time, so the goal is to improve the constant factors. The constant factors are large enough that there is ample room for improvement, and these codes use sufficient computational resources that even modest improvements

in the size of the constants would be of significant practical importance. Almost all of the work is in the force computation, so we want to find a data structure which reduces the number of node evaluations. In the current algorithms, tree construction is a very small amount of the run time. This means that a more complicated tree construction strategy may be practical if it succeeds in reducing the number of force computations.

The majority of N-body algorithms use a spatial decomposition that is derived from a subdivision of orthogonal planes. One of the simplest schemes is the oct-tree which is used in the Barnes-Hut algorithm. This partition scheme is oblivious to any structure that the data might have, for example, a separating plane might pass directly through a tightly packed cluster. It is very likely that other decomposition could yield better performance. The constant factor in the Barnes-Hut algorithm is moderately large. It is determined by the number of cubical cells which intersect a fixed radius sphere in three dimensions. This number turns out to be alarmingly big. Cubes are not the optimal shape of a fixed volume cell when attempting to minimize intersections with a sphere, so it is conceivable that a different tiling would give a better result.

There are many different directions to explore in modifying the Barnes-Hut algorithm to improve the constant factors. These problems could be addressed both analytically as well as experimentally.

1. Determine the constant factor in the Barnes-Hut algorithm (as a function of θ). This could be done with respect to the average case as well as to the worst case.

2. Can better data structures be constructed using arbitrary planes, instead of restricting attention to orthogonal planes?

3. Does the use of bounding boxes which are tight against the data improve oct-tree behavior?

4. Can a data structure which splits the points roughly evenly, and achieves a guaranteed $O(\log n)$ depth improve the algorithm's performance?

5. Is the high branching factor a detriment to the oct-tree based algorithm? Can a binary tree lead to fewer comparisons?

6. Can the theory of geometric separators [23] be used to build better trees?

7.2 Nearest Neighbor Trees

An alternative to the top down approach is to build the tree bottom up by combining close together particles. This method was independently proposed by Benz et al. [2] and by Jernigan and Porter [11]. This method appears to be competitive with the top down approaches [12].

We define a nearest neighbor tree to be a tree that is formed by repeatedly collapsing mutually nearest neighbors[6] until a single point is left. This naturally gives a binary tree. We give an algorithmic definition, which gives substantial flexibility in how the tree is constructed. A tree is a nearest neighbor tree for a point set if it can be constructed by the following algorithm. T_i denotes a tree which has coordinates associated with its root.

$Collapse(S = \{T_1, T_2, \ldots, T_N)$
  if $N = 1$ **then return**
  Let $\langle T_{i_1}, T_{i_2}\rangle, \langle T_{i_3}, T_{i_4}\rangle, \ldots \langle T_{i_{2k-1}}, T_{i_{2k}}\rangle$ be mutually closest pairs.
  for $j := 1$ **to** k
    Replace $T_{i_{2j-1}}$, $T_{i_{2j}}$ by a node with coordinates corresponding to their combined center of mass and with $T_{i_{2j-1}}$, $T_{i_{2j}}$ as its children. Update the set S.
  $Collapse(S)$.

The nondeterministic choice in which mutually nearest neighbors are merged allows a wide range of trees to be constructed. The choices can be restricted to get smaller families of trees. For example, the choice could be restricted to the globally closest pair. This would make the algorithm deterministic, and also would make it easier to prove structural theorems about the trees. However, it could make the trees more difficult to construct, and would certainly make parallel construction of the trees harder. A weaker restriction would be to only combine a single pair of trees. This gives a smaller class of trees, but is a fairly natural restriction. We shall use the general definition here. It includes the classes of trees used by Benz and by Jernigan and Porter. Many generalization of this definition are possible, such as allowing several several trees to be combined instead of just pairs of trees. We defer the generalization until we have a better understanding of the pairwise combination.

[6] A pair of points x and y are mutually nearest neighbors if x is a nearest neighbor of y and y is a nearest neighbor of x. Any point set has at least one pair of mutually nearest neighbors.

The nearest neighbor scheme appears to work well in practice. The motivation for the data structure is that it should adapt to the structure of the point set, and should put clusters in the same subtree. The intuition and observed behavior is that the trees have low depth. The following heuristic arguments suggest that the trees should have small depth.

Pseudo-theorem 1.1 *A nearest neighbor tree on n points has $O(\log n)$ height.*

Argument 1: If points are randomly distributed than a constant fraction of the points will be involved in mutual nearest neighbor pairs. These can be collapsed, reducing the number of pairs by a constant fraction, leading to logarithmic depth. [11]

Argument 2: Let δ denote the distance from p to its nearest neighbor. Suppose p is merged with all points within a radius of 2δ. The nearest neighbor distance of the resulting point will be a least 2δ. The point p can be involved in at most a constant number of mergers within this radius. Since the nearest neighbor distance doubles, the height is logarithmic.

Both of these arguments have several holes, however, they do provide intuition as to why the trees should be shallow. First of all, there should be a large number of independent nearest neighbor pairs, and second of all, as particles are merged, the nearest neighbor distance should increase exponentially.

The conjecture of logarithmic height is false, since in common with the oct-tree structure, there are distributions of points which give linear depth. The bad case again is when points have exponentially increasing separations, for example, if point p_i had coordinates $(0, 0, 2^i)$. If the masses are allowed to be of exponential size, then trees of linear height can be constructed even if the positions are bounded by a polynomial. However, the cases with an exponential range of masses or positions are not of much interest. It would be of interest to prove that the tree height was logarithmic if the positions and masses were polynomially bounder, or to establish a logarithmic dependence on position and maximum mass. We believe that the following conjecture is true:

Conjecture 1.1 *Let δ denote the ratio between the maximum and minimum separation of points, and let M denote the total mass of the system. The height of any nearest neighbor tree is $O(\log \delta + \log M)$.*

This conjecture is turning out to be remarkably difficult to prove. To establish that nearest neighbor trees are good for simulation, we would like to prove the following:

Conjecture 1.2 *Let T be a near neighbor tree with total leaf depth L. The approximate n-body problem can be solved in $O(L)$ time using T.*

The reason that this conjecture is not obviously true is the the regions represented by the nodes are not necessarily disjoint, so the lemmas used for oct-trees do not apply directly.

Dimension one We have only succeeded in establishing the first conjecture for some very simple cases. We can prove that it holds in one dimension.

Lemma 3 *Let $p_1, p_2, \ldots, p_n$ be n points on a line. Let δ denote the ratio between the maximum and minimum separation of points, and let M denote the total mass of the system. The height of any nearest neighbor tree is $O(\log \delta + \log M)$.*

Proof: The advantage of having points in a line is that it is easy to characterize the intermediate nodes. Each intermediate node is the center of mass of an interval of points. Each intermediate node has a mass, a distance to its left and right neighbors. Consider a path from a leaf to the root in the tree. The distances to the left and right neighbors and the mass increase as we move up the tree. We argue that at least one of these quantities increases by a factor of at least $\frac{3}{2}$. Suppose we are at a node p' which is merged with p'' to form $\hat{p}$, and without loss of generality, p' is to the left of p''. If the mass of p'' is greater than the mass of p', then the mass of $\hat{p}$ is at least twice the mass of p'. Otherwise, $\hat{p}$ is located close to p' than to p'' and the distance from $\hat{p}$ to its right neighbor is at least $\frac{3}{2}$ the distance from p' to p''. This allows us to bound the height by $2 \log_{3/2} \delta + \log M$.

∎

We can generalize the result to a set of n points which lie on the radius of the circle. The key for the result on the circle is to show that each of the intermediate points formed is the center of mass of a set of adjacent points along the radius of the circle. The fact that the intermediate points correspond to arcs provides enough structure that the result can be proved.

Higher dimensions Unfortunately, the approach used in the one-dimensional case does not generalize to higher dimensions. For example, the proof depends upon the distance of a point to its nearest neighbor being non-decreasing. However, this is not always true, even in two dimensions. Suppose a point p at the origin has neighbors at $(-\frac{1}{2}, \frac{\sqrt{3}}{2})$ and $(\frac{1}{2}, \frac{\sqrt{3}}{2})$. The three points form an equilateral triangle with side length one. Assuming that the neighbors of p have equal weight, when they are merged they give a point at $(0, \frac{\sqrt{3}}{2})$, decreasing the distance from p to its nearest neighbor. In looking at related examples, it is possible to merge points to get a point that has distance slightly less than $\frac{\sqrt{3}}{2}$ to p. It is natural to ask how close is it possible to get a point to p, where p is situated at the origin, and initially there are no other points in the unit circle. It appears that solving this will be a major step towards proving the earlier conjectures. We conjecture that it is not possible to get an induced point within a distance of $\frac{1}{2}$ of p, but give a weaker version of the conjecture.

Conjecture 1.3 *Suppose that there are no points within a distance of one of the point p. There exists an $\epsilon > 0$ such that no point can get closer than ϵ of p as long as p is not merged with any points.*

The importance of this conjecture is that it seems to be a step in bounding the number of times a point can merge with other points, and also suggests a way of showing that after some fixed number of merges, the point free neighborhood of a point must increase.

Constructing nearest neighbor trees We have not considered the problem of how to build the nearest neighbor tree. Since the problem of finding a nearest neighbor pair in a point set can be solved in $O(n \log n)$ time[24, 25, 10], there is an $O(n^2 \log n)$ algorithm for the problem. It is very likely that the result can be improved to $O(n \log n)$ time. The parallel complexity is also of interest. There is not an obvious NC algorithm for the problem, even if the tree height is logarithmic. Since there are optimal bounds for nearest neighbors [26], it is likely that highly efficient parallel algorithms exist for constructing nearest neighbor trees.

7.3 Simulation Algorithms

The work on simulation has generally concentrated on getting a good solution to the N-body problem. However, the real problem of interest is to perform an accurate simulation over a moderately large number of time steps as opposed to just quickly and accurately evaluating forces at a single point in time. There are advantages in preserving information across time steps as well as in weakening the definition of time steps.

A simulation will have a very strong correlation between time steps. It is possible to take advantage of this in many ways. The correlation can be used in a parallel implementation to improve load balancing and to reduce memory access costs. For load balancing, it is often useful to use the execution time for a particle on the previous time step as an estimate of its execution time on the current time step. Memory behavior can be improved by relying on caching (either by the hardware, or in software) to increase locality across time steps. This can be achieved if processor allocation does not change dramatically between time steps. Another use of the correlation between time steps is to use an incremental data structure for the spatial decomposition tree instead of rebuilding from scratch. The implementation of Appel [9] and Jernigan and Porter [11] use incremental data structures. In some cases, such as oct-trees, the data structure construction is so efficient that it is not necessary to use an incremental version. The advantage of incremental structures is that they give an opportunity to reduce the cost of using a more complicated spatial data structure.

Another reason to look at simulation as opposed to just the N-body computation is to consider schemes which use different time scales. When integration is performed, it is useful to be able to adapt the time step to the rate of change of the function. In particle simulation, accuracy can be increased by using a smaller timestep for closer interactions than for far away interactions. This can be implemented by only traversing a portion of the tree during some time steps. The algorithm then becomes asynchronous since updates proceed at different rates. The use of variable time steps also impacts the choice of data structure.

The use of variable time steps has been introduced into a few astrophysical codes although the technique is not widely used. In order to take advantage of variable time steps, it will be necessary to develop general techniques which allow them to be incorporated into simulation algorithms, and also to gain a formal understanding of how they work. The theory of simulation under variable time steps is not

well developed, in fact, it is difficult to give a precise statement of the problem to be solved. The problem that the physicists want to solve is "Given a finite (but hopefully large) amount of computer time, perform as accurate a simulation as possible". Natural ways to study this problem are to fix several of the parameters, and then express a solution in terms of one or two variables. For the general problem, it is natural to treat the number of particles, the total time period, and the error threshold as variables. A force evaluation is to approximate the gravitational interaction between two sets of particles given their positions at some point in time. The general simulation problem can be expressed as "Given N, T and θ, find a set of force evaluations such that the simulation of the N particles over T units of time is within θ of the motion of the particles under exact, continuous forces." It would be necessary to define a measure of the solution being close to the exact solution, as well giving a measure of cost for the force evaluations. Thus, the final open problem, is to find the best algorithm!

8 Conclusion

There are a large number of interesting computer science problems that are motivated by astrophysical simulation. These include problems ranging from how to design easily portable scientific codes to challenging problems in computational geometry. These problems cannot be addressed by computer scientists alone, since physical scientists have a key role in problem formation and in the evaluation of solutions. The aim of this paper was to give an example of an area where collaboration between computer science and an application domain is likely to yield many interesting results. The hope is to attract computer scientists to look into these problems as well as to encourage computer scientists to look for other problem domains arising out of scientific and engineering applications.

References

[1] J. Hartmanis and Herbert Lin, editors. *Computing the Future*. National Academy Press, Washington, DC, 1992.

[2] W. Benz, R. L. Bowers, A. G. W. Cameron, and W. H. Press. Dynamic mass exchange in doubly degenerate binaries. I. 0.9 and 1.2 $M_\odot$ stars. *The Astrophysical Journal*, 348:647–667, 1990.

[3] J. K. Salmon and M. S. Warren. Astrophysical n-body simulations using hierarchical tree data structures. In *Supercomputing*, pages 570–576, 1992.

[4] R. Carlberg, 1993. Personal Communication.

[5] E. Holmberg. On the clustering tendencies among the nebulae. II. a study of encounters between laboratory models of stellar systems by a new integration procedure. *The Astrophysical Journal*, 94(3):385–395, 1941.

[6] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Adam Hilger, 1988.

[7] J. E. Barnes and P. Hut. A hierarchical $o(n \log n)$ force-calculation algorithm. *Nature*, 324:446–449, 1986.

[8] J. K. Salmon and M. S. Warren. Skeletons from the treecode closet. Technical report, Los Alamos National Laboratory, 1992.

[9] A. W. Appel. An efficient program for many-body simulation. *SIAM Journal of Scientific and Statistical Computing*, 6:85–103, 1985.

[10] P. B. Callahan and S. R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields. In *Proceedings of the 24th ACM Symposium on Theory of Computation*, pages 546–555, 1992.

[11] J. G. Jernigan and D. H. Porter. A tree code with logarithmic reduction of force terms, hierarchical regularization of all variables and explicit accuracy controls. *The Astrophysical Journal Supplement*, 71:871, 1989.

[12] J. Makino. Comparison of two different tree algorithms. *The Journal of Computational Physics*, 88:393, 1990.

[13] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[14] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. PhD thesis, Yale University, 1987.

[15] F. Zhao. An $O(n)$ algorithm for three-dimensional n-body simulations. Master's thesis, Massachusetts Institute of Technology, 1987.

[16] J. K. Salmon. *Parallel Hierarchical N-body Methods*. PhD thesis, California Institute of Technology, 1990.

[17] J. P. Singh. *Parallel Hierarchical N-body Methods and Their Implications for Multiprocessors*. PhD thesis, Stanford University, 1993. Computer Systems Laboratory Technical Report CSL-TR-93-565.

[18] J. P. Singh, W-D. Weber, and A. Gupta. SPLASH: Stanford parallel applications for shared memory. *Computer Architecture News*, 20(1):5–44, 1992.

[19] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. L Hennessy. Load balancing and data locality in hierarchical N-body methods. Technical Report CSL-TR-92-505, Stanford University, 1992. To appear in *Journal of Parallel and Distributed Computing*.

[20] S. Bhatt, M. Chen, C-Y. Lin, and P. Liu. Abstractions for parallel n-body simulations. In *Supercomputing*, 1992. Reference needs to be verified.

[21] Calvin Lin and Lawrence Snyder. A portable implementation of SIMPLE. *International Journal of Parallel Processing*, 20(5):363–401, 1991.

[22] Lawrence Snyder. Foundations of practical parallel programming languages. In *Proceedings of the Second International Conference of the Austrian Center for Parallel Computation*. Springer-Verlag, 1993.

[23] G. L. Miller and W. Thurston. Separators in two or three dimensions. In *Proceedings of the 22nd ACM Symposium on Theory of Computation*, pages 300–307, 1990.

[24] K. Clarkson. Fast algorithms for the all-nearest-neighbors problem. In *24th Symposium on Foundations of Computer Science*, pages 226–232, 1983.

[25] P. M. Vaidya. An optimal algorithm for the all-nearest-neighbors problem. In *27th Symposium on Foundations of Computer Science*, pages 117–122, 1986.

[26] P. B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *34th Symposium on Foundations of Computer Science*, 1993.