Tree codes for vortex dynamics: Application of a programming framework

Sandeep Bhatt Pangfeng Liu
Bell Communications Research, Morristown NJ 07960, and
Computer Science, Rutgers University, Piscataway NJ 08855.
Victor Fernandez Norman Zabusky
Mechanical and Aerospace Engineering, Rutgers University, Piscataway NJ 08855.

Abstract

This paper describes the implementation of a fast N-body algorithm for the study of multi-filament vortex simulations. The simulations involve distributions that are irregular and time-varying. We adapt our programming framework which was earlier used to develop high-performance implementations of the Barnes-Hut and Greengard-Rokhlin algorithms for gravitational fields. We describe how the additional subtleties involved in vortex filament simulations are accommodated efficiently within the framework.

We describe the ongoing experimental studies and report preliminary results with simulations of one million particles on the 128-node Connection Machine CM-5. The implementation sustains a rate of almost 30% of the peak machine rate. These simulations are more than an order of magnitude larger in size than previously studied using direct methods. We expect that the use of the fast algorithm will allow us to study the generation of small scales in the vortex collapse and reconnection problem with adequate resolution.

1 Introduction

Computational methods to track the motions of bodies which interact with one another, and possibly subject to an external field as well, have been the subject of extensive research for centuries. So-called "N-body" methods have been applied to problems in astrophysics, semiconductor device simulation, molecular dynamics, plasma physics, and fluid mechanics.

This paper describes the implementation of a tree code for vortex dynamics simulations, which we apply to the study of vortex collapse and reconnection [20]. We expect that the use of the fast algorithm will allow us to study the generation of small scales with adequate resolution, and make it possible to realistically simulate the flows around bodies of complex shapes encountered in engineering applications. The advantages of the fast particle vortex methods will be especially important in the high Reynolds number regime, which cannot be studied with the current grid based methods (finite difference, finite volume, and spectral) because of the resolution limitations imposed by current computers.

Computing the field at a point involves summing the contributions from each of the $N-1$ particles. The direct method evaluates all pairs of two-body interactions. While this method is conceptually simple, vectorizes well, and is the algorithm of choice for many applications, its $O(N^2)$ arithmetic complexity rules it out for large-scale simulations involving millions of particles iterated over many time steps. For vortex simulations the largest number of particles reported is around fifty thousand [3, 21, 36].

Larger simulations require faster methods involving fewer interactions to evaluate the field at any point. In the last decade a number of approximation algorithms have emerged. These algorithms exploit the observation that the effect of a cluster of particles at a distant point can be approximated by a small number of initial terms of an appropriate power series. This leads to an obvious divide-and-conquer algorithm in which the particles are organized in a hierarchy of clusters which allows the approximation to be applied efficiently. Barnes and Hut [5] applied this idea to gravitational simulations. More sophisticated schemes were developed by Greengard and Rokhlin [15] and subsequently refined by Zhao [37], Anderson [2]. Better data structures have recently been developed by Callahan and Kosaraju [9].

Several parallel implementations of the algorithms

mentioned above have been developed. Salmon [27] implemented the Barnes-Hut algorithm on the NCUBE and Intel iPSC, Warren and Salmon [34] reported experiments on the 512-node Intel Touchstone Delta, and later developed hashed implementations of a global tree structure which they report in [35, 18]. They have used their codes for astrophysical simulations and also for vortex dynamics. This paper builds on our CM-5 implementation [25] of the Barnes-Hut algorithm for astrophysical simulations and contrasts our approach and conclusions with the aforementioned efforts.

This abstract is organized as follows. Section 2 describes the application problem in some detail, and outlines the Barnes-Hut fast summation algorithm. Section 3 describes how we use our framework for building implicit global trees in distributed memory as well as methods for accessing and modifying these structures efficiently. Section 4 describes experimental results on the Connection Machine CM-5.

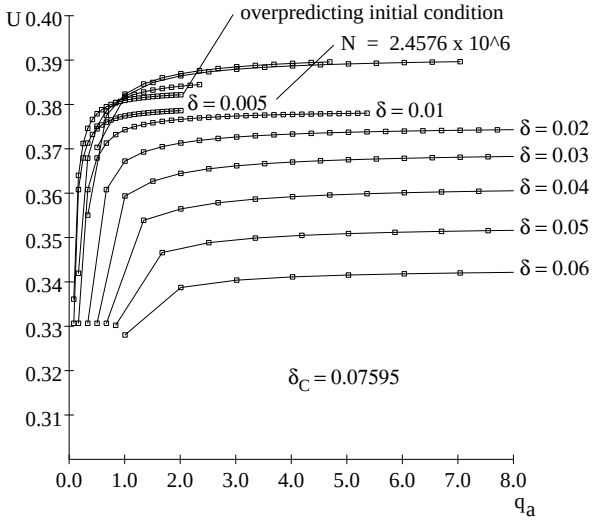

Figure 1: Induced velocity U vs. nondimensional grid size q_a.

2 Vortex methods in fluid dynamics

Many flows can be simulated by computing the evolution in time of vorticity using the Biot-Savart law. Biot-Savart models offer the advantage that the calculation effort concentrates in the small regions of the fluid that contain the vorticity, which are usually small compared to the domain that contains the flow [23]. This not only reduces considerably the computational expense, but allows better resolution for high Reynolds number flows. Biot-Savart models also allow us to perform effectively inviscid computations with no accumulation of numerical diffusion [3, 22, 23].

Vortex methods are also appropriate for studying the many complex flows that present coherent structures. These coherent structures frequently are closely interacting tube-like vortices. Some of the flows of practical interest that present interacting tube-like vortex structures are wing tip vortices [10, 19], a variety of 3D jet flows of different shapes [16, 17] and turbulent flows [11]. A generic type of interaction between vortex tubes is collapse and reconnection, which is the close approach of vortex tubes that results in large vorticity and strain-rate amplification [20]. Collapse and reconnection is an example of small scale formation in high Reynolds number flows. Small scale generation is very important for the energy transfer and dissipation processes in turbulent flows [26].

The study of small scales formation in high Reynolds number flows tends to require substantial amount of computational resources. In multi-filament models, the $O(N^2)$ nature of the computational expense of the Biot-Savart direct method (where N is the number of grid points) severely limits the vortex collapse simulations, leaving the most interesting cases of collapse beyond the cases that have been examined to date [12, 13].

Recent multi-filament simulations have raised doubts about the convergence of the results [1, 3]. Figure 1 presents a convergence study of the velocity U induced on itself by a circular vortex ring with thickness $\delta_C = 0.07595$ and radius $a = 1$. The resolution is represented (along the horizontal axis) by the nondimensional grid size $q_a = 2\delta/h_a$, where δ is the smoothing parameter in the simulation and h_a is the minimum distance between grid points. Two types of discretization schemes are employed. One underpredicts and the other overpredicts the velocity U. We observe in Figure 1 that, in order for convergence, it is necessary to have both a sufficiently high number of grid points N and a sufficiently small value of the smoothing parameter δ, in agreement with the convergence theorems [6, 14]. The small values of δ necessary for convergence make it necessary to employ millions of particles for these particular values of the ratio δ_C/δ.

2.1 The Biot-Savart model

The version of the vortex method we use was developed by Knio and Ghoniem [21]. The vorticity of a vortex tube is represented by a bundle of vortex filaments $\boldsymbol{\chi}_l(\sigma, t^*)$, each of them with circulation Γ_l. The

N_f filaments forming the bundle are advected according to the velocity field

$$\mathbf{u}(\mathbf{x}) \;=\; -\sum_{l=1}^{N_f} \frac{\Gamma_l}{4\pi} \int_C \frac{(\mathbf{x}-\boldsymbol{\chi}_l)\times d\boldsymbol{\chi}_l}{|\mathbf{x}-\boldsymbol{\chi}_l|^3}\, g(|\mathbf{x}-\boldsymbol{\chi}_l|)\;, \tag{1}$$

where $g(\rho) \;= 1 \;-\; \exp(\,-\rho^3/\delta^3\,)$.

2.2 Discretization

Each filament of the vortex ring is discretized by n_0 grid points or vortex elements. Once this is done, the order of the summations in Equation (1) is unimportant, i.e. (1) is solved numerically at N discrete points or vortex elements $\boldsymbol{\chi}_p$ by using the approximation

$$\mathbf{u}(\mathbf{x}) \;=\; -\frac{1}{4\pi} \sum_{p=1}^{N} \frac{(\mathbf{x}-\boldsymbol{\chi}_p)\times \Delta\boldsymbol{\chi}_p\;\Gamma_p}{|\mathbf{x}-\boldsymbol{\chi}_p|^3}\, g(|\mathbf{x}-\boldsymbol{\chi}_p|)\;, \tag{2}$$

where the filament ordering has to be considered in the computation of the central difference $\Delta\boldsymbol{\chi}_p$

$$\Delta\boldsymbol{\chi}_p \;=\; \frac{1}{2}\,(\boldsymbol{\chi}_{p+1}-\boldsymbol{\chi}_{p-1})\;. \tag{3}$$

This is a characteristic of the filament approach in 3D vortex methods. In contrast with the "vortex arrow" approach [18, 23], updating the "strength" of the vortex elements in the filament method does not require the evaluation of the velocity gradient, which involves the computation of another integral over all of the particles. Also, filaments with form of closed curves, satisfy the divergence free condition of the vorticity field. This is not always the case at all times in the vortex arrow approach.

The velocity field in eq. (2) can be computed more efficiently by using the multipole expansion

$$\begin{aligned}\mathbf{u}(\mathbf{x}) = -\sum_{n=0}^{m}\sum_{j+k\le n} \quad & \boldsymbol{M}(j,k,n-j-k) \\ \times \quad & D(j,k,n-j-k)\;\nabla\psi(\mathbf{x}-\boldsymbol{\chi}_0) \\ + \quad & \nabla\times\boldsymbol{\Phi}_m\;,\end{aligned} \tag{4}$$

where

$$\boldsymbol{M}(j,k,n-j-k) \;= \sum_{p=1}^{N} \boldsymbol{\alpha}_p\;(\chi_p-\chi_0)_1^j\,(\chi_p-\chi_0)_2^k\,(\chi_p-\chi_0)_3^{n-j-k}, \tag{5}$$

and

$$D(j,k,n-j-k) \;=\; \frac{(-1)^n}{(n-j-k)!\;j!\;k!}\;\partial_1^j\,\partial_2^k\,\partial_3^{n-j-k}\;. \tag{6}$$

The strength of the particles is $\boldsymbol{\alpha}_p \;=\; \Gamma_p\;\Delta\boldsymbol{\chi}_p$ and $\psi(\rho) \;=\; \int_0^\rho g(s)\,ds/s^2 \;+\; \psi(0)$. The term $\nabla\times\boldsymbol{\Phi}_m$ is the truncation error that results from using a finite number of terms in the multipole expansion (4).

2.3 Initial Conditions

The multi-filament ring is constructed around the center line

$$(\chi_1,\;\chi_2,\;\chi_3) \;=\; (a\;\cos\phi,\;b\;\sin\phi,\;c\;\sin 2\phi)\;, \tag{7}$$

where $0 \le \phi \le 2\pi$. The grid points are located at equally spaced intervals $\Delta\phi$ or at variable intervals, with the smaller intervals on the collapse region. We call this geometry the "Lissajous-elliptic" ring because of its projections on two orthogonal planes (for $c > 0$). The thickness of the multi-filament ring is δ_C. The circulation distribution Γ_i, and the initial filament core radius δ_i also need to be specified. Besides the fact that its parameter space contains cases of very rapid collapse, the low number of parameters of this configuration allows a complete parameter study at less computational expense. The case $a = b = 1$, $c = 0$ corresponds to the circular ring we use for the static studies (Figure 1). Low aspect ratio elliptic rings $a > b, c = 0$ correspond to rings with periodic behavior that can be used for dynamic and long time behavior testing of the algorithm. After thorough testing is carried out, production simulations of collapsing vortex configurations with $c > 0$ will be performed.

In order to evaluate the quality of the simulations we have also implemented different types of diagnostics. These include the motion invariants (linear and angular impulse and energy). For low aspect ratio elliptic rings we measure the period of oscillation and the deviation of the initial planar shape of the ring at each period. Other types of diagnostics have been implemented to obtain physical insight into the collapse problem. These include measurements of vorticity, strain-rate and vortex stretching. The efficient evaluation of these diagnostics requires the use of the fast summation algorithm employed in the simulation itself.

2.4 Fast summation method

To reduce the complexity of computing the sums in Equation 2, we use the Barnes-Hut algorithm. To organize a hierarchy of clusters, we first compute an oct-tree partition of the three-dimensional box (region of space) enclosing the set of bodies. The partition is computed recursively by dividing the original box into

eight octants of equal volume until each undivided box contains exactly one body. Alternative tree decompositions have been suggested [4, 9]; the Barnes-Hut algorithm applies to these as well.

Each internal node of the oct-tree represents a cluster. Once the oct-tree has been built, the moments of the internal nodes are computed in one phase up the tree, starting at the leaves. The next step is to compute induced velocities; each particle traverses the tree in depth-first manner starting at the root. For any internal node, if the distance D from the corresponding box to the particle exceeds the quantity R/θ, where R is the side-length of the box and θ is an accuracy parameter, then the effect of the subtree on the particle is approximated by a two-body interaction between the particle and a point vortex located at the geometric center of the tree node. The tree traversal continues, but the subtree is bypassed.

Once the induced velocities of all the bodies are known, the new positions and vortex element strengths are computed. The entire process, starting with the construction of the oct-tree, is repeated for the desired number of time steps.

For convenience we refer to the set of nodes which contribute to the acceleration on a particle as the *essential* nodes for the particle. Each particle has a distinct set of essential nodes which changes with time.

One remark concerning distance measurements is in order. There are several ways to measure the distance between a particle and a box. Salmon [27] discusses various alternatives in some detail. For consistency, we measure distances from bodies to the perimeter of a box in the L_∞ metric. In our experiments we vary θ to test the accuracy of the code compared to solutions computed using the direct method at specific points.

3 The framework and its application

While the physical models of N-body applications differ greatly, the computational structure is more-or-less application-independent. It is natural, therefore, to develop programming tools which reduce the time to develop high-performance application codes. This section describes our programming framework for hierarchical N-body algorithms; this framework has been used earlier for implementing the Barnes-Hut and Greengard-Rokhlin algorithms for gravitational fields. Warren and Salmon [35] have developed a different framework; we contrast the two approaches in this section.

All tree codes resolve common issues of building, traversing, updating and load-balancing large trees in distributed memory. The tension between the communication overhead and computational throughput is of central concern to obtaining high performance. The challenges can be summarized as follows.

1. The oct-tree is irregularly structured and dynamic; as the tree evolves, a good mapping must change adaptively.

2. The data access patterns are irregular and dynamic; the set of essential tree nodes cannot be predicted without traversing the tree. The overhead of traversing a distributed tree to find the essential nodes can be prohibitive unless done carefully.

3. The number of floating point operations necessary to update the position can vary tremendously between bodies; the difference often ranges over an order of magnitude. Therefore, it is not sufficient to map equal numbers of bodies among processors; rather, the work must be equally distributed among processors. This is a tricky issue since mapping the nodes unevenly can create imbalances in the work required to build the oct-tree.

The techniques we use to guarantee efficiency include: (a) exploiting physical locality, (b) an implicit tree representation which requires only incremental updates when necessary, (c) sender-driven communication, (d) aggregation of computation and communication in bulk quantities, and (e) implementing the "all-to-some" communication abstraction.

3.1 Exploiting physical locality

We use orthogonal recursive bisection (ORB) to distribute bodies among processors. The space bounding all the bodies is is partitioned into as many boxes as there are processors, and all bodies within a box are assigned to one processor. At each recursive step, the separating hyperplane is oriented to lie along the smallest dimension; the intuition is that reducing the surface-to-volume ratio is likely to reduce the volume of data communicated in later stages. Each separator divides the workload within the region equally. When the number of processors is not a power of two, it is a trivial matter to adjust the division at each step accordingly.

The ORB decomposition can be represented by a binary tree, the ORB tree, a copy of which is stored in every processor. The ORB tree is used as a map which locates points in space to processors. Storing a copy at each processor is quite reasonable when the

number of processors is small relative to the number of particles.

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. This latter property is essential for sender-directed communication of essential data, for relocating bodies which cross processor boundaries, for finding successor particles along a filament, and also for building the global BH-tree. Furthermore, ORB preserves data locality reasonably well[1] and permits simple load-balancing. While it is expensive to recompute the ORB at each time step [28], the cost of incremental load-balancing is negligible [25].

The ORB decomposition is incrementally updated in parallel as follows. At the end of an iteration, each ORB tree node is assigned a weight equal to the total number of operations performed in updating the states of particles in each of the processors which is a descendant of the node. A node is overloaded if its weight exceeds the average weight of nodes at its level by a small, fixed percentage, say 5%. We identify nodes which are not overloaded but one of whose children is overloaded; call each such node an initiator. Only the processors within the corresponding subtree participate in balancing the load for the region of space associated with the initiator. The subtrees for different initiators are disjoint so that non-overlapping regions can be balanced in parallel. At each step of the load-balancing step it is necessary to move bodies from the overloaded child to the non-overloaded child. This involves computing a new separating hyperplane; we use a binary search combined with a tree traversal on the local BH-tree to determine the total weight within a parallelpiped.

3.2 Virtual global trees

Unlike the original implementation of Warren and Salmon [34], we chose to construct a representation of a distributed global oct-tree. An important consideration for us was to investigate abstractions that allow the applications programmer to declare a global data structure, a tree for example, without having to worry about the details of distributed-memory implementation. For this reason we separated the construction of the tree from the details of later stages of the algorithm. The interested reader is referred to [7] for further details concerning a library of abstractions for N-body algorithms.

[1] Clustering techniques which exploit the geometrical properties of the distribution will preserve locality better, but might lose some of the other attractive properties of ORB.

Representation. We assign each BH-tree node an owner as follows. Since each tree node represents a fixed region of space, the owner is chosen to be the processor whose domain contains a canonical point, say the center of the corresponding region. The data for a tree node, the multipole representation for example, is maintained by the owning processor. Since each processor contains the ORB-tree it is a simple calculation to figure out which processor owns a tree node.

The only complication is that the region corresponding to a node can be spanned by the domains of multiple processors. In this case each of the spanning processors computes its contribution to the node; the owner accepts all incoming data and combines the individual contributions. This can be done efficiently when the combination is a simple linear function, as is the case with all tree codes.

Construction. Each processor first builds a local oct-tree for the bodies which are within its domain. At the end of this stage, the local trees will not, in general, be structurally consistent. For example, Figure 2 shows a set of processor domains that span a node; the node contains four bodies, one per processor domain. Each local tree will contain the node as a leaf; but this is inconsistent with the global tree.

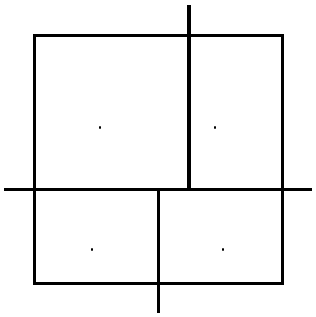

Figure 2: An internal node which appears as a leaf in each local tree.

The next step is to make the local trees be structurally consistent with the global oct-tree. This requires adjusting the levels of all nodes which are split by ORB lines. We omit the details of the level-adjustment procedure in this paper. A similar process was developed independently in [28]; an additional complication in our case is that we build the oct-tree until each leaf contains a number, L, of bodies. Choosing L to be much larger than 1 reduces the size of the BH-tree, but makes level-adjustment somewhat tricky.

The level adjustment procedure also makes it easy to update the oct-tree incrementally. We can insert and delete bodies directly on the local trees because we do not explicitly maintain the global tree. After the insertion/deletion within the local trees, level adjustment restores coherence to the implicitly represented distributed tree structure.

3.3 Computing multipole moments

For gravitational simulations once level-adjustment is complete, each processor computes the centers-of-mass and multipole moments on its local tree. This phase requires no communication. Next, each processor sends its contribution to a node to the owner of the node. Once the transmitted data have been combined by the receiving processors, the construction of the global oct-tree is complete. This method of reducing a tree computation into one local step to compute partial values, followed by one communication step to combine partial values at shared nodes is widely useful.

For multi-filament vortex dynamics, computing the vortex element strengths requires an extra phase in the tree construction. The vortex element strength of a particle is defined as the displacement of its two neighbors along the filament (equation 3). Once the vortex element strengths are computed, we compute the multipole moments on the local trees. Finally, each processor sends its contribution to a node to the owner of the node so that individual contributions are combined into globally correct information.

Finding neighbors along a filament requires communication when the neighbors are owned by other processors. To efficiently determine which particles have missing neighbors and require communication, each processor needs to maintain a linear ordering of particles in the same filament within its domain. Once the linear ordering is computed, a processor can go through the list to compute the vortex element strength, and request information from missing neighbors.

The linear ordering must be maintained dynamically as particles move across processor boundaries, either because of their induced velocity or during workload remapping. We use a *splay search tree* [29] to store the indices of particles of the same filament. When particles enter/leave a processor domain, its filament number and index within the filament (f, i) are sent to the new processor, so that i can be inserted into the search tree for filament f in the destination. We use a splay tree implementation because of its simplicity and self-adjusting property.

3.4 Sender-driven communication

Once the global oct-tree has been constructed it is possible to start calculating accelerations. For Barnes-Hut algorithm each processor can compute the acceleration for its particles by traversing the oct-tree, and requesting essential tree data from owners whenever it is not locally available.

This naive strategy of traversing the tree, and transmitting data-on-demand, has several drawbacks: (1) it involves two-way communication, (2) the messages are fine-grain so that either the communication overhead is prohibitive or the programming complexity goes up, and (3) processors can spend substantial time requesting data for tree nodes that do not exist.

It is significantly easier and faster to first construct the locally essential trees. The owner of a node computes the destination processors for which the node might be essential; this involves the intersection of the annular region of influence of the node with the ORB-map. Each processor first collects all the information deemed essential to other nodes, and then sends long messages directly to the appropriate destinations. Once all processors have received and inserted the data received into the local tree, all the locally essential trees have been built.

Sender-driven communication is used not only in transferring essential data but also in all the other communication phases. This is true for the Greengard-Rokhlin algorithm as well [8]. For example, to compute the vortex element strength for each particle in the multi-filament vortex method, its two neighboring particles in the filament must be fetched through communication if they are not in the local tree. Since the data dependency is symmetrical for the computation, the owner of a particle can easily determine if this particle should be sent out. By maintaining an upper bound on the maximum distance between two neighboring particles, it becomes possible to locate the neighbors of a particle quickly.

3.5 Bulk processing

We separate control into a sequence of alternating computation and communication phases. This helps maintain simple control structure; efficiency is obtained by processing data in bulk. For example, up to a certain point, it is better to combine multiple messages to the same destination and send one long message. Similarly, it is better to compute the essential data for several bodies rather than for one at a time. Another idea that proved useful is sender-directed communication, send data wherever it might

be needed rather than requesting it whenever it is needed. Indeed, without the use of the CM-5 vector units we found that these two ideas kept the overhead because of parallelism minimal.

3.6 All-to-some communication

The communication phases can all be abstracted as the "all-to-some" problem. Each processor contains a set of messages; the number of messages with the same destination can vary arbitrarily. The communication pattern is irregular and unknown in advance. For example, level adjustment is implemented as two separate all-to-some communication phases. The phase for constructing locally essential trees uses one all-to-some communication.

The first issue is detecting termination: when does a processor know that all messages have been sent and received? The naive method of acknowledging receipt of every message, and having a leader count the numbers of messages sent and received within the system, proved inefficient.

A better method is to use a series of global reductions the control network of the CM-5 to first compute the number of messages destined for each processor. After this the send/receive protocol begins; when a processor has received the promised number of messages, it is ready to synchronize for the next phase.

We noticed that the communication throughput varied with the sequence in which messages were sent and received. As an extreme example, if all messages are sent before any is received, a large machine will simply crash when the number of virtual channels has been exhausted. In the CMMD message-passing library (version 3.0) each outstanding send requires a virtual channel [32] and the number of channels is limited.

Instead, we used a protocol which alternates sends with receives. The problem is thus reduced to ordering the messages to be sent. For example, sending messages in order of increasing destination address gives low throughput since virtual channels to the same receiver are blocked. In an earlier paper [24] we developed the atomic message model to investigate this phenomenon. Consistent with the theory, we find that sending messages in random order worked best.

3.7 Application to vortex simulation

We implemented the multi-filament vortex simulation using our framework. Figure 3 gives a high-level description of the code structure. Most of the code was taken directly from our earlier gravitational simulation code. For example, the ORB decomposition, load remapping, communication protocol, and the BH tree module are exactly the same. The only modifications required for the vortex simulation are the extra phase (step 1.2) to compute the vortex element strength of each particle, and the dynamic update of filament-trees which contain indices of particles in one filament (step 5).

```
0. build local BH trees
   for every time step do:
1.  construct the BH-tree representation
1.1  adjust node levels
1.2  compute the strength of each particle
1.3  compute partial node values on local trees
1.4  combine partial node values at owners
2.  owners send essential data
3.  calculate induced velocity
4.  update velocities and positions of bodies
5.  update incremental data structures:
      local BH-tree and filament-trees
6.  if the workload is not balanced
      update the ORB incrementally
   enddo
```

Figure 3: Outline of code structure

Step 0 builds a local BH tree in each processor. After that the local BH trees are maintained incrementally and never rebuilt from scratch. Step 1 constructs an implicit representation of the global BH tree by combining information in the local trees. Step 5 updates the local trees after particles move to their new positions.

Step 2 broadcasts essential information and is the most communication intensive phase. The sender-oriented communication avoids two-way traffic in fetching remote data, and aggregates messages for higher communication throughput.

Step 3 computes the induced velocity for each vortex and is most time-consuming. We use the Knio-Ghoniem algorithm to solve the Biot-Savart equations, using monople, dipole and quadrapole terms in the multipole expansion to approximate the effect of a cluster. We use CDPEAC assembly language to implement this computation intensive routine. One significant optimization was to approximate the exponential smoothing term in equation 1. When ρ/δ is larger than 20 we ignore the term; for smaller values we compute the first seven terms in the Taylor expansion. This guarantees accuracy to within at least five

decimal places.

Step 6 redistributes particles when workload becomes imbalanced. The ORB is incrementally updated so that workload is balanced among processors.

3.8 Comparisons with previous work

We sketch the important differences between our framework and the global hashed structure of Warren and Salmon [35, 18]. Warren and Salmon [35] use a different criterion for applying approximations. They do not construct essential trees thereby saving the space overhead associated with local essential trees. Instead, they construct an explicit representation of the oct-tree. Each body is assigned a key based on its position, and bodies are distributed among processors by sorting the corresponding keys. Besides obviating the need for the ORB decomposition, this also simplifies the construction of the oct-tree.

These advantages are balanced by other factors: (1) the advantages of sender-directed communication are lost, (2) the data structures are not maintained incrementally, but rebuilt from scratch each iteration, and (3) the computation stage is slowed down by communication; the latency is largely hidden by multiple threads to pipeline tree traversals and to update accelerations, but the program control structure is complicated and less transparent. Finally, it is not clear how hashed oct-trees can be extended gracefully to multi-filament simulations where each filament must maintain a cyclic ordering. Indeed, their fluid dynamics simulation involves only "free" particles (vortex arrows).

While our vortex simulations are very different from those of [18] to make meaningful comparisons, the experiments for gravitational simulations are quite similar. Our performance figures compare favorably with those reported by Warren and Salmon [34, 35]. With 10 million particles distributed uniformly in space, the total time per iteration for our code is 27 seconds on a 256 node CM-5; less than 30% of the time is devoted to managing the Barnes-Hut tree, constructing locally essential trees, preparing essential data via caching, and balancing workload. In contrast, the time per iteration for a uniform distribution of 8.8 million particles on the 512-node Delta Touchstone reported by Warren and Salmon is 77 seconds [34] and 114 seconds [35] using the hashed data structure. In the latter implementation over 53% of the time is spent constructing and manipulating the tree structure.

4 Performance

Our platform is the Connection Machine CM-5 with SPARC vector units [30]. Each processing node has 32M bytes of memory and can perform floating point operations at peak rate of 128 Mflop/s [33]. We use the CMMD communication library [32]. The vector units are programmed in CDPEAC which provides an interface between C and the DPEAC assembly language for vector units [31]. The rest of the program is written in C.

The vortex dynamics code along with several diagnostics are operational. Starting with the gravitational code, the effort involved adding the phase to compute vortex element strength and writing the computational CDPEAC kernel. As expected, the latter part took most of the effort; the data structures were adaptedand debugged in less than a week.

We are currently running extensive tests to check the accuracy of the Barnes-Hut method as a function of the approximation parameter θ. Preliminary results show that with quadrupole expansions we can obtain up to five digits of accuracy when θ is about 0.25, and the time step is 0.002.

phase	time (second)
adjust level	1.407
compute vortex strength and moments	1.435
construct global tree	0.317
collect local essential data	15.873
compute the new velocity	307.221
update particle position	0.144
move particles into right processors	1.720
remap the workload	0.141
total time	328.258

Figure 4: Timing breakdown for vortex simulation.

Figure 4 shows the timing breakdown for a 3D vortex simulation on a 128-node CM-5. The input configuration is a 22 layers, 1012 filament vortex ring[2]. Each filament has 1024 particles and the total number of particles is about one million. The radius of the cross section of the vortex ring is 0.07595 and the core radius is 0.06. We use $\theta = \frac{1}{4}$ in the opening test for multipole approximation so that the error in total induced velocity is within 0.00001. The vector units compute the induced velocity at 54 Mflop/sec, and the simulation sustains an overall rate of over 37

[2]We use $a = 1$, $b = 0.6$, and $c = 0$ in Equation 7.

Mflop/sec per node (averaged over 5 time steps, not including the first).

Since the locally essential trees overlap considerably, the total size of local trees exceeds the size of the global tree. Figure 5 shows the memory overhead (ratio of total distributed memory used divided by global tree size) versus granularity (number of particles per node) on a 32-node CM-5. This ratio is less than 2.4 as the granularity exceeds 10000 particles per processors, even for very small $\theta = 0.25$.

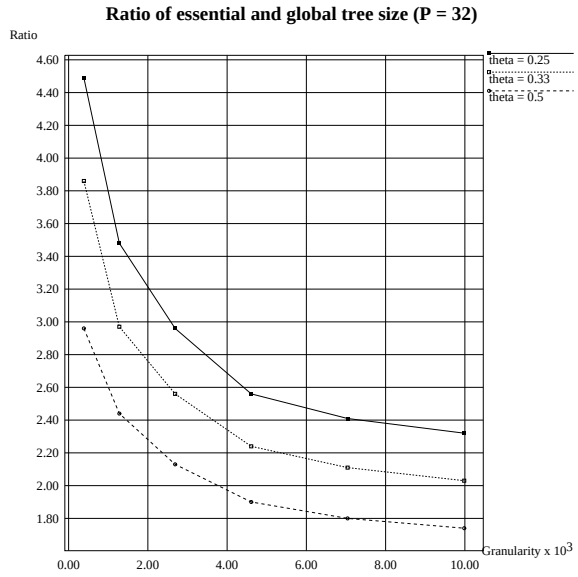

Figure 5: Memory overhead versus granularity for different θ.

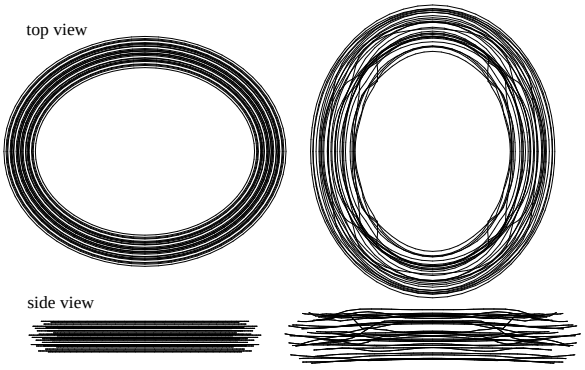

Figure 6: Low aspect ratio elliptic ring oscillates between the states shown. The ring has $N = 15360$ particles. The last time shown is after 1000 time steps $\Delta t = 6.28\text{x}10^{-3}$.

As part of the initial testing, we have examined two cases. The first is a low aspect ratio elliptic ring with $a = 1, b = 0.8$ and $c = 0$ (see Eq. 7). This ring presents an oscillating behavior between the two states presented in Fig 6. The ring was discretized with $N_f = 60$ filaments with a total number of particles $N = 15360$. The final state in Fig 6 is after 1000 time steps $\Delta t = 6.28\text{x}10^{-3}$. The second case tested corresponds to a collapsing vortex ring with $a = 1, b = 0.4$ and $c = 0.5$. The initial condition and the collapse state are shown in Fig 7. This ring has $N_f = 220$ filaments with a total number of particles $N = 112640$. The final state corresponds to 2000 time steps $\Delta t = 10^{-3}$.

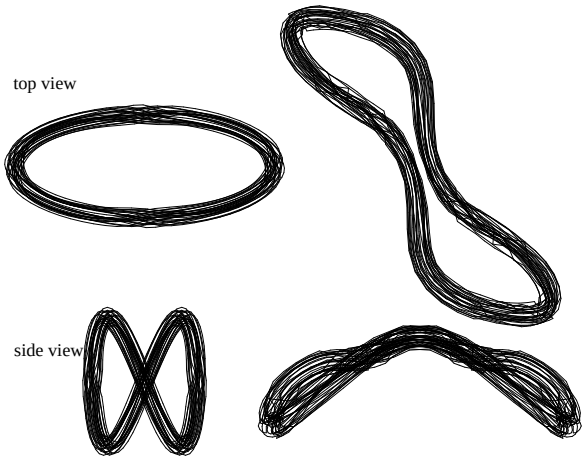

Figure 7: Collapsing vortex ring. This ring is discretized with $N = 112640$ particles. The collapse state shown is obtained by 2000 time steps $\Delta t = 10^{-3}$.

Acknowledgments

We thank Apostolos Gerasoulis of Rutgers for helping set up the collaboration and helpful discussions. This work is supported in part by ONR Grant N00014-93-1-0944 and ARPA contract DABT-63-93-C-0064 "Hypercomputing & Design." The content of the information herein does not necessarily reflect the position of the Government and official endorsement should not be inferred.

The code was developed and tested on the CM-5 at the Naval Research Laboratories, the National Center for Supercomputer Applications, University of Illinois at Urbana-Champaign, and the Minnesota Supercomputing Center.

References

[1] A. S. Almgren, T. Buttke, and P. Colella. A fast adaptive vortex method in three dimensions.

Journal of Computational Physics, 113:177–200, 1994.

[2] C. Anderson. An implementation of the fast multipole method without multipoles. *SIAM Journal on Scientific and Statistical Computing*, 13, 1992.

[3] C. Anderson and C. Greengard. The vortex merger problem at infinite Reynolds number. *Communications on Pure and Applied Mathematics*, XLII:1123–1139, 1989.

[4] R. Anderson. Nearest neighbor trees and N-body simulation. *manuscript*, 1994.

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

[6] J. T. Beale and A. Majda. Vortex methods. I: Convergence in three dimensions. *Mathematics of Computation*, 39(159):1–27, 1982.

[7] S. Bhatt, M. Chen, J. Cowie, , C. Lin, and P. Liu. Object-oriented support for adaptive methods on parallel machines. In *Object Oriented Numerics Conference*, 1993.

[8] S Bhatt and P. Liu. A framework for parallel n-body simulations. *manuscript*, 1994.

[9] P. Callahan and S. Kosaraju. A decomposition of multi-dimension point-sets with applications to k-nearest-neighbors and N-body potential fields. *24th Annual ACM Symposium on Theory of Computing*, 1992.

[10] S. C. Crow. Stability theory for a pair of trailing vortices. *AIAA Journal*, 8(12):2172–2179, 1970.

[11] S. Douady, Y. Couder, and M. E. Brachet. Direct observation of the intermittency of intense vorticity filaments in turbulence. *Physical Review Letters*, 67(8):983–986, 1991.

[12] V. M. Fernandez. *Vortex intensification and collapse of the Lissajous-Elliptic ring: Biot-Savart simulations and visiometrics.* Ph.D. Thesis, Rutgers University, New Brunswick, New Jersey, 1994.

[13] V.M. Fernandez, N.J. Zabusky, and V.M. Gryanik. Near-singular collapse and local intensification of a "Lissajous-elliptic" vortex ring: Non-monotonic behavior and zero-approaching local energy densities. *Physics of Fluids A*, 6(7):2242–2244, 1994.

[14] C. Greengard. Convergence of the vortex filament method. *Mathematics of Computation*, 47(176):387–398, 1986.

[15] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73, 1987.

[16] H. S. Husain and F. Hussain. Elliptic jets. Part 3. Dynamics of preferred mode coherent structure. *Journal of Fluid Mechanics*, 248:315–361, 1993.

[17] F. Hussain and H. S. Husain. Elliptic jets. Part 1. Characteristics of unexcited and excited jets. *Journal of Fluid Mechanics*, 208:257–320, 1989.

[18] M. Warren J. Salmon and G. Winckelmans. Fast parallel tree codes for gravitational and fluid dynamical N-body problems. *Intl. J. Supercomputer Applications*, 8.2, 1994.

[19] R. T. Johnston and J. P. Sullivan. A flow visualization study of the interaction between a helical vortex and a line vortex. *Submitted to Experiments in Fluids*, 1993.

[20] S. Kida and M. Takaoka. Vortex reconnection. *Annu. Rev. Fluid Mech.*, 26:169–189, 1994.

[21] O. M. Knio and A. F. Ghoniem. Numerical study of a three-dimensional vortex method. *Journal of Computational Physics*, 86:75–106, 1990.

[22] A. Leonard. Vortex methods for flow simulation. *Journal of Computational Physics*, 37:289–335, 1980.

[23] A. Leonard. Computing three-dimensional incompressible flows with vortex elements. *Ann. Rev. Fluid Mech.*, 17:523–559, 1985.

[24] P. Liu, W. Aiello, and S. Bhatt. An atomic model for message passing. In *5th Annual ACM Symposium on Parallel Algorithms and Architecture*, 1993.

[25] P. Liu and S. Bhatt. Experiences with parallel n-body simulation. In *6th Annual ACM Symposium on Parallel Algorithms and Architecture*, 1994.

[26] A.J. Majda. Vorticity, turbulence and acoustics in fluid flow. *SIAM Review*, 33(3):349–388, September 1991.

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

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

[29] R.E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, 1983.

[30] Thinking Machines Corporation. *The Connection Machine CM-5 Technical Summary*, 1991.

[31] Thinking Machines Corporation. *CDPEAC: Using GCC to program in DPEAC*, 1993.

[32] Thinking Machines Corporation. *CMMD Reference Manual*, 1993.

[33] Thinking Machines Corporation. *DPEAC Reference Manual*, 1993.

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

[35] M. Warren and J. Salmon. A parallel hashed oct-tree N-body algorithm. In *Proceedings of Supercomputing*, 1993.

[36] G. S. Winckelmans. *Topics in vortex methods for the computation of three- and two-dimensional incompressible unsteady flows.* Ph.D. Thesis, California Institute of Technology, Pasadena, California, 1989.

[37] F. Zhao. An $O(N)$ algorithm for three dimensional N-body simulation. Technical report, MIT, 1987.