Asian Computing Science Conference, Lecture Notes in Computer Science xxx, pp. x-xx, Springer-Verlag, Hanoi, Vietnam, December 2002.

Solving the N-Body Problem with the ALiCE Grid System

Dac Phuong Ho[1], Yong Meng Teo[2] and Johan Prawira Gozali[2]

[1]Department of Computer Network, Vietnam National University of Hanoi
144 Xuan Thuy Street, Hanoi, VIETNAM
hdphuong@vnuh.edu.vn
[2]Department of Computer Science, National University of Singapore
3 Science Drive 2, SINGAPORE 117543
teoym@comp.nus.edu.sg

Abstract. The grid enables large-scale aggregation and sharing of computational resources. In this paper, we introduce a method for solving the N-body problem on a cluster-grid using our grid system, ALiCE (Adaptive and scaLable internet-based Computing Engine). The modified Barnes-Hut algorithm allows the N-body problem to be solved adaptively using compute resources on-demand. The N-body program is written using ALiCE object programming template. Our experiments varying the number of bodies per task and the number of computation nodes demonstrate the feasibility of exploiting parallelism on a grid system.

1 Introduction

The grid [4, 5] promises to change the way we tackle complex problems. It enables large-scale aggregation and sharing of computational, and data resources across institutional boundaries. As technologies, networks, and business models mature, it is expected to become commonplace for small and large communities of scientists to create "Science Grids" linking their various resources to support human communication, data access and computation.

The N-body problem is the study of how *n* number of particles will move under one of the physical forces. Modern physics has found that there are only four fundamental physical forces, namely: gravity, electro-magnetic, strong nuclear, and weak nuclear. These forces have a few things in common: they can be expressed by using simple formulas, they are all proportional to some properties of a particle (mass, electrical charge, etc.), and they get weaker the further apart the particles are from each other [3]. However, very small differences in initial conditions of an N-body problem can lead to unpredictably differing results.

The simplest and most straightforward manner of modeling the N-body problem is the direct method. Here, each pair of particles is visited in turn to calculate and accumulate the appropriate forces. Conceptually, this solution is implemented as two nested loops, each running over all particles. This algorithm scales as N-squared, and cannot be considered sufficiently efficient to enable the solution of interesting (large N) problems. A galaxy might have, to say 10^{11} stars. This suggests repeating 10^{22} calculations. Even using an efficient approximate algorithm, which requires $N\log_2 N$ calculations, the number of calculations is still enormous ($10^{11}\log_2 10^{11}$). It would require significant time on a single processor system. Even if each calculation takes one microsecond, it would take 10^9 years for N^2 algorithm and almost a year for $N\log_2 N$ algorithm. Therefore, there is a need for a better and faster way to solve the N-body problem.

This paper presents a distributed object-oriented method for solving the N-body problem using our grid system ALiCE (the Adaptive and scaLable internet-based Computing Engine). ALiCE also makes it possible to solve the N-body problem adaptively, using resources on-demand. Our experiments showed a reduction in parallel execution times as we increase the number of available compute resources.

This paper is organized as follows. Section 2 presents the Barnes-Hut algorithm - an algorithm that uses a divide-and-conquer strategy to recursively sub-divide the N-body problem space to facilitate calculation of inter-particle distances. Section 3 presents ALiCE – our Java-based grid computing system [6]. The mapping of the N-body problem onto the ALiCE framework is presented in Section 4. We present our experimental results in Section 5, and the summary of our work in Section 6.

2 Barnes-Hut Algorithm

Suppose we compute the gravitational force on the earth from the known stars and planets. A glance skyward on a clear night reveals a dauntingly large number of stars that must be included in the calculation, each one contributing a term to the force sum.

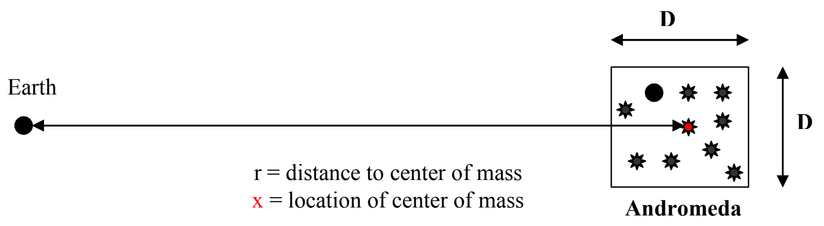
D
Earth
D
r = distance to center of mass
x = location of center of mass
Andromeda

Figure 1: Viewing the Andromeda Galaxy from Earth

One of those dots of light we might want to include in our sum is, however, not a single star (particle) at all, but rather the Andromeda galaxy, which itself consists of billions of stars. These stars appear so close that they appear as a single dot to the naked eye. We model the Andromeda galaxy as a single point, located at the center of mass of the Andromeda galaxy and with a mass equal to the total mass of the Andromeda galaxy. This is indicated in Figure 1, with an "x" marking the center of mass.

Since the ratio $\frac{D}{r}$, is so small, we can accurately replace the sum over all stars in Andromeda with one term at their center of mass. *D* denotes the width and height of the box containing Andromeda, and *r* is the distance between Andromeda and Earth centre of masses. We explain this in more detail below.

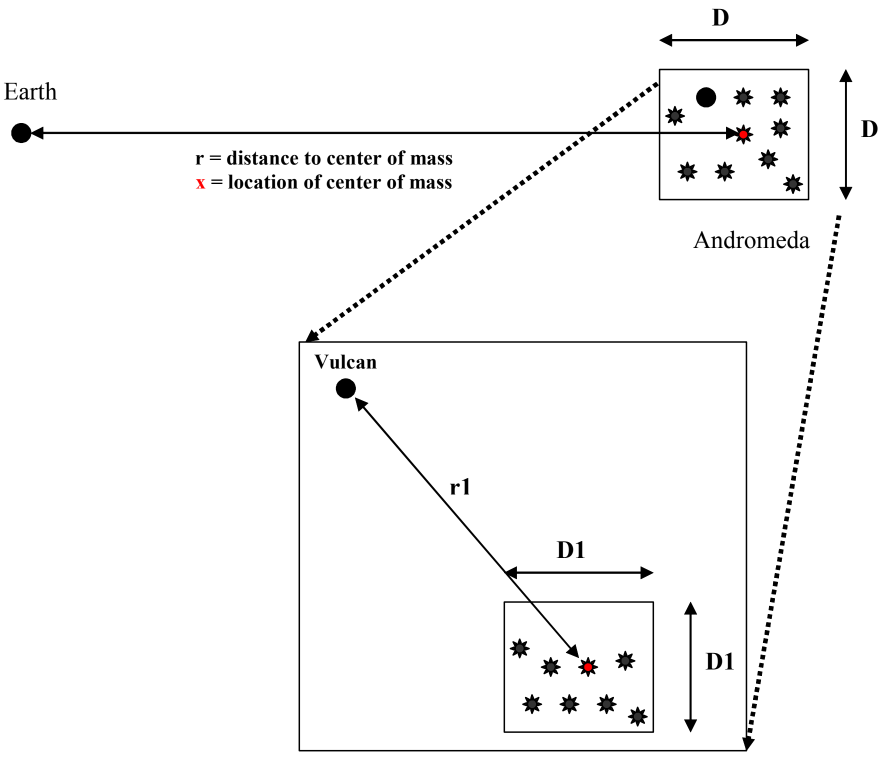
D
Earth
D
r = distance to center of mass
x = location of center of mass
Andromeda
Vulcan
r1
D1
D1

Figure 2: Replacing Clusters by their Centers of Mass Recursively

This idea is hardly new. Indeed, Newton modeled the earth as a single point mass located at its center of mass in order to calculate the attracting force on the falling apple, rather than treating each tiny particle making up the earth separately. What is new is applying this idea recursively. First, it is clear that from the point of view of an observer in the Andromeda galaxy that our Milky Way galaxy can also be approximated by a point mass at its center of mass. But more importantly, within the Andromeda (or Milky Way) galaxy itself, this geometric picture repeats itself as shown below: As long as the ratio D1/r1 is also small, the stars inside the smaller box can be replaced by their center of mass in order to compute the gravitational force on, say, the planet Vulcan. This nesting of boxes within boxes can be repeated recursively (see Figure 2).

Details of Barnes-Hut algorithm can be found at [2]. This algorithm consists of two main phases:

a. Creating a data structure to represent the space. For example, in 3D this can be modeled as the *OctTree*, and in 2D using the *QuadTree*.

b. Traversing the tree to carry out the force calculations. This is accomplished by a simple post-order traversal of the tree, i.e., the child nodes are processed before their parent node. Since the primary objective is to maximize efficiency in calculating inter-particle distances, a group of particles "far away enough" is treated as a single particle with a composite mass, located at the center of mass of the array. Thus, for each particle in turn, the tree is traversed starting from the topmost "universe" node. The spatial extent of the node is divided by the distance from the center of mass of the node to the particle. If this quotient is less than a specified quantity called the theta parameter, the particles in the node are "far away enough" to be considered as a single particle. If the quotient is greater than theta, the tree is recursively descended.

2.1 Task Partitioning

With the Barnes-Hut algorithm mentioned above, the first phase of creating tree and calculating the center of mass and total mass must be done serially. But, with the QuadTree created, new positions for each body can be calculated independently. Thus, second phase of the algorithm can be done in parallel.

Another important aspect of the algorithm is that execution time for the first phase is much lower than that of the second phase. In Table 1 below, we can see the greater the number of bodies, the lower the ratio of time between the first stage and the total of time. Therefore, we can reduce total execution time by calculating positions for each body in parallel.

Number of bodies	First Phase Execution Time (s)	Total Sequential Execution Time (s)
1000	1	23
2000	4	193
4000	5	403
8000	9	1537
10000	11	2401
15000	17	5349
20000	23	9428

Table 1: Sequential Execution Time Varying the Number of Bodies

3 The ALiCE Grid System

ALiCE (Adaptive and scaLable internet-based Computing Engine) is a portable software technology for developing and deploying general-purpose grid applications and systems [5]. It virtualizes computer resources on the Internet/intranet into one computing environment through a platform-independent consumer-producer resource-sharing model, and harnesses idle resources for computation to increase the usable power of existing systems on the network.

3.1 ALiCE Consumer-producer Model

Figure 3 shows our ALiCE consumer-producer model. Applications are submitted by the consumer for execution on idle computers (referred to as producers) through a resource broker residing on another computer. The resource broker regulates consumer's resource demand and producer idle cycles, and dispatches tasks from its task pool for execution at the producers. A novel application-driven task-scheduling algorithm allows a consumer to select the performance level for each application.

ALiCE supports *sequential* or parametric computer applications to maximize computer throughput. For *parallel* computer applications, ALiCE breaks down large computations into smaller tasks and distribute for execution among producers tied to a network to exploit parallelism and speedup.

Parallel programming models are supported through a programming template library. Task and result objects are exchanged between consumers and producers through the resource broker.

ALiCE is scalable and is implemented in Java, Java Jini [1] and JavaSpaces [7] for full cross-platform portability, extensibility and scalability. To the best of our knowledge, ALiCE is the first grid-computing implementation in the world developed using Sun's Java Jini and JavaSpaces.

Efficient task scheduling on a non-dedicated distributed computing environment is a critical issue especially if the performance of task execution is important. The main contributing factors include dynamic changes in computer workload and variations in computing power and network latency. ALiCE's load distribution technology is based on a novel application-driven, adaptive scheduling strategy.

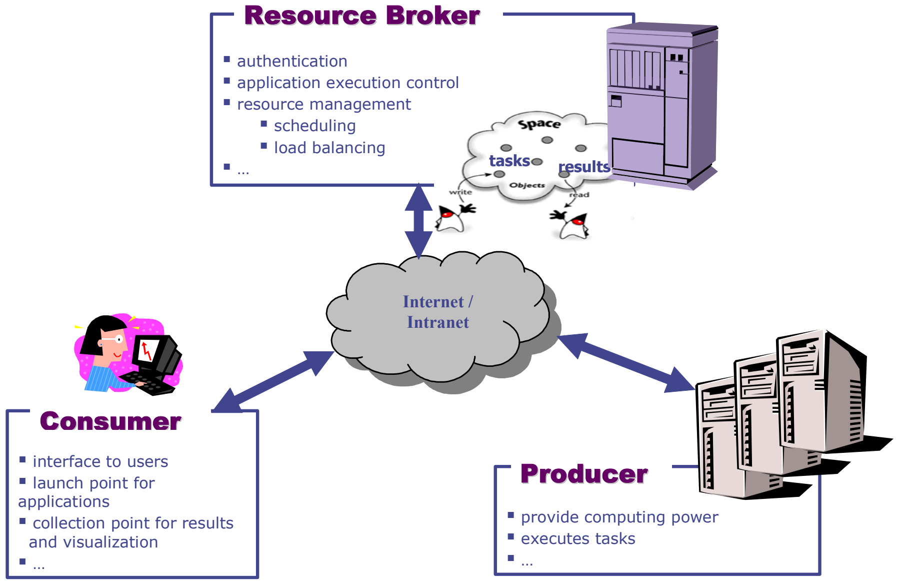

Figure 3: ALiCE Architecture

3.2 ALiCE Components

ALiCE consists of the following main components:

- A programming model consisting of class libraries and a set of design patterns to support both sequential and parallel computer applications.

- A user interface supports the submission of task by consumers.

- A generic computing engine at each producer supports a number of functions. It notifies the resource broker of its availability, monitors and sends its performance to the resource broker, accepts tasks from the resource broker for execution and estimates its execution performance and returns the result to the resource broker.

- A resource broker that hides the complexities of distributed computing, and consists of three main components:

    - Task Manager – This includes a consumer list containing all registered consumers, a task pool containing computer applications submitted by consumers, a task monitor that monitors the progress of task execution, and for storing the application's data and computed results.

    - Resource Manager – This includes a producer list containing all registered producers, a performance monitor containing workload and performance information received from producers, and a security manager.

    - Task Scheduler – Based on the information supplied by the task manager and resource manager, the scheduler performs task assignments by matching the consumer's computational requirement with the available resources in the network.

3.3 ALiCE Object-based Programming Model

The ALiCE Programming Template in Figure 4 implements our *TaskGenerator-ResultCollector model*. This model describes the basic components of a parallel ALiCE application.

The TaskGenerator-ResultCollector model defines two entities: TaskGenerator and ResultCollector. The TaskGenerator is executed at the Resource Broker, and is responsible for generating new tasks. The Resource Broker distributes these tasks to the Producers for execution. The Producers upon completion will send back the results to the Resource Broker which in turn will send the results back to the ResultCollector. The ResultCollector is executed at the Consumer, and it is responsible for collecting results from the Resource Broker.

There are four components that make up the ALiCE programming template: TaskGenerator, ResultCollector, Task, and Result. TaskGenerator component allows application to be invoked at the resource broker by invocation of the user's main method. ResultCollector component allows application to be invoked at the Consumer node, waiting for results to be returned from the Resource Broker. Task component allows the producer nodes to return a Result object to the Resource Broker upon completing the execution. Result component provides an interface for producer to instantiate and returns any evaluated or intermediate data.

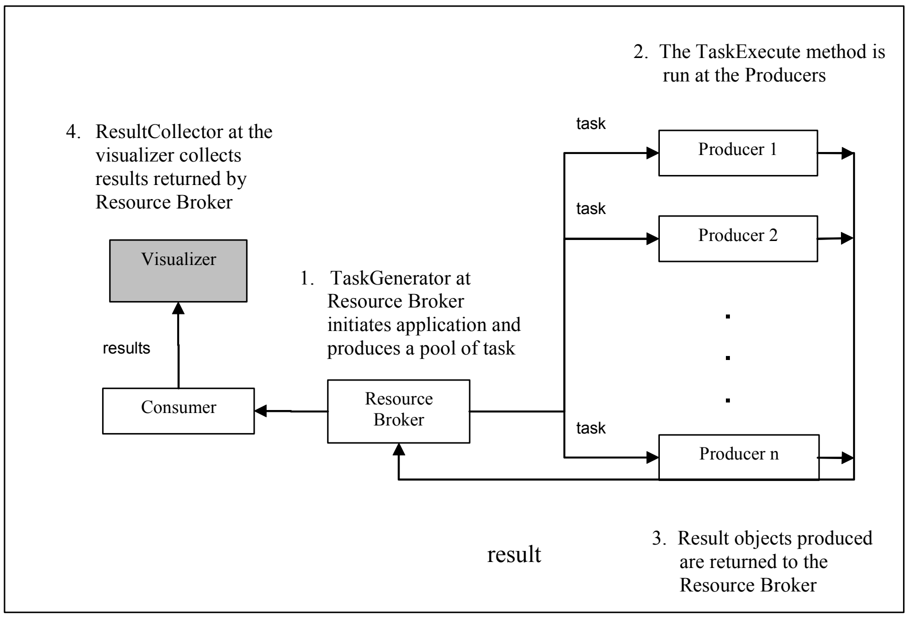
2. The TaskExecute method is run at the Producers
4. ResultCollector at the visualizer collects results returned by Resource Broker
task
Producer 1
task
Producer 2
Visualizer
1. TaskGenerator at Resource Broker initiates application and produces a pool of task
results
Consumer
Resource Broker
task
Producer n
result
3. Result objects produced are returned to the Resource Broker

Figure 4: ALiCE Execution Model

4 Mapping the N-Body Problem onto ALiCE

In our implementation, the Consumer reads data for each body consisting of its position, velocity, and mass, from a file. Data is transferred to the Resource Broker to construct the QuadTree.

The Resource Broker would then send the tree and the body data to the Producers. Each Producer will calculate the new position and velocity of body assigned to it. The result of each calculation is returned to the Consumer.

However, if Producers calculate new positions one body at a time, the volume of data transferred through network is very large, making the network's latency as a possible bottleneck for performance. This is true especially when the number of bodies is large; thus, the ratio between computation time and data transfer time is small.

We can overcome this problem by calculating not only one but many (say *m*) body at each Producer. As the result, we will save (*m-1*) network latency delays as the data for the m bodies is sent to each producer in bulk.

Figure 5 shows the mapping of the application onto the ALiCE architecture.

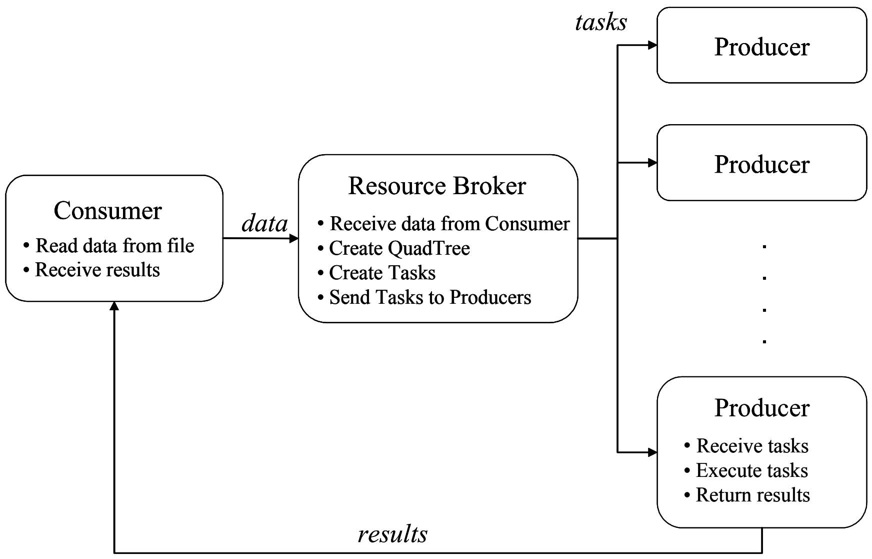

Figure 5: Mapping N-Body onto ALiCE Architecture

Our algorithm takes the number of bodies (*N*) as an argument. The number of tasks that the Task Generator creates is *N/M*, where *M* is the number of bodies executed on a producer. Section 2.1 discussed the partitioning algorithm. We cannot implement the distributed Barnes-Hut trees because with the current version of ALiCE, producers cannot communicate with each other. The N-body program is outline in Figure 6.

```
TASK_GENERATOR
1: A ←.new Tree
2: Initialize (A) //Compute particle mass & center of mass
3: for i in 1 to N/M
4:     T ←.new TASK containing (Tree A, NodeID body[M])
5:     send T to Resource Broker
6: endfor

RESULT_COLLECTOR
1: for i in 1 to N
2:     RESULT R ←.incoming Result from Resource Broker
3:     Write R to the file
4: endfor

TASK_EXECUTE (Tree A, NodeID i)
1: Calculate the total force of all bodies to node i
2: Calculate the new position of M bodies in array body[M]
3: Result R ← new Result
4: Insert new positions into R
5: Return R
```

Figure 6: N-Body Problem Algorithm in ALiCE

5 Experiments

Our experiments were conducted on a cluster of 24 nodes consisting of eight Intel Pentium III 866MHz with 256MB of RAM, and sixteen Intel Pentium II 400MHz with 256MB of RAM running Red Hat Linux 7.0. Nodes are inter-connected via a 100Mbps switch.

5.1 Varying the Number of Bodies

In this experiment, we compare the sequential run with ALiCE (with only one producer). The Producer, Consumer, and Resource Broker are run on Pentium III nodes.

#Bodies	Sequential Execution Time (second)	ALiCE Execution Time for One Producer (second)
100	1	4
200	2	4
500	13	6
1000	23	12
2000	193	41
4000	403	125
8000	1537	1427
10000	2401	4357
15000	5349	33457
20000	9428	43721

Table 2: Sequential and ALiCE (one Producer) Execution Time

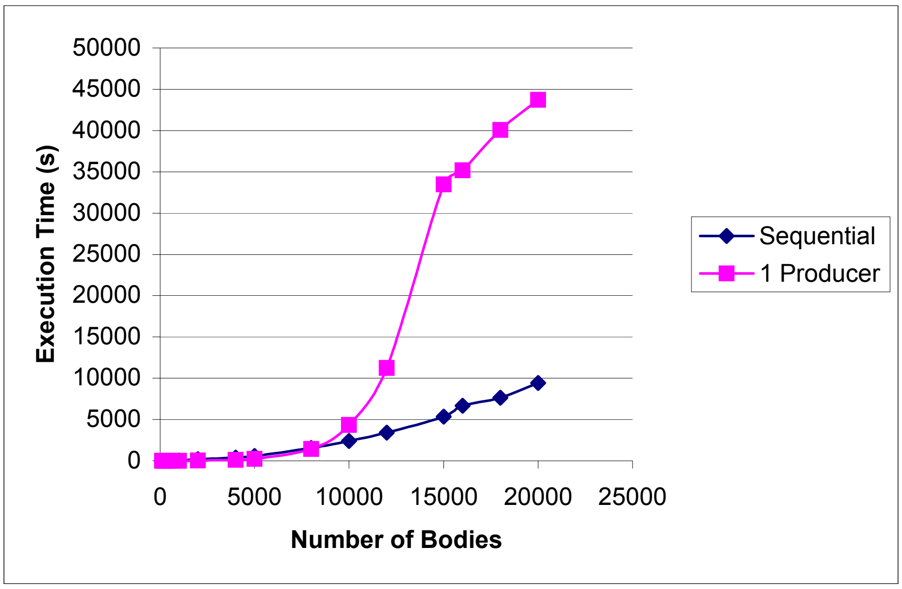

Figure 7: Sequential versus ALiCE with one Producer

We observe that a single producer runs significantly slower than the sequential version. We attribute this slow-down to the communication overhead between the various components of the ALiCE system, i.e. the Resource Broker, the Producer and the Consumer. Such communication overheads do not exist in the sequential version.

5.2 Varying Task Sizes and the Number of Tasks

We vary the numbers of producers and we partitioned the problem into different number of bodies per ALiCE task yielding different number of tasks. The problem size is 25,000 bodies.

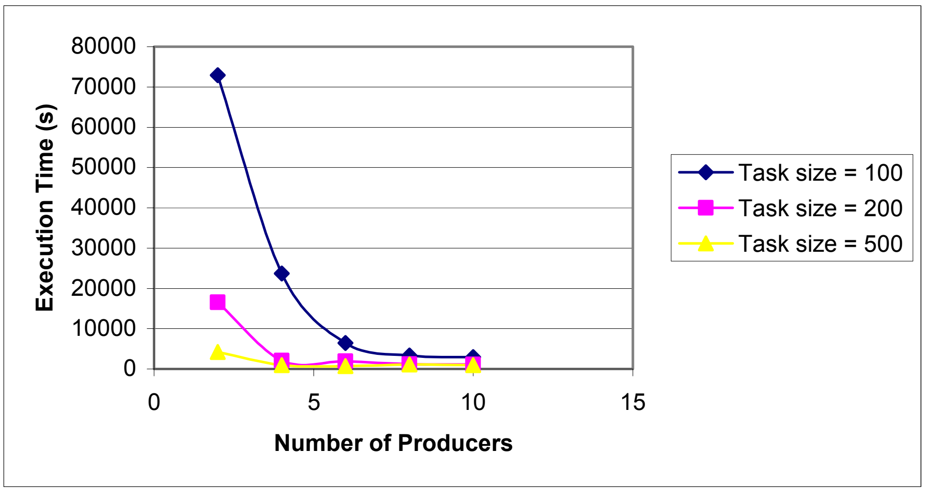

Figure 7: Grid Computation Time for 25,000 Bodies

Task size (#Bodies/task)	#Tasks	#Producers	Execution Time (s)
100	250	2	72951
200	125	2	16574
500	50	2	4218
1000	25	2	1582
100	250	4	23673
200	125	4	2044
500	50	4	932
1000	25	4	957
100	250	6	6476
200	125	6	1900
500	50	6	713
1000	25	6	731
100	250	8	3350
200	125	8	1178
500	50	8	1141
1000	25	8	1239
100	250	10	2910
200	125	10	1100
500	50	10	980
1000	25	10	789

Table 7: Varying the Number of Bodies per Task and Task Size

As shown in Figure 7, the communication overheads due to data transfer and the overhead of ALiCE can be amortized over task of larger granularity. For example, the sequential execution time of over three hours is reduced to 16 minutes on four producers with 25 tasks (1000 bodies per task).

6 Conclusion

We have discussed a distributed object-oriented method for solving N-body problems on a cluster-based grid system using ALiCE that can be extended to include resources in a wide-area distributed computing environment. The method provides *on-demand* computing, i.e. the ability for applications to dynamically adapt to the computing resources available. Our experiments show that our method reduces the time required for solving N-body problems.

References

1. Arnold, K., O'Sullivan, B., Scheifler, W., Waldo, J., and Wollrath, A.: The Jini Specification. The Java Technology Series. Addison-Wesley (1999)
2. Barnes, J. and Hut, P.: A Hierarchical O(N log N) force calculation algorithm. *Nature* (1986) 324:446—449
3. Feynman, R.: The Character of Physical Law. The MIT Press (1965)
4. Foster I. and Kesselman C., editors.: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers (1999)
5. Foster, I., Kesselman, C., and Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications* (2001) 15
6. Gozali, J.P., ALiCE: Java-based Grid Computing System.: Honours Year Thesis. National University of Singapore (2001)
7. Sun Microsystems: JavaSpaces Specification (1998)