A Distributed Object-Oriented Method for Particle Simulations on Clusters*

Yudong Sun , Zhengyu Liang , and Cho-Li Wang

Department of Computer Science and Information Systems
The University of Hong Kong , Pokfulam Road, Hong Kong
{ydsun, zyliang, clwang}@csis.hku.hk

Abstract. This paper describes a distributed object-oriented method for solving N -body problem of particle simulations. The method allows dynamic construction of a collaborative system based on the computational requirement of an application and the available resources in the cluster. In the system, a group of objects on distributed hosts cooperate to execute the application. The method is implemented in Java and RMI. The platform-independent features of Java enable the method to support efficient distributed computing in heterogeneous environment. The performance test shows that the method can achieve good speedup and portability. The proposed method can be extended to support other scientific computing applications in distributed environment.

1 Introduction

N-body problem studies the evolution of a system of N bodies (particles) under the cumulative influences on every body from all other bodies that cause the sustained body movement. It is an irregularly structured problem that exists broadly in astrophysics, plasma physics, molecular dynamics, fluid dynamics, radiosity calculations in computer graphics, and etc. [1]. The common feature of these applications is the large range of scales in the information requirements. A body in a physical domain requires progressively less information in less frequency from parts of the domain that are farther away. The domain parts and the influences from those parts are continuously changing during the evolution of the system.

Many algorithms have been proposed to solve N -body problem [2,3,4,5,6]. The Barnes-Hut method [2], for example, is a typical tree-based algorithm which computes the interaction between particles in space by constructing and traversing a

* This research was supported by Hong Kong Research Grants Council (RGC) grant 10201696 and The University of Hong Kong CRCG grant 10200544.

tree which represents the interrelationship between the particles. Fig. 1 gives an example of body distribution in a two-dimensional domain and its Barnes-Hut tree representation. This is a quadtree for the bodies in 2D domain. The Barnes-Hut tree for three-dimensional domain is an octree.

Fig. 1. Barnes-Hut tree in 2D domain: (a) Body distribution and domain partitioning; (b) Barnes-Hut quadtree.

Many parallel N -body algorithms are derived from Barnes-Hut method. Singh and et al. [7] presented a method in shared-address-space multiprocessing model. An octree is created in shared memory segment and it is globally accessible by all concurrent processes on multiple processors. Such a programming model is appropriate for shared memory systems like SMP machines. Multiple processes cooperately build a globally accessible octree and then calculate the force exerted on each body from all other bodies by tree traversal. All inter-process communication is via the shared memory. Hence the communication overhead is quite low. When solving N -body problem on distributed memory systems, message passing is a usual way for inter-process communication [8]. MPI and PVM are common message passing libraries.

This paper proposes a distributed object-oriented method (DOO) for solving N -body problems on a cluster of heterogeneous workstations and PCs. Our final goal is to extend this distributed computing framework to an Internet-based computing environment [9,10]. We choose Java and RMI (Remote Method Invocation) [11] to implement the distributed object-oriented method. With the architecture-neutral and object-oriented features, Java supports applications running in a system which is composed of different platforms. RMI provides an easy way to register remote objects to be the usable computing resources in a distributed system and allocate the computational tasks to the remote objects [11]. The mechanisms can transmit either data or methods between distributed objects and load the program codes into remote objects at runtime. Thus, the algorithm will be adaptive to the system resources and the computing requirements of an application. Java and RMI endue the distributed

object method with high portability and flexibility in heterogeneous and dynamically changed environment [12].

Our method for N -body problem is a distributed object-oriented method rather than simply a message-passing scheme. The new method involves a group of collaborative objects. One of them plays the role of *compute coordinator*. It initiates the computing procedure by invoking remote objects on other hosts, dispatches computational tasks to them, and collects the results at the end. The registration and all interaction between those objects are through RMI interface. The distributed object-oriented method has the following advantages:

- *Platform-independent*: Code in Java and RMI is widely executable on any platform as long as JVM (Java Virtual Machine) is supported. No code modification or recompilation is needed when porting a program onto various platforms.
- *Network-computing*: DOO method is suitable for network computing on distributed platforms. It is an ideal object-oriented framework for Internet-based computation.
- *Object-migrateable*: Distributed objects can be moved from a heavily loaded machine to a less loaded one to make full use of the available computing power.
- *System-adaptive*: Remote objects can be dynamically invoked or terminated to match the system reconfiguration. Thus, the number of objects can be decreased or increased subject to the runtime configuration.
- *Method-passing*: In addition to data, method code can be transmitted from the home server to remote objects. With this capability, the new method can be automatically inserted to remote objects.

In the following sections, we illustrate the distributed object-oriented method in Section 2 and describe the DOO method for solving N -body problem in Section 3. The performance of running the N -body application on clusters will be discussed in Section 4. Finally Section 5 will give the conclusions about this research and briefly discuss the future work.

2. Collaborative System

The distributed object-oriented method is developed based on a collaborative computing model. As shown in Fig. 2, a *collaborative system* consists of a group of distributed objects on different hosts. The object can be referred to as a *compute engine*.

At the beginning, the objects start to run independently on the hosts to participate in the computing over network. The objects *register* themselves to the RMI registry. After the registration, a collaborative system is established and these objects can locate each other by looking up the registry. The distributed objects will accept the computing tasks and cooperatively process an application.

One of the distributed objects, compute coordinator, is responsible for initiating the computation. When distributed computing is demanded, it looks up the registry to

find the available compute engines, and then distributes the computing tasks to these compute engines by invoking the computing method on them. The coordinator coordinates the entire computing procedure in the collaborative system and collects final results. In our N -body application, the compute coordinator dispatches a portion of bodies (a subset of N bodies) to each engine via the RMI *interaction mechanism*. The distributed objects also interact with one another through the interaction mechanism. The interaction includes the invocation of the methods on the compute engines from the coordinator and the data exchange between the engines.

Fig. 2. The collaborative system on a P -node cluster.

The configuration of a networked system can be changed dynamically. Hosts may be turned off for maintenance or be removed from the system. New hosts can be added to the system. The workload on the hosts varies from time to time. Only idle or lightly-loaded hosts are going to accept computational tasks from other hosts. The registration mechanism enables distributed objects to freely join or disjoin a collaborative system, so that the system can be dynamically configured with the available resources. In our implementation, a class, `ComputerImpl()`, is defined to represent the compute engine. To join the collaborative system, each host creates an instance of this class.

When running the N -body application, a collaborative system is established. The procedure of particle simulation proceeds by iterating the computing round (see Section 3.2). It is the opportunity to reconstruct the collaborative system at the beginning of each computing round in correspondence with the current registration of the remote objects. The compute coordinator can redistribute the computing tasks to those objects. After the initiation, the coordinator takes a share of computation and works as a compute engine as well.

3. DOO-based N -body Algorithm

We developed an N -body algorithm based on the Barnes-Hut application in SPLASH2 suite [13]. In our algorithm, a distributed Barnes-Hut tree structure is constructed for facilitating the DOO method.

3.1 Distributed Barnes-Hut Tree

Let N denote the total number of bodies and P the number of hosts. Each host starts a compute engine. If there are four hosts, the domain in Fig. 1(a) will be partitioned into four sub-domains shown in Fig. 3(a). The domain partitioning is based on spatial locality. Thus, the number of bodies in each sub-domain is not necessarily equal. The coordinator divides N bodies into P portions. A portion contains the bodies in one sub-domain. Each compute engine takes one portion of bodies and builds a *local subtree* for the portion. Therefore, the global Barnes-Hut tree in Fig. 1(b) is decomposed into four subtrees shown in Fig. 3(b). The N -body methods in [8] and [14] also adopted distributed tree schemes.

Fig. 3. A distributed Barnes-Hut tree in 2D for four compute engines: (a) Sub-domains after partitioning; (b) Distributed Barnes-Hut tree.

Besides its local subtree, a compute engine needs to access all remote subtrees to compute the force influences from remote bodies. To make the distributed subtrees accessible for all compute engines, each engine also creates a *partial subtree* which consists of the top levels of the local subtree and broadcasts it to all other engines. So every computer engine obtains all remote partial subtrees for its local access.

3.2 Computing Round

After the coordinator has assigned the portions of bodies to remote engines, the particle simulation begins and it is advanced by iterating the computing round, i.e., one step of body movement. A computing round is carried out in three phases.

1. *Subtree construction and propagation*

Compute engines build the subtrees and broadcast the partial subtrees to all other engines. This needs an all-gathering communication. The coordinator gathers all partial subtrees and then propagates them to all compute engines by remote method calls.

2. *Force calculation*

Each engine calculates the forces on the bodies by traversing its local subtree and the received partial subtrees. The bodies that need to visit the lower levels of a subtree not existing in the particle subtree will be sent to the compute engine containing that subtree. Then the forces on these bodies is evaluated by traversing the complete subtree on that compute engine. After that, the bodies return to their original engines. Finally the forces on a body calculated both on its home engine and on remote engines are summed. The new states of a body, including its position and velocity, are determined. In this phase, all-to-all communication is performed among the compute engines.

3. *Body redistribution*

At the end of force calculation, all bodies have moved to their new positions. Some bodies may move to other sub-domains. To keep the spatial locality of the bodies, those bodies should be sent to the destination engines where their new positions locate by an all-to-all communication. In general, only a few bodies will be transmitted in this phase because of the slow pace of body movement.

4. Performance Tests and Analyses

The N -body application in distributed object-oriented method has been tested on homogeneous and heterogeneous clusters. It simulates particles' movement in 3D space. The performance has been inspected in the speedup of the application on clusters and the portability of the DOO method on heterogeneous platforms.

4.1 Speedup

The N -body application in Java and RMI has been executed on a local cluster of Pentium family PCs, including PentiumPro/233MHz, 450MHz, PentiumII/233MHz and K6-2/350MHz linked by 100Mbps Fast Ethernet, running Linux 2.0.3x.

To verify the reduction of communication traffic by the distributed tree structure, we implemented a straight-forward N -body method as a comparison. In this method, the compute coordinator starts a computing round by propagating N bodies wholly to each compute engine, where a *complete tree* containing all bodies is built. Although each engine still computes the forces on a portion of bodies, it has to receive and store all bodies. In contrast to the *distributed tree* method in Section 3, this method can be called *complete tree* method. The force calculation for each body is local to a compute engine by traversing the complete tree. No remote tree access is required. At

the end of the computing round, the compute coordinator collects the new state data of all bodies from the remote compute engines, and then broadcasts the data to all compute engines to start next computing round. There are one broadcast and one gather operation performed in each computing round. So the communication overhead is much higher than the distributed tree method.

Fig. 4. The execution time of the two N -body methods on the homogeneous cluster of Pentium family PCs, with 8K or 10K bodies.

Fig. 4 compares the execution time of the two methods. For the distributed tree method, speedup can be observed on one to eight processors since lower communication is brought about in the broadcast of subtrees and the transmission of bodies. The distributed Barnes-Hut tree is appropriate for distributed computing. On the other hand, speedup merely occurs for smaller machine sizes in the complete tree method. When running on five processors or above, the heavy communication overhead in the broadcasting and gathering of N bodies deteriorates the performance.

4.2 Test in Heterogeneous Cluster

The distributed tree method has also been tested on a wider-range heterogeneous cluster of different platforms. There are totally six processors in the cluster. The processors take part in the computation in the order of two PentiumPro/450MHz PCs running Linux 2.0.36, two Sun UltraSPARC-1 workstations running Solaris 2.6 and one SGI PowerChallenge SMP (two processors are used for this computing) running IRIX 6.2. These hosts locate at different sites. Two PentiumPro's lies in the local cluster; two Sun UltraSPARC-1's belong to another distant cluster; and the SGI PowerChallenge is also a remote server. The hosts are linked together by campus network. Our code can successfully run on these platforms without any modification. Fig. 5 shows the execution time of the distributed tree method on the heterogeneous cluster.

Speedup can be attained in the heterogeneous network. It confirms that the DOO method has platform-independent features and the method supports network-computing well.

Fig. 5. The execution time of the distributed tree method with 10K bodies on the heterogeneous cluster. The processor order is two PentiumPro/450MHz PCs, two Sun UltraSPARC-1 workstations and two processors in SGI PowerChallenge SMP.

The execution time of the Java code is not satisfactory due to the inefficiency of Java interpreter. It is ten times slower than the implementation using C++ and MPI communication library in [15]. But Java owns attractive features of strong platform-independent portability and flexibility. We select Java and RMI to implement the method to take advantage of those features. The performance of the DOO method is expected to be enhanced with the improvement of Java packages.

5. Conclusions and Future Work

We have discussed a distributed object-oriented method for solving N -body problems on clusters. The new method is implemented in Java and RMI protocol in order to support distributed computing in heterogeneous network environment. Distributed objects can work together in the model of collaborative system. Such a system is established on the available resources in a cluster. The DOO method, with the architectural-neutral features of Java and RMI, can be considered as a promising model for distributed and parallel computing in heterogeneous environment. The model can be further expanded to a wide-area distributed computing. It also provides the opportunity to dynamically configure a computing environment whenever the computing resources have been changed. So the new method will yield high flexibility, portability and adaptability.

In our N -body algorithm, we proposed a distributed tree structure to reduce the communication overhead. The runtime tests show that our method is effective for

solving N -body problems on distributed and heterogeneous systems. The DOO method is also feasible for other scientific computing applications.

In our future research, we plan to make the method adaptive to the network computing environment. Compute engines (remote objects) can join the collaborative system at runtime when they are available, and the computing tasks can be redistributed to the compute engines based on the status of the collaborative system. The method will be extended to more wide-range distributed computing.

References

1. Singh, J. P., Hennessy, J. L., Gupta, A.: Implications of Hierarchical N -Body Methods for Multiprocessor Architectures. *ACM Transactions on Computer Systems*, Vol. 13, 2 (1995) 141-202
2. Barnes, J., Hut, P.: A Hierarchical $O(N \log N)$ Force-Calculation Algorithm. *Nature*, Vol. 324, 4 (1986) 446-449
3. Greengard, L., Rokhlin, V.: A Fast Algorithm for Particle Simulations. *Journal of Computational Physics*, Vol. 73, (1987) 325-348
4. Hernquist, L.: Hierarchical N -body Methods. *Computer Physics Communications*, Vol. 48, (1988) 107-115
5. Hu, Y. C., Johnsson, S. L., Teng, S. H.: A Data-Parallel Adaptive N -body Method. *Proceedings of 8th SIAM Conference on Parallel Processing for Scientific Computing* (1997)
6. Salmon, J., Warren, M. S.: Parallel, Out-of-core methods for N -body Simulation. *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing* (1997), also available at <http://www.cacr.caltech.edu/~johns/pubs/siam97/>
7. Singh, J. P., et al.: Load Balancing and Data Locality in Adaptive Hierarchical N -body Methods: Barnes-Hut, Fast Multipole, and Radiosity. *Journal of Parallel and Distributed Computing*, Vol. 27, 2 (1995) 118-141
8. Grama, A. Y., Kumar, V., Sameh, A.: n -body Simulation Using Message Passing Parallel Computers. *Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing* (1995) 355-360
9. Khokhar, A., Shaaban, M., Prasanna, V., Wang, Cho-Li: Heterogeneous Computing: Challenges and Opportunities. *IEEE Computer magazine*, Vol. 26, 6 (1993) 18-27
10. Berman, F., Wolski, R.: Scheduling from the Perspective of the Application. *Proceedings of Symposium on High Performance Distributed Computing* (1996), also available at <http://www-cse.ucsd.edu/groups/hpcl/apples/apples.html>
11. Farley, J.: *Java Distributed Computing*. O'Reilly & Associates Inc (1998)
12. Keren, A., Barak, A.: Adaptive Placement of Parallel Java Agents in a Scalable Computing Cluster. *ACM 1998 Workshop on Java for High-Performance Network Computing* (1998), also available at <http://www.cs.ucsb.edu/conferences/java98/program.html>
13. Woo, S. C. and et al.: The SPLASH-2 Programs: Characterization and Methodological Considerations. *Proceedings of the 22nd Annual International Symposium on Computer Architecture* (1995), also available from <http://www-flash.stanford.edu/SPLASH>

14. Bhatt, S., Chen, M., Cowie, J., Lin, C. Y., Liu, P.: Object-Oriented Support for Adaptive Methods on Parallel Machines. *Scientific Computing*, Vol. 2, (1993) 179-192
15. Liu, P., Wu., J. J.: A Framework for Parallel Tree-Based Scientific Simulations. *Proceedings of 26th International Conference on Parallel Processing* (1997)