Journal of the Islamic University of Gaza, V. 10 no.1 P43-P73, 2002

DISTRIBUTED TREE CODE ON CLUSTER OF WORKSTATIONS

MOHAMMAD A. MIKKI*
Islamic University –Gaza
P.O. Box 108, Gaza, Palestine
mmikki@mail.iugaza.edu

تنفيذ الشيفرة الشجرية التوزيعية على مجموعة محطات عمل

Abstract **In this paper, we design four performance BH algorithm optimization techniques to the distributed versions of the that run on clusters of workstations and use message passing communication model. The first technique partitions the data and balances the load among the processors so that the algorithm becomes fully distributed with no initialization overhead. The second method uses pipelining and processor domain partitioning to enhance the overlapping between force computation and processor communication. It also makes communication asynchronous to minimize processor waiting time. The third method changes the processor communication model from peer-to-peer to master-slave. This change minimizes the total communication overhead. The fourth method uses one tree traversal for computing the force on all particles in the processor domain instead of traversing the tree once for each particle. This method reduces the run time very significantly. Our code is written in C++ and uses MPI (Message Passing Interface) functions. MPI functions are supported on almost all parallel machines so the code is portable to different platforms. We use simulation experiments to measure the efficiency of our performance optimization techniques. From these experiments we measure various performance metrics such as speedup, system efficiency, total execution time, and communication overhead. The proposed optimization techniques achieve 10-45% of performance optimization compared to the non-optimized distributed BH algorithms. Our code is far more flexible and efficient than the current existing code used to solve the n-body problem. Although our algorithm is not unique, it is a robust, scalable, load balancing and fault–tolerant algorithm. It complements, enhances and extends the previous work done in this field.**

* An assistant professor at the electrical and computer engineering department in the Islamic University of Gaza, Gaza, Palestine

DISTRIBUTED TREE CODE ON CLUSTER

ملخص في هذا البحث نقوم بتصميم اربع طرق لتحسين أداء الاصدارة التوزيعية لخوارزميةBH. هذه الطرق يتم تنفيذها على مجموعة من محطات العمل و تقوم باستخدام نموذج الاتصال عبر تمرير الرسائل. الطريقة الاولى تقوم بتجزئة البيانات و موازنة الحمل على المعالجات بحيث تصبح الخوارزمية توزيعية بشكل كامل و بدون حمل زائد أولي. الطريقة الثانية تستخدم التقنية الانبوبية و تجزئة مجالات المعالجات لتحسين التداخل بين حسابات القوة و اتصالات المعالجات. هذه الطريقة تقوم أيضا بجعل اتصالات المعالجات غير متزامنة لتقليل زمن انتظار المعالجات. الطريقة الثالثة تقوم بتغيير نموذج اتصال المعالجات من الند للند الى الخادم العميل. هذا التغيير يقلل من الحمل الزائد الكلي للاتصال بين المعالجات. الطريقة الرابعة تقوم بالتجوال مرة واحدة في شجرة الاجسام لحساب القوة المؤثرة على جميع الاجسام التابعة لمجال معالج واحد بدلا من التجوال مرة لكل جسم. هذه الطريقة تقلل من زمن تنفيذ البرنامج بشكل كبير.

لقد تم استخدام لغة C++ لكتابة شيفرة البرامج التي تم بواسطتها تصميم الطرق الاربعة، كما تم ايضا استخدام دوال مكتبة البرامج MPI (الاتصال بواسطة تمرير الرسائل). ان دوال مكتبة MPI مدعومة من قبل جميع الحاسبات المتوازية و التوزيعية تقريبا، لذلك فان الطرق المقترحة يمكن نقلها و تنفيذها على مختلف نماذج الحاسبات المتوازية و التوزيعية.

لقد قمنا بتصميم تجارب محاكاة لقياس فاعلية التقنيات الاربعة المقترحة. و باستخدام هذه التجارب قمنا بقياس مختلف معايير الفاعلية مثل السرعة، فعالية النظام، الزمن التنفيذي الكلي، و الحمل الزائد للاتصال. ان التقنيات المقترحة تقوم بتحسين الكفاءة من 10-45% مقارنة بخوارزمية الشيفرة الشجرية التوزيعية التي لا تستخدم تلك التقنيات. ان البرامج التي قمنا بتصميمها مرنة و فعالة بدرجة اكبر بكثير من الخوارزميات الحالية المستخدمة لحل مشكلة الاجسام ذات العدد س. رغم ان الخوارزميات المقترحة ليست فريدة الا انها متينة، قابلة للتوسع، موازنة للحمل، و متحملة للاخطاء. انها تقوم بتدعيم و استكمال الاعمال السابقة في هذا المجال.

INTRODUCTION

The computation of the trajectories of N particles in physical systems over time due to mutual force given the gravitational forces and underlying physical laws among them and the initial conditions (position, velocity, mass) is called the N-body problem. The physical laws are usually described in partial differential equations. This problem is studied in physics, smoothed particle hydrodynamics, fluid mechanics, chemistry, plasma physics, molecular dynamics, and semiconductor device simulation. N-body problem has also been discussed as a pure computer science problem [Bhatt 1992, Singh 1993, Singh 1995].

The process by which galaxies form is undoubtedly among the most important unsolved problems in physics. **Figure 1** shows a general view of interacting galaxies. Astrophysical N-body problem represents a formidable challenge for parallel computation. The difficulties stem from some fundamental properties of the problem [Warren 1992] such as:

1- The distribution of bodies is highly non-uniform and is dynamic.
2- The data structures are adaptive and moderately complicated.
3- Each body needs both global and local data for its update.

Figure 1: General view of interacting galaxies

The complexity of the problem arises from that each of the particles exerts a force on all other particles. Given the initial condition of N-bodies, the underlying physical laws between them, in the simple algorithms that solve the N-body problem the computation time is divided into time steps, the forces on each particle are computed at each time step and are used to update the particle position and velocity at the beginning of the next time step. For the simple algorithms that apply direct summation of the contribution of all particles to calculate the force on each particle the time complexity of the N-body problem is O (N^2) which becomes prohibitively computationally large for N larger than a few tens of thousands even on the fastest parallel supercomputers. In these algorithms the force on each particle is calculated by considering all force exerted by all N –1 other particles. The number of particles used for studying the astrophysics problem exceeds 10^8 particles in many cases, so the

computational efficiency is of great concern. Tree code simulations running on the latest workstations and vector supercomputers are generally restricted to N $< 10^6$ because of memory and time limitations [Dubinski 1996].

In astrophysics, some small error in force calculation is tolerated to improve performance by using some techniques including using fast Fourier methods to resolve the long range forces, breaking the simulation into sub-domains and taking multiple expansions of the forces from these domains, and using tree codes. In tree code algorithms an adaptive tree-like data structures are used to form hierarchical clusters of particles where the entire set of particles is recursively divided into groups of equal number of particles until each group contains at most one particle (or a given number of particles). The algorithms approximate the force exerted on a particle by an equivalent mass located at the center of mass of that cluster. The time complexity of tree code algorithms is O (N log N) [Wang 1995] and these algorithms behave appropriately even with the very large ranges of density spanned in cosmological simulations.

The force on a particle is then calculated by searching the tree in a depth-first tree traversal. To compute the accelerations, each particle traverses the tree starting from the root, trying to find clusters for which the center of mass approximation applies. If the distance between the particle and the center of mass of the cluster is far enough, with respect to the radius of the cluster, then the acceleration due to that cluster is approximated by a single interaction between the particle and a point mass located at the center of the cluster, i.e., if

$$r_{cm} \geq \frac{r}{\theta} \qquad \textit{where} \quad 0 \leq \theta \geq 1 \qquad \textit{Equation (1)}$$

Where r_{cm} is the distance between a particle and the center of mass of a cluster

r is the radius of the cluster

θ is a fixed tolerance parameter

then the center of mass approximation applies and the internal distribution of the particles in the cluster is neglected. We can adjust θ to balance the approximation error and the execution time [Liu 1997]. **Equation 1** represents the tolerance criterion approximation that makes the complexity of the force computation O (N log N) rather than O (N^2). Tree code algorithms are significantly faster than the direct N-body algorithms because of the approximation technique they use for force calculation.

The physical laws that govern the forces between the particles in the astrophysical simulations of the N-body problem are formulated in the following differential equations:

$\vec{v}_i = \frac{d\vec{x}_i}{dt}$, $\vec{v}_i$ *is the velocity of particle* i *where* $\vec{x}_i$ *is the position of particle* i

$\vec{a}_i = \frac{d\vec{v}_i}{dt} = \frac{d^2\vec{x}_i}{dt^2}$, $\vec{a}_i$ *is the acceleration of particle* i

$\vec{F}_{ij} = \frac{Gm_i m_j \vec{r}_{ij}}{|r_{ij}|^3}$, $\vec{F}_{ij}$ *is the force between particles* i *and* j *where* G *is a constant*,

m_i *and* m_j *are the masses of particles* i *and* j *respectively and* $\vec{r}_{ij} = \vec{x}_j - \vec{x}_i$

$\vec{F}_i = \sum_{j,i\neq j} \vec{F}_{ij} = m_i\vec{a}_i = m_i \frac{d\vec{v}_i}{dt} = m_i \sum_{j,i\neq j} \vec{a}_{ij}$ *where* $\vec{F}_i$ *is the total force on particle* i

$\vec{F}_i = \sum_j \frac{Gm_i m_j \vec{r}_{ij}}{|r_{ij}|^3}$ (*Newtoinan gravity*)

Several implementations of the tree code have been developed [Greengard 1987, Barnes 1986, Appel 1985]. One popular tree-code algorithm is BH algorithm which uses multipole expansion and hierarchical data structures to reduce the complexity of the N-body problem. Multipole expansion allows one to treat a collection of bodies as a point mass located at the center of mass as shown in **Figure 2**. Quantitatively,

$\vec{F}_i = \sum_j \frac{Gm_i m_j \vec{r}_{ij}}{|r_{ij}|^3} \approx \frac{Gm_i m_{cm} \vec{r}_{cm}}{|r_{cm}|^3}$

where m_{cm} *is the equivalent mass of the cluster and* r_{cm} *is the distance between particle* i *and* m_{cm}

```
1. all_particles = ReadParticleInfo(in_file)
2. Loop for the desired number of time steps
2.1 T = BuildBHTree(all_particles)
2.2 ComputeCenterOfMass(T)
2.3 For every particle pt in all_particles do
2.3.1 CalcForces(T, pt)
2.3.2 UpdateParticleInfo(pt)
3. End loop
```

Figure 3: The Barnes-Hut algorithm

s-

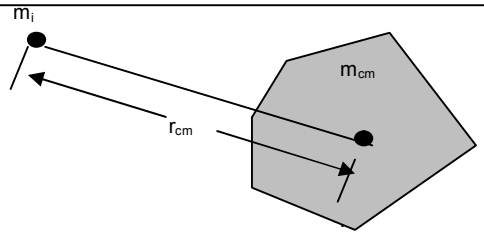
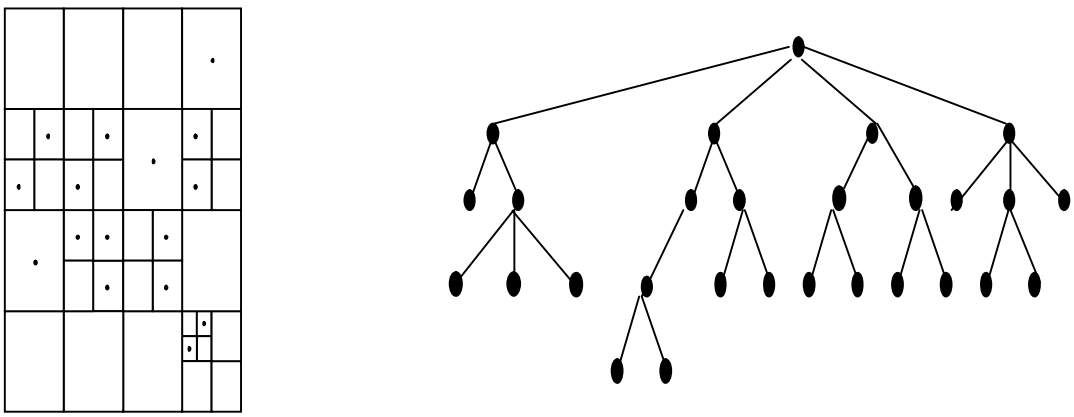


Figure 2: Center of mass approximation

DISTRIBUTED TREE CODE ON CLUSTER

In the BH algorithm of **Figure 3**, the ReadParticleInfo procedure reads the data of the particles (the initial conditions such as mass, position and velocity) from an input file. The BuildBHTree procedure builds an oct-tree partition of the region of space enclosing the set of particles. An example of a recursive tree partition in two dimensions and the corresponding BH-tree are shown in **Figure 4**. The tree is constructed anew at the beginning of each time step before forces can be calculated. The partition is computed recursively by dividing the original space into eight octants of equal volume until each undivided area contains exactly one particle. The root of the octal tree corresponds to a cube that encloses all the bodies in the simulation. Non-terminal nodes in the tree have up to eight children, corresponding to eight cubes that result from cutting the parent in half in each dimension. Any node containing one body is a leaf. The Compute Center Of Mass procedure calculates the center of mass of the internal nodes in the BH tree in a bottom-up fashion starting from the leaves. Each internal node in the tree contains the total mass of the bodies in it and their center of mass and their quadrupole moments. The CalcForces procedure calculates the force and acceleration on any particle by traversing the tree in a depth-first tree traversal. To compute the accelerations, each particle traverses the tree starting from the root, trying to find clusters for which the center of mass approximation in (1) applies. The particles visited in the traversal form a sub-tree of the entire BH tree and different particles in general traverse different sub-trees [Liu 1997]. The UpdateParticleInfo procedure calculates the particle new position and velocity and updates the particle information accordingly.



(a) Recursive tree partitioning in 2 dimensions of a region of space enclosing set of particles

(b) The corresponding BH tree of Figure 4 (a)

Figure 4: BH tree decomposition

Traversing the tree by each particle to calculate the force takes O (log N) time, and computing the forces on all N bodies requires N tree traversals and hence the BH algorithm time complexity is O (N log N).

To measure the relative cost of each step in the sequential BH algorithm, a number of runs were performed to estimate the efficiency of the sequential BH algorithm and the fraction of time each step of the algorithm takes. In **Table 1** we show the time costs of the main steps of the sequential BH algorithm when run on a single workstation for 16,000 particles and 1000 time steps. As the **Table 1** shows, most of the computation is incurred by CalcForces procedure which is about 96% of the total execution time of the algorithm. This alerts us that performance optimization techniques should focus on this step. Tree construction requires O(N log N) steps and typically requires less than 2% of the total execution time. Parallel and distributed versions of the sequential BH algorithm have been designed. These algorithms do not fully exploit parallelism in the BH algorithm and they need some performance optimization. In this paper we design four optimization techniques to the distributed versions of the BH algorithm that run on clusters of workstations and use message passing communication model.

Table 1: Time costs of the procedures in the sequential BH algorithm

Procedure	Time (sec)	Time percentage (%)
ReadParticleInfo	2	0.022
BuildBHTree	238	2.621
ComputeCenterOfMass	93	1.024
CalcForces	8695	95.759
UpdateParticleInfo	52	.572
Total time	9080	

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 presents our performance optimization techniques. Section 4 presents the simulation and experimental results. And finally, section 5 concludes the paper.

1. RELATED WORK

In this section we present related work in the area of N body problem and tree code. [Barnes 1986] developed a new tree code algorithm; the BH algorithm to

solve the N-body problem. The algorithm works by grouping particles using a hierarchy of cubes arranged in an oct-tree structure. The force on particles is calculated by doing a depth-first tree traversal, truncating at internal nodes where approximation is applicable. [Dubinski 1996] implements a parallel N-body tree code by making load balancing using orthogonal recursive bisection to subdivide the N-body system into independent rectangular volumes each of which is mapped to a processor on the Cray T3D. [Salmon 1990] implements a parallel version of the Barnes-Hut N-body algorithm by assembling a tree data structure which represents the distribution of the bodies. They run the code on Ncube system. Domain decomposition is used to assign regions of space to processors. An adaptive load balancing technique is used. A tree is built in each processor and after $\log_2 N_{proc}$ exchanges of data each processor has a restricted version of the tree which is sufficient for force calculation on the bodies which lie within its spatial domain. [Warren 1992] discusses techniques to parallelize the astrophysical N-body simulations. It reports N-body simulations executed on the Intel Touchstone Delta system. The methods scale as O(N log N) for large N, and also scale linearly with the number of processors. [Wang 1995] develops an optimized program for the N-body problem on the CM-5 with vector units. The work makes use of the power of the vector pipelines provided by the CM-5 equipped with vector units to improve the computation performance. [Liu 1997] describes an implementation of a platform-independent parallel C++ N-body framework that can support various scientific simulations that involve tree structures. Within the framework the users are able to concentrate on the computation kernels that differentiate different N-body problems, and let the framework take care of the tedious details that are in common among the N-body applications. This framework is based on the techniques learned from CM-5 C implementations. [Greengard 1987] developed the fast multipole method with O (N) arithmetic complexity under uniform particle distribution. [Sundaram 1993] extended the method used in [Greengard 1987] to allow different bodies to be updated at different rates. Because of the complexity and overhead in the fully adaptive three dimensional multipole method, the BH algorithm continues to enjoy popularity and applications in the astrophysics simulations [Liu 1997]. [Grama 1994] presented two parallel formulations of the BH algorithm. The first uses static partitioning of the domain and assignment of subdomains to processors. And the second combines static decomposition of the domain with an assignment of

subdomains to processors based on Morton ordering. [Lu 1996] design a parallel fast multipole algorithm in three dimensions.

2. PERFORMANCE OPTIMIZATION TECHNIQUES OF THE DISTRIBUTED BH ALGORITHM

There is a lot of parallelism in the tree code algorithms including computation of the BH tree in parallel, computation of force on each leaf node independently, and data parallelism. Our goal is to improve the performance of the distributed versions of the BH code which run on a cluster of workstations and use MPI for communication. We design four performance optimization techniques. These techniques improve the performance metrics such as speedup, system efficiency, run time and communication overhead. The first technique partitions the data and balances the load among the processors so that the algorithm becomes fully distributed with no initialization overhead. The second method uses pipelining and processor domain partitioning to enhance the overlapping between force computation and processor communication. It also makes communication asynchronous to minimize processor waiting time. The third method changes the processor communication model from peer-to-peer to master-slave. This change minimizes the total communication overhead. The fourth method uses one tree traversal for computing the force on all particles in the processor domain instead of traversing the tree once for each particle. This method reduces the run time very significantly. For performance comparison purpose, the distributed version of the BH code which we intend to optimize is the code written by Chris Gottbrath from the astronomy department at the University of Arizona as part of a research project and is summarized in **Figure 5**.

As will be seen in the following subsections, we take the non-optimized BH algorithm and make some modification in the communication model and in partitioning and load balancing to improve performance by utilizing the distributed computing capability of the cluster of workstations. The basic approach of the non-optimized algorithm is kept the same in regards to how BH tree is constructed, forces are calculated etc. Hence, the optimized algorithms will keep the same level of results accuracy and errors as the non-optimized algorithm.

In the non-optimized distributed BH algorithm of **Figure 5**, MPI_Initialize initializes the MPI environment. Then the processor with rank = 0 calls the ReadParticleInfo procedure to read the particles data from an inputfile in

a similar way to that used in the sequential BH algorithm and stores it in an array of structures all_particles. Then it broadcasts all_particles to all other processors using MPI_Bcast. Then, all the processors proceed in parallel to build the BH tree, calculate the center of mass of all nodes in the tree, calculate the force on the particles in their domain, and update the particle information in the same fashion used in the sequential BH algorithm. Each processor is responsible for calculating the force on the particles in its domain. Finally, all the processors send the updated data of their domain to all other processors using MPI_Allgather. MPI_Allgather is a synchronous (blocking) collective operation.

```
1. MPI_Initialize
2. If the rank of the processor is 0
2.1 all_particles = ReadParticleInfo (in_file)
2.2 MPI_Bcast ( .. , all_particles, .. )
3. Parallel do: For every processor PROC[i] in the set of workstations
3.1 my_particles = all_particles[j],          i(N/P) ≤  j < (i+1)(N/P)
3.2 Loop for a given number of time steps
3.2.1 T = BuildBHTree (all_particles)
3.2.2 ComputeCenterOfMass(T)
3.2.3 For every particle pt in the domain my_particles do
3.2.3.1 CalcForces(T, pt)
3.2.3.2 UpdateParticleInfo(pt)
3.2.4 MPI_Allgather(.. ,my_particles, .. )
3.3 End loop
4. End parallel do
```

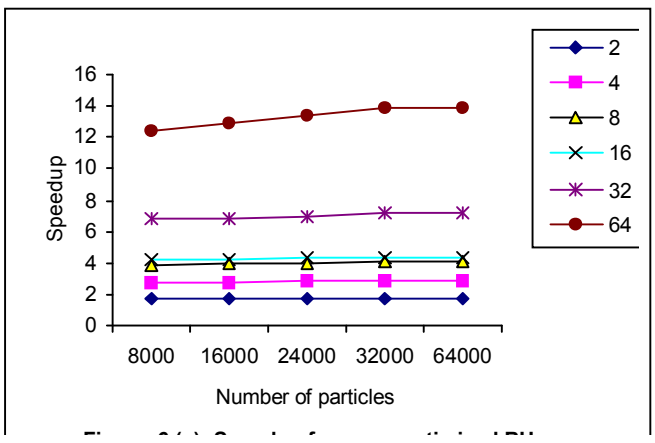
Figure 5: Non-optimized distributed version of the BH code

Since the BH tree shown in Figure 4 is valid only as an initial condition and the particles migrate from one cube to another during simulation, the non-optimized distributed BH algorithm as well as the four proposed optimized distributed BH algorithms solve this particle migration problem by rebuilding the BH tree at the beginning of each time step. In brief, we use the group partition approach in which particles are divided into equal groups. Equal groups of particles are assigned to processors. Each processor follows its assigned group particles through space. The number of particles assigned to a processor remains the same throughout the computation. Therefore, we can spread the workload evenly across the processors to achieve the maximum utilization of processing. The assignment of particles to processors and load balancing is treated and explained in more detail in Section 3.1 (Data partitioning and load balancing).

Table 2 shows the performance results (speedup and efficiency) of the non-optimized distributed algorithm. **Figure 6** plots the results of **Table 2** graphically. **Figure 6 (a)** plots the speedup and **Figure 6 (b)** plots the efficiency. The efficiency of the cluster becomes smaller as the number of processors increase. This is due to the increase of the communication overhead. The speedup increases slightly by increasing the number of particles for a fixed number of processors. This implies that the algorithm is input data size scalable. Also, the speedup grows when the number of processors is increased. For **Table 2**, the sequential times are 37, 87, 141, 202, and 456 seconds for 8,000, 16,000, 24,000, 32,000 and 64,000 particles respectively. The performance results of the non-optimized distributed BH algorithm will be used to measure the amount of performance increase due to using the four performance optimization techniques designed in this paper.

Table 2: Speedup (Sp) and efficiency (E) of the non-optimized distributed BH algorithm for different number of particles (N) and different number of cluster workstations (P)

P	N=8,000		N=16,000		N=24,000		N=32,000		N=64,000	
	Sp	E (%)	Sp	E (%)	Sp	E (%)	Sp	E (%)	Sp	E (%)
2	1.68	84	1.70	85	1.74	87	1.75	87.5	1.77	88.5
4	2.72	68	2.76	69	2.79	70	2.85	71	2.89	72
8	3.85	48	3.90	48.8	3.94	49	3.99	50	4.05	50.6
16	4.12	25.8	4.20	26	4.23	26	4.29	27	4.36	27
32	6.72	21	6.8	21	6.88	21.5	6.98	21.8	7.15	22
64	11.92	18.6	12.40	19	12.9	20	13.45	21	13.85	21.6



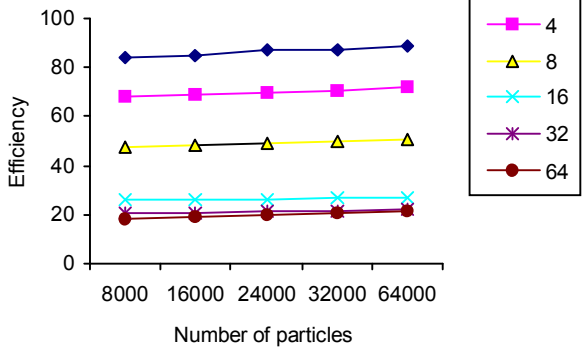


Figure 6 (b): Efficiency of the non - optimized BH algorithm

In the following subsections we describe the four performance optimization techniques in detail. In section 4 we measure the performance optimization achieved by these techniques.

3.1 DATA PARTITIONING AND LOAD BALANCING

For load balancing purposes both data and computational work must be partitioned and distributed evenly among the workstations. Data balancing is achieved through domain decomposition. Load balancing exploits the available parallelism in the BH algorithm. In the current distributed versions of the BH algorithm the computation is load-balanced using the method of orthogonal recursive bisection (ORB). It takes into consideration the fact that if a cluster of particles can be approximated for a given particle then it is more likely that it will be approximated to the nearby particles as well. The disadvantages of ORB are: First, the synchronization penalty since the processors have to wait until the whole locally essential tree is available. Second, it is expensive to re-compute the ORB at each time step. Third, the demand of large memory due to the storage of the ORB tree in each processor. The gain by the decomposition is lost by the storage of the ORB tree in every processor. At present, the code is still marginally time limited and available memory will be greater in the next generation of machines [Dubinski 1996]. Finally, the dynamic reassignment of particles to processors each time step includes a lot of computational overhead and hinders the benefits of the use of the ORB algorithm.

In a cluster of workstations there is a relatively small amount of memory available on each processor [Dubinski 1996]. If a copy of the complete BH tree is stored in each processor's memory then there will be prohibitively large amount of memory required at each processor, but the processors can proceed in parallel to evaluate the forces in each time step. Without data partitioning, the BH algorithm requires that all particle data be stored in all processors main memories. But the entire BH tree is too large and too expensive to be stored in each processor. In fact, the size of main memory restricts the number of particles that can be simulated, this will remain true until the available memory will be greater in the next generation of machines. This makes data partitioning an important issue. Our data partitioning approach decreases memory size required to store local domain of particles.

Our partitioning approach implies static data partitioning and load balancing because the size of the data and load are known prior to running the program.

DISTRIBUTED TREE CODE ON CLUSTER

Static data partitioning and load balancing eliminate the extra scheduling and data partitioning overhead during program execution. The approach provides a simple way to decompose space among processors, and a way to quickly map particles in space to processors. Each processor domain will contain ⌈ N/P ⌉ particles. This divides the data and load equally between processors and minimizes communication overhead. Using our approach ensures data and load balancing and minimizes total communication. The workstations calculate forces for particles in their domain in parallel and require only 2⌈ N/P ⌉ * (particle structure size) bytes of memory instead of the N * (particle structure size) bytes of memory required by the non-optimized distributed BH algorithm. The data partitioning and load balancing algorithm is summarized in **Figure 7**.
In the algorithm of **Figure 7**, MPI_Initialize initializes the MPI environment. Then all the processors read their domain's particles data from an input file in parallel using the ReadParticleDomain procedure and store it in my_particles list. Then the processor's domain is copied into PartialBH_Particles list. All processors proceed simultaneously to send their domain to all other processors and receive the other processors domains using non-blocking MPI send and receive operations respectively. Non-blocking send and receive replace the blocking MPI_Allgather operation used for processor communication in the non-optimized distributed BH algorithm. Using non-blocking communication operations enables overlapping between computation and communication and minimizes task synchronization time and processor waiting time. While processors exchange their domains they proceed to build partial BH tree, compute the center of mass and calculate forces on their domain particles. In CalcForces procedure in the above algorithm, each processor calculates the force on the processor's domain by doing a depth-first tree traversal, truncating at internal nodes where approximation is applicable. Using this approach enables the processors to proceed with force calculation before the whole BH tree is available. After every domain exchange between two processors they proceed to build the partial BH tree and force calculation for the forces exerted by the partial tree built from the received processor's domain. This process continues until each processor exchanges data with all other processors. MPI_Barrier synchronizes tasks and ensures that another force calculation does not proceed until an exchange of domain with another processor is completed.

```
1. MPI_Initialize
2. Parallel do: for every processor PROC[i] in the set of workstations
2.1 my_particles = ReadParticleDomain(i , in_file)
2.2 PartialBH_Particles = my_particles
2.3 Loop for the required number of iterations
2.3.1 for every processor PROC[j] in the set of workstations, j ≠ i
2.3.1.1 non_blocking_MPI_send(j, my_particles)
2.3.1.2 tmp_particle_list = non_blocking_MPI_receive(j)
2.3.2  T= BuildLocalBHTree(PartialBH_Particles)
2.3.3  ComputeCenterOfMass(T)
2.3.4   For every particle pt in the domain my_particles do
2.3.4.1 CalcForces(T, pt )
2.3.5 MPI_Barrier
2.3.6 if  PROC[i] exchanged domain with all other processors
                PartialBH_Particles = my_particles
2.3.7 else                      PartialBH_Particles = tmp_particle_list
2.3.8 UpdateParticleInfo(pt)
2.4 End loop
3. End parallel do
```

Figure 7: The data partitioning and load balancing algorithm

Some of the advantages of our data partitioning and load balancing approach are: First, it enable users to run N-body tree codes for larger number of particles on the same cluster of workstations without additional resources. Second, there is no sequential initialization phase in the algorithm. A sequential fraction of a program limits the maximum speedup of a parallel program according to Amdahl's law. Amdahl observed that for a fixed problem size, if s is the fraction of uni-processor execution time that can not be parallelized, then the maximum obtainable speedup is 1/s, even if the rest of the program is executed in infinitesimal time by an infinite number of processors [Singh 1993]. Processors proceed from the beginning to the end in a complete distributed fashion starting from reading input data to calculating forces. Third, replacing the blocking MPI_Allgather by non-blocking MPI send and receive enables overlapping between communication and computation. Finally, the force calculation proceeds before the whole BH tree is available.

3.2 PIPELINING AND DOMAIN PARTITIONING

To enhance the overlapping between processor communication and force calculation and to minimize processor waiting time and task synchronization

time we present in this section a processor domain partitioning performance optimization technique. In this techniques the processor's domain is partitioned into M partitions and each processor exchanges one partition at a time with all other processors using non-blocking MPI send and receive operations. Processors proceed in calculating force on their domain particles for the received ith partition while they wait for the (i+1)th partition until all the partitions are processed. This approach uses pipelining principle by utilizing the communication network and CPU time simultaneously for different data. This technique is especially powerful when a fast data communication network is used in the cluster of workstations. The pipelined distributed BH algorithm is summarized in **Figure 8**.

The algorithm in **Figure 8** is similar to the data partitioning and load balancing algorithm except that the processors communication is at a finer granularity i.e., the domain partition level. In addition, the BH sub-tree built at each step in each processor consists of only $\lceil N/P \rceil / M$ particles and not $\lceil N/P \rceil$ particles as in the case of the non-pipelined algorithm presented in section 3.

3.3 MASTER-SLAVE COMMUNICATION MODEL

The non-optimized distributed BH algorithm uses MPI_Allgather for processor communication and exchange of data. The outcome of a call to MPI_Allgather(...) is as if all processes executed P calls to MPI_Gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm), for root = 0 , ..., P-1. The outcome of a call to MPI_Gather(…) is as if each of the P processes in the group (including the root process) had executed a call to MPI_Send(sendbuf, sendcount, sendtype, root, ...), and the root had executed P-1 calls to MPI_Recv(recvbuf, recvcount, recvtype, , root, ...)[Snir 1996]. So the total number of MPI_Send( )is $P*P= P^2$ in each time step. Each MPI_Send sends $\lceil N/P \rceil$ particles. This results in the total communication traffic complexity of $O(P^2 * N/P) = O(P*N)$. We can optimize this communication traffic complexity to O(N) by changing the communication model from peer-to-peer (each processor sending to each other processor) to master-slave (each processor sending to one master processor and the master processor sends to the slave processors).

```
1. MPI_Initialize
2. Parallel do: for every processor PROC[i] in the set of workstations
2.1 my_particles = ReadParticleDomain(i , in_file)
2.2 PartialBH_Particles = my_particles[m]                    0 ≤ m < ⌈ N/P ⌉/M
2.3 Loop for the required number of iterations
2.3.1 set k = 0
2.3.2 for every processor PROC[j] in the set of workstations, j ≠ i
2.3.2.1 non_blocking_MPI_send(j, my_particles[m])   k⌈ N/P ⌉/M ≤ m < (k+1)⌈ N/P ⌉/M
2.3.2.2 tmp_particle_list = non_blocking_MPI_receive(j)
2.3.3  T= BuildLocalBHTree(PartialBH_Particles)
2.3.4  ComputeCenterOfMass(T)
2.3.5   For every particle pt in the domain  my_particles do
2.3.5.1 CalcForces( T, pt )
2.3.6 MPI_Barrier
2.3.7 increment k
2.3.8 if k = M go to 2.3.11
2.3.9 if  PROC[i] exchanged partition k with all other processors
                    PartialBH_Particles = my_particles[m]        k⌈ N/P ⌉/M ≤ m < (k+1)⌈ N/P ⌉/M
2.3.10 else         PartialBH_Particles = tmp_particle_list
2.3.11 UpdateParticleInfo(pt)
2.4 End loop
3. End parallel do
```

Figure 8: The pipelined distributed BH algorithm

In the master-slave communication model, MPI_Allgather is replaced by the following send of MPI operations:

1. Each slave processor issues one asynchronous MPI send operation to send its domain (⌈ N/P ⌉ particles) to the master processor.
2. The master processor issues P-1 asynchronous MPI send operations to send ⌈ N/P ⌉ particles to the slave processors in each operation.

The total communication complexity of the master-slave communication model = O(2(P-1)* ⌈ N/P ⌉) = O (N). The master-slave communication model distributed BH algorithms is summarized in **Figure 9**.

Most of the algorithm in **Figure 9** is similar to the algorithms presented in sections 3.1 and 3.2. The difference between this algorithm and the algorithm presented in section 3.1 is the way processors communicate. In the algorithm above, if the processor's root ≠ 0 then it is a slave processor, and it is responsible for calculating the forces on particles in its domain. It sends its updated domain to the master at each time step using the non_blocking_MPI_send operation. If the root = 0 then the processor is used as the master processor and as a communication server since its only responsibility

is to communicate particle data between the slave processors. It receives the other processors domains and broadcasts them to all other processors using the non_blocking_MPI_Bcast operation.

```
1. MPI_Initialize
2. Parallel do: for every processor PROC[i] in the set of workstations
2.1 if root ≠ 0
2.1.1 my_particles = ReadParticleDomain(i , in_file)
2.1.2 PartialBH_Particles = my_particles
2.2 Loop for the required number of iterations
2.2.1 if root ≠ 0
2.2.1.1 non_blocking_MPI_send(0, my_particles)
2.2.1.2 tmp_particle_list = non_blocking_MPI_receive(0)
2.2.2 if root = 0
2.2.2.1 my_particles = non_blocking_MPI_receive(i)    i=1,2,…P-1
2.2.2.2 non_blocking_MPI_Bcast(0, my_particles)
2.2.3 if root ≠ 0
2.2.3.1  T= BuildLocalBHTree(PartialBH_Particles)
2.2.3.2  ComputeCenterOfMass(T)
2.2.3.3   For every particle pt in the domain my_particles do
2.2.3.3.1 CalcForces(T, pt )
2.2.3.4 MPI_Barrier
2.2.3.5 if  this is the first iteration and the processor received its own domain from the master
processor Do nothing
2.2.3.6 else          PartialBH_Particles = tmp_particle_list
2.2.3.7 UpdateParticleInfo(pt)
2.3 End loop
3. End parallel do
```

Figure 9: The master-slave communication model distributed BH algorithms

3.4 Tree Traversal once for each processor domain

From the run time cost analysis of the BH algorithm shown in **Table 1**, the CalcForces procedure takes more than 96% of the total run time of the algorithm. So, one of the ways to optimize the algorithm is to optimize the CalcForces procedure. CalcForces procedure traverses the BH tree once to calculate the force on each particle in the processor's domain. So the tree is traversed a number of times equals to the total number of particles in order to calculate the force on all particles. The partial code of the non-optimized distributed BH algorithm responsible for force calculation is shown in **Figure10**.

```
For every particle pt in the processor's domain my_particles do
        CalcForces(T, pt )
Procedure CalcForces(T,pt)   Begin
        …….      TraverseTree(T,pt);          …….
End;
```

Figure 10: The partial code of the non-optimized distributed BH algorithm responsible for force calculation

We optimize CalcForces procedure by traversing the tree only once to calculate the forces exerted on all particles in the processor's domain by the particles in the BH tree. The tree-traversal once optimized distributed algorithm is summarized in **Figure 11**.

AS IS THE CASE OF ALL THE FOUR OPTIMIZED ALGORITHMS, THE BH TREE IS NOT

```
1. MPI_Initialize
2. Parallel do: for every processor PROC[i] in the set of workstations
2.1 my_particles = ReadParticleDomain(i , in_file)
2.2 PartialBH_Particles = my_particles
2.3 Loop for the required number of iterations
2.3.1 for every processor PROC[j] in the set of workstations, j ≠ i
2.3.1.1 non_blocking_MPI_send(j, my_particles)
2.3.1.2 tmp_particle_list = non_blocking_MPI_receive(j)
2.3.2  T= BuildLocalBHTree(PartialBH_Particles)
2.3.3  ComputeCenterOfMass(T)
2.3.4   TraverseTreeOnce (T, my_particles)
2.3.5 MPI_Barrier
2.3.6 if  PROC[i] exchanged domain with all other processors
                    PartialBH_Particles = my_particles
2.3.7 else          PartialBH_Particles = tmp_particle_list
2.3.8 UpdateParticleInfo(pt)
2.4 End loop
3. End parallel do
```

Figure 11: The tree-traversal once optimized distributed algorithm

instantly available within each processor to be traversed once. What is available is the sub-tree corresponding to one processor's domain at a time. But this is

one of the advantages of our approach: the overlapping between computation of forces and communication between processors. Each processor traverses this sub-tree once to calculate the forces on all particles in its local domain. This tree-traversal once is done in parallel by all processors in a distributed environment fashion.
TraverseTreeOnce procedure traverses the BH tree once to calculate the forces on all particles in the processor's domain exerted by particles in the BH tree by doing depth-first tree traversal truncating at internal nodes where approximation is applicable.

3. EXPERIMENTAL RESULTS

In this section we present some simulation results. A large number of runs were performed to estimate the efficiency of the distributed version of the BH algorithm and the proposed optimization techniques. These runs were performed on Roadrunner Linux super-cluster of dual SMP workstations at University of New Mexico. The super cluster consists of 64 workstations and is connected by Myrinet (data) and fast Ethernet (control). Each workstation has the following specification: The CPU is Intel Pentium II 450 MHz, The cache is 512 KB ECC RAM, The RAM is 512 MB synchronous DRAM, The hard disk is 6.4 GB UltraDMA EIDE. The programs are written in C++ using MPI version 1.2. The input particle data is generated randomly in three dimension space. All the experiments are conducted for 10 time steps. The reported results in tables 4,5,6, and 7 include the performance gain due to using the data partitioning and load balancing, as well as using asynchronous communication operations that overlap communication and computation.
Experimental results were based on a 10-time steps simulation. Such number of steps is not practical of real problem solving. But here we only use simulation for performance comparison only between the non-optimized and optimized distributed BH algorithm for the different optimization techniques. Number of steps here is then irrelevant. Using a large number of time-steps will result is prohibitive overall program execution time.
In **Table 3** we measure the performance (speedup and efficiency) of the algorithm that uses the data partitioning and load balancing performance optimization technique. **Figure 7** plots the results of **Table 3** graphically. **Figure 7 (a)** plots the speedup and **Figure 7 (b)** plots the efficiency. **Table 3** shows the speedup and efficiency for different number of workstations and

different number of particles. Data partitioning and load balancing technique improves the efficiency of the non-optimized distributed BH algorithm by up to 10% (see **Table 2** for the performance of the non-optimized distributed BH algorithm). As **Table 3** shows, the speedup grows when the number of processors is increased. Also, the algorithm is scalable for large N. From the results, also the efficiency decreases as the number of workstations increases. For example the efficiency is up to 91.5% for 2 processors, 82% for 4 processors, 60% for 8 processors, 33.7% for 16 processors, 27.3% for 32 processors, and 22.4% for 64 processors. This is attributed mainly to the added communication cost for large number of processors. Larger numbers of workstations are probably not practical since communication costs will begin to dominate unless the network bandwidth increases. Evidently, speedup flattens out as a function of the number of processors at some point for a fixed number of particles. In fact there is a point beyond which, using more processors to solve a fixed-size problem actually increases the execution time.

Table 3: Speedup (Sp) and efficiency (E) for different number of workstations, P and different number of particles (N) for the algorithm that uses data partitioning and load balancing performance optimization technique

P	N=8,000		N=16,000		N=24,000		N=32,000		N=64,000	
	Sp	E (%)	Sp	E (%)	Sp	E (%)	Sp	E (%)	Sp	E (%)
2	1.75	87.5	1.80	90	1.82	91	1.83	91.5	1.83	91.5
4	3.20	80	3.21	80	3.24	81	3.25	81	3.28	82
8	4.70	58	4.74	59	4.79	59.9	4.80	60	4.82	60
16	5.20	32	5.23	33	5.25	32.8	5.30	32	5.39	33.7
32	8.05	25	8.22	25.7	8.39	26.2	8.65	27	8.75	27.3
64	13.82	21.5	14.03	21.9	14.14	22.1	14.20	22.2	14.35	22.4

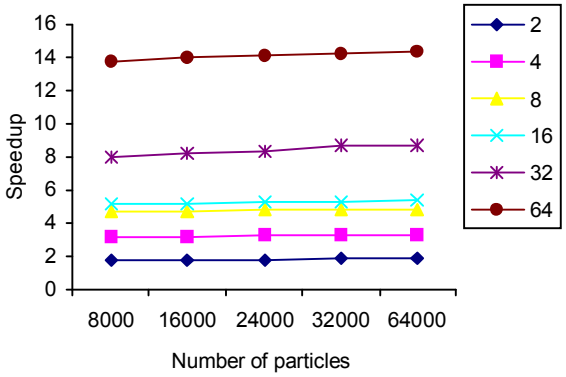


Figure 7 (a): Speedup for data partitioning and load balancing algorithm

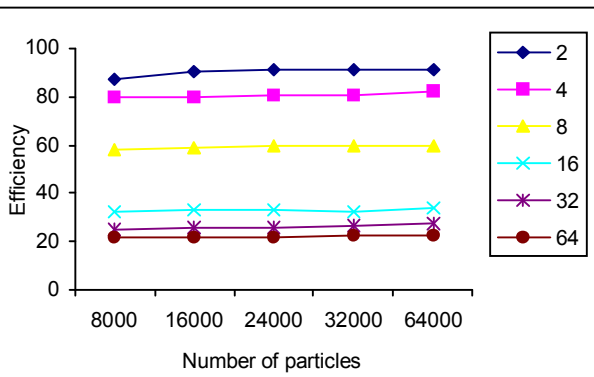


Figure 7 (b): Efficiency for data partitioning and load balancing algorithm

In **Table 4** we measure the performance (speedup and efficiency) of the algorithm that uses the pipelining performance optimization technique. **Figure 8** plots the results of **Table 4** graphically. **Figure 8 (a)** plots the speedup and **Figure 8 (b)** plots the efficiency. **Table 4** shows the speedup and efficiency for different number of workstations and 16,000 particles. The pipelining technique improves the efficiency of the non-optimized distributed BH algorithm by up to 31.1% (see **Table 2** for the performance of the non-optimized distributed BH

algorithm). As **Table 4** shows, the speedup grows when the number of processors is increased. Also the algorithm is scalable for large N. In addition, the results show that the efficiency decreases as the number of workstations increases. The speedup grows when the number of domain partitions is increased up to a certain value, then it starts decreasing. This is due to the increase in the time it takes to traverse the smaller BH sub-trees for larger number of partitions, and to the increased communication overhead due to finer granularity of the communicated data. The optimal value of number of partitions depends on both the speed of the CPU of the workstations and the speed of the communication network used in the cluster of workstations.

In **Table 5** we measure the performance (speedup and efficiency) of the algorithm that uses the master-slave performance optimization technique. **Figure 9** plots the results of **Table 5** graphically. **Figure 9 (a)** plots the speedup and **Figure 9 (b)** plots the efficiency. **Table 5** shows the speedup and efficiency for different number of workstations and different number of particles. Master-slave technique improves the efficiency of the non-optimized distributed BH algorithm by up to 41.7% (see **Table 2** for the performance of the non-optimized distributed BH algorithm). As **Table 5** shows, the speedup grows when the number of processors is increased. Also, the algorithm is scalable for large N. In addition, the efficiency decreases as the number of workstations increases. As the results also show, the speedup and efficiency are small compared to the other optimization techniques. This is due to the use of one workstation as a communication server. The communication server does not contribute to the force computation.

Table 4: Speedup (Sp) and efficiency (E) for different number of domain partitions (M) for 16,000 particles and different number of workstations

P	M=2		M=4		M=6		M=8		M=10		M=12		M=14		M=16	
	Sp	E%	Sp	E%	Sp	E%	Sp	E%	Sp	E%	Sp	E%	Sp	E%	Sp	E%
2	1.82	91.0	1.85	92.5	1.87	93.5	1.88	94.0	1.90	95	1.87	93.5	1.84	92.0	1.83	91.5
4	3.28	82.0	3.34	83.5	3.39	84.8	3.50	87.5	3.55	88.8	3.43	85.8	3.39	84.8	3.32	83.0
8	4.90	61.1	5.12	64.0	5.29	66.1	5.52	69.0	5.53	69.1	5.40	67.5	5.32	66.5	5.18	64.8
16	5.82	36.4	6.40	40.0	7.81	48.8	9.11	56.9	8.62	53.8	8.13	50.8	7.55	47.2	7.04	44.0
32	8.90	27.8	9.67	30.2	10.22	31.9	11.15	34.8	10.95	34.2	10.42	32.6	10.04	31.4	9.65	30.2
64	15.6	24.4	17.20	26.9	18.46	28.8	20.17	31.5	19.22	30.0	18.6	29.1	18.00	28.1	17.55	27.4

(P) for the algorithm that uses pipelining performance optimization technique

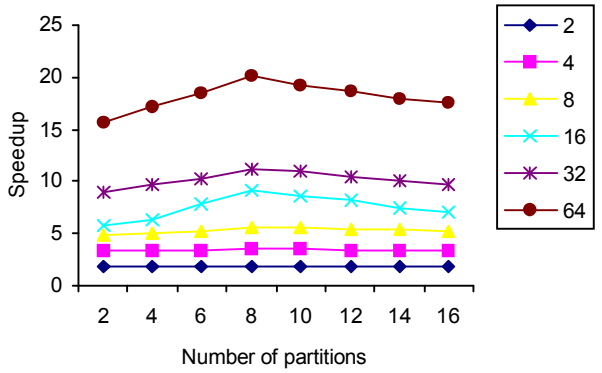


Figure 8 (a): Speedup for the pipelining algorithm

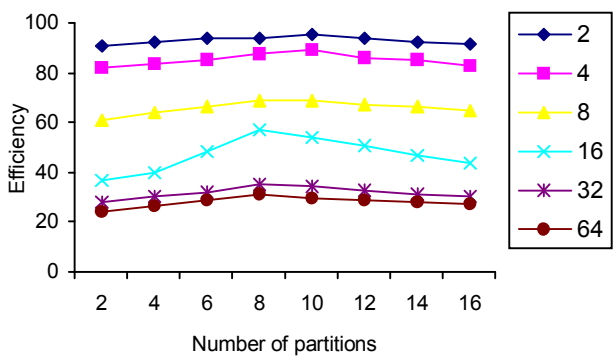


Figure 8 (b): Efficiency for the pipelining algorithm

Table 5: Speedup (Sp) and efficiency (E) for different number of workstations (P) and different number of particles (N) for the algorithm that uses the master-slave performance optimization technique

	N=8,000		N=16,000		N=24,000		N=32,000		N=64,000	
	Sp	E %	Sp	E %	Sp	E %	Sp	E %	Sp	E %
2	0.80	40	0.83	41.5	0.85	42.5	0.89	44.5	0.9	45 %
4	2.31	57.8	2.40	60	2.49	62.3	2.51	62.8	2.57	64.3
8	5.13	64.1	5.60	70.0	5.90	73.8	6.23	77.9	6.50	81.3
16	6.15	38.4	7.24	45.3	8.40	52.5	9.54	59.6	10.80	67.5
32	9.51	29.7	11.22	35.1	13.52	42.3	15.92	49.8	18.25	57.0
64	15.40	24.1	17.34	27.1	19.50	30.5	22.13	34.6	24.37	38.1

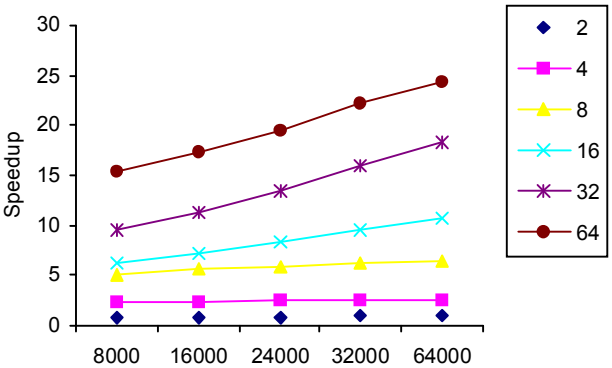


Figure 9 (a): Speedup for the master -slave algorithm

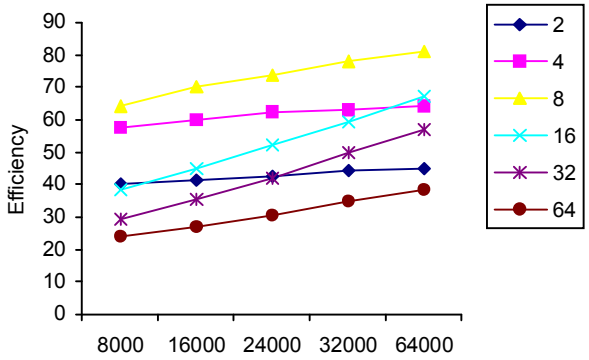


Figure 9 (b): Efficiency of the master -slave algorithm

In **Table 6** we compare the communication overhead of the algorithm that uses master-slave performance optimization technique with that of the non-optimized distributed BH algorithm. **Figure 10** plots the results of **Table 6** graphically for N=64000. **Table 6** shows the communication overhead for different number of workstations and different number of particles. Communication overhead of the algorithm that uses the master-slave technique contributes up to 21% and the non-optimized distributed BH algorithm contributes up to 43% of the total run time. Master-slave technique optimizes the communication overhead of the non-optimized distributed BH algorithm by up to 22%. As **Table 6** shows, for a fixed number of particles, the communication overhead grows when the number of processors is increased. For example, for 64,000 particles, the communication overhead of the algorithm that uses master-slave performance optimization technique is up to 8% for 2 processors, 11% for 4 processors, 15% for 8 processors, 18% for 16 processors, 19% for 32 processors, and 21% for 64 processors. This can be attributed mainly to the increased network traffic for large number of processors. Larger numbers of workstations are probably not practical since communication costs will begin to dominate unless the network bandwidth increases. The results show also that the communication overhead increases as the number of particles increase. This means that we can expect improved performance by using clusters of workstations with faster networks.

Table 6: Comparison between the communication overhead for different number of workstations (P) and different number of particles (N) for the algorithm that uses the master-slave performance optimization technique (Master/salve) and the non-optimized distributed BH algorithm (Non-opt.)

P	Communication overhead									
	N=8,000		N=16,000		N=24,000		N=32,000		N=64,000	
	Master/ slave	Non-opt.	Master/ slave	Non-opt.	Master/ slave	Non-opt.	Master/ slave	Non-opt.	Master/ slave	Non-opt.
2	0.005	0.01	0.01	0.03	0.03	0.04	0.05	0.08	0.08	0.15
4	0.01	0.03	0.03	0.06	0.06	0.11	0.08	0.15	0.11	0.23
8	0.03	0.05	0.05	0.11	0.08	0.17	0.11	0.24	0.15	0.30
16	0.06	0.10	0.08	0.15	0.11	0.25	0.14	0.29	0.18	0.34
32	0.09	0.16	0.11	0.21	0.13	0.28	0.15	0.33	0.19	0.39
64	0.13	0.25	0.15	0.31	0.18	0.38	0.19	0.41	0.21	0.43

In **Table 7** we measure the performance (speedup and efficiency) of the algorithm that uses the tree-traversal once performance optimization technique. **Figure 11** plots the results of **Table 7** graphically. **Figure 11 (a)** plots the speedup and **Figure 11 (b)** plots the efficiency. **Table 7** shows the speedup and efficiency for different number of workstations and different number of particles. Tree-traversal once technique improves the efficiency of the non-optimized distributed BH algorithm by up to 45.2% (see **Table 2** for the performance of the non-optimized distributed BH algorithm). As the results show, the speedup grows when the number of processors is increased. Also the algorithm is scalable for large N. The results show also that the efficiency decreases as the number of workstations increases.

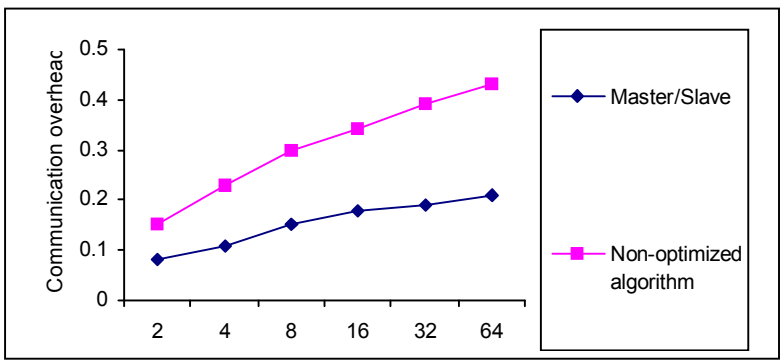


Table 7: Speedup (Sp) and efficiency (E) for different number of workstations (P) and different number of particles (N) for the algorithm that uses tree-traversal once performance optimization technique

P	N=8,000		N=16,000		N=24,000		N=32,000		N=64,000	
	Sp	E (%)	Sp	E (%)	Sp	E (%)	Sp	E (%)	Sp	E (%)
2	1.88	94	1.89	94.5	1.9	95	1.92	96	1.93	96.5
4	3.51	87.8	3.56	89 %	3.62	90.5	3.67	91.8	3.70	92.5
8	5.85	73.1	5.94	74.3	6.25	78.1	6.68	83.5	6.93	86.6
16	8.03	50.2	8.58	53.6	9.26	57.9	10.46	65.4	11.12	69.5
32	11.85	37.0	13.73	42.9	16.43	51.3	20.40	63.8	21.2	66.2
64	21.60	33.8	25.15	39.3	30.40	47.5	35.08	54.8	40.03	62.5

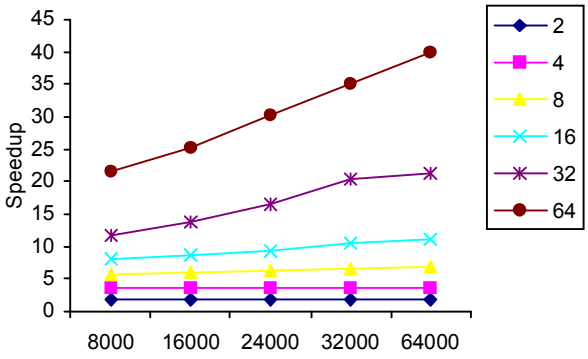


Figure 11 (a): Speedup for three -traversal once algorithm

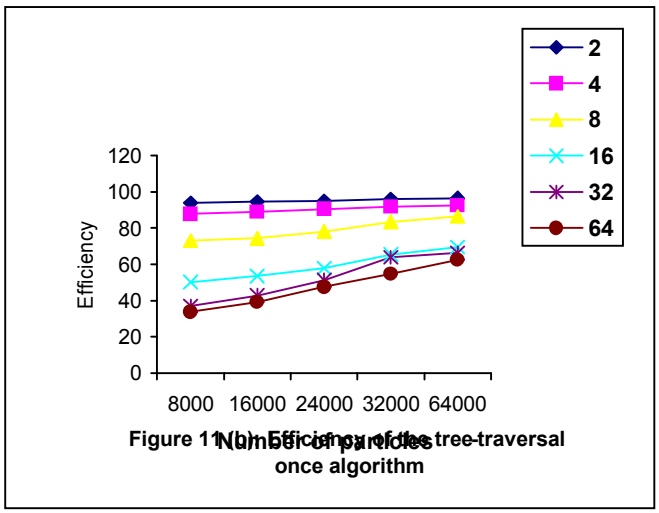


Figure 11 (b) Efficiency of the tree-traversal once algorithm

4. CONCLUSION

In this paper we presented four performance optimization techniques to enhance the performance of the distributed BH algorithms running on clusters of workstations and using message passing interface (MPI). We experimentally demonstrated the performance of these techniques. The experiments were run on Roadrunner Linux super-cluster of dual SMP workstations at University of New Mexico. The super-cluster consists of 64 workstations and is connected by Myrinet(data) and fast Ethernet (control). As expected, using our optimization techniques greatly enhances the performance of the distributed BH code. From the results, it is possible to obtain high performance optimization using the presented techniques out of a cluster of workstations given the workstations are not overloaded, the communication and computation are overlapped, and the workstations are connected using a fast data network.

ACKNOWLEDGEMENTS

The authors sincerely acknowledge Chris Gottbrath from the astronomy department at the University of Arizona for providing the source code of the basic MPI code on which our optimization techniques were based. We would like also to sincerely acknowledge Dr. Salim Hariri from the electrical and computer engineering department at the University of Arizona for the useful discussions we had at a certain time during the early development of this

research. The authors also acknowledge the Albuquerque High Performance Computing Center at the University of New Mexico for providing access to the Roadrunner Linux super-cluster of dual SMP workstations at university of New Mexico. This research was funded by the Fulbright scholarship awarded to the author during the academic year of 1999-2000.

REFERENCES

A. W. Appel. An efficient Program for Many Body Simulation. SIAM Journal on Specific and Statistical Computing, 6, 1985.

Barnes J. and P. Hut 1986, A Hierarchical O(N Log N) force-calculation algorithm. Nature pages 324-446

Joshua Barnes and Piet Hut. Error Analysis of a Tree Code. The Astrophysical Journal Supplement Series, June 1989. Pages 389-417

S. Bhatt; M. Chen; C-Y. Lin; P. Liu. In Proceedings of Scalable High Performance Computing Conference, Los Alimitos

John Dubinski. A Parallel Tree Code. New Astronomy, Jan. 1996. Pages 133-147

H. El-Rewini and T. Lewis. Distributed and Parallel computing. 1998. Manning Publications Company.

Lars Hernquiest. Performance Characteristics of Tree Codes. The Astrophysical Journal Supplement Series, June 1989. August 1987. Pages 715-734

Ananth Grama, Vipin Kumar and Ahmed Sameh. Scalable Parallel Formulation of the Barnes-Hut Method for n-Body Simulations.In Proceedings of Supercomputing 1994, Nov. 14-18. Pages 439-448

L. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations. Journal of Computational Physics, 73, 1987

Jaswinder Pal Singh, John L. Hennessy, and Anoop Gupta. Scaling Parallel Programs for Multiprocessors: Methodology and Examples. Computer, July 1993. Pages 42-50

J. Pal Singh. Ph.D. Thesis, 1993, Stanford University

Pangfeng Liu and Jan-Jan Wu. A Framework for Parallel-Tree-Base Scientific Simulations. In the Proceedings of International Conference on Parallel Processing, 1997, Pages 137-144

Eric Jui-Lin Lu and Daniel I. Okunbor. A Massively Parallel Fast Multipole Algorithm in Three Dimensions. In the Proceedings of the High Performance Distributed computing, 1996, August 6-9, 1996, Pages 40-48

John Salmon. Parallel Hierarchical N-body Methods. 1990. Ph.D. Thesis, California Institute of Technology, 1990

S. Sundaram. Fast Algorithms for N-body Simulations. Ph.D. Thesis, Cornell University, 1993

Fang Wang and Young-il Choo. Vectorizing the N-body Problem on the CM-5. First International conference on Algorithms and Architectures for Parallel Processing, Vol. 2, 1995. Pages 863-866

Michael Warren and John Salmon. Astrophysical N-body Simulations Using Hierarchical Tree Data Structures. In Proceedings of Supercomputing, 1992. Pages 570-576

Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. MPI: The Complete Reference. 1996 Massachusetts Institute of Technology [Rewini 1998] Hesham El-Rewini and Ted Lewis, Distributed and Parallel Computing, Manning Publication Co., 1998