Solving the N-Body Problem with the ALiCE Grid System

Dac Phuong HO, Y M Teo## and J P Gozali##

Department of Computer Network
Vietnam National University of Hanoi

##Department of Computer Science
National University of Singapore

Outline

- Introduction
- Barnes-Hut Algorithm
- ALiCE Grid System
- Mapping the N-Body Problem onto ALiCE
- Experiment
- Conclusion

Introduction

A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast.

Almasi and Gottlieb, *Highly Parallel Computing* ,1989

What is Grid Computing?

"... coordinated **resource sharing** and **problem solving** in dynamic, multi-institutional virtual organizations."

N-Body Problem

How *n* number of particles will move under one of the physical forces.

Applications include:

- Astronomy
- Molecular Dynamics
- Fluid Dynamics
- Plasma Physics
- ………..

Physical forces:

- Gravity
- electro-magnetic
- strong nuclear
- weak nuclear

Common

- simple formulas
- some properties of a particle:
  - Mass
  - Position
  - Electrical charge

Newtonian Physics

$$x$$

$$v = x'$$

$$a = v' = x''$$

$$F = ma$$

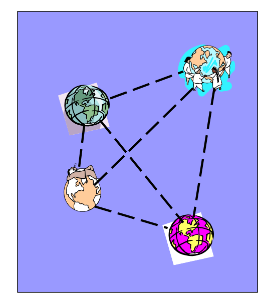
$$F = \frac{Gm_1 m_2}{r_{12}^2}$$

Static and Dynamic Mapping

- The N-body problem: Given n bodies in 3D space, determine the gravitational force F between them at any given point in time.

$$F = \frac{Gm_a m_b}{r^2}$$

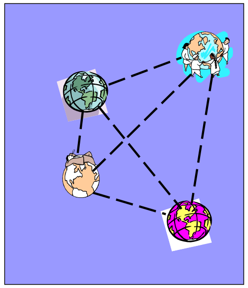
where G is the gravitational constant, r is the distance between the bodies, m_a and m_b are the masses of the bodies.



Exact N-body Serial Algorithm

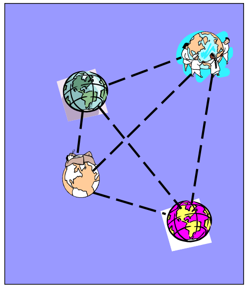
- At each time t, velocity v and position x of body, i may change.
- Real problem a bit more complicated than this.
- For (t=0: t<max; t++)

```
For (t=0: t<max; t++)
  For (i=0; i<N; i++) {
      F= Force_routine(i);
      v[i]_new = v[i]+F*dt;
      x[i]_new=x[i]+v[i]_new*dt;
  }

  For (i=0; i<nmax; i++) {
     x[i] = x[i]_new;
     v[i]=v[i]_new;
     }
```



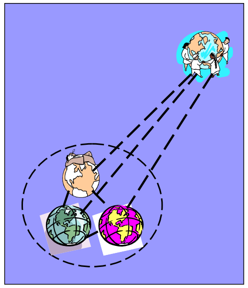
Improving the N-body Algorithm

- Complexity of serial n-body algorithm very large: $O(n^2)$ for **each** iteration.
- Communication structure **not** local – each body must gather data from all other bodies.
- Most interesting problems are when **n is large** – not feasible to use exact method.
- Barnes-Hut algorithm is well-known approximation to exact n-body problem and can be efficiently parallelized.



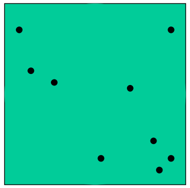
Barnes-Hut Approximation

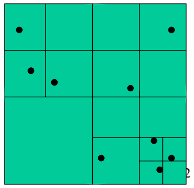
- Barnes-Hut algorithm based on the observation that a cluster of distant bodies can be **approximated** as a single distant body
  - Total mass = aggregate of bodies in cluster
  - Distance to cluster = distance to center of mass of the cluster
- This clustering idea can be applied recursively.



Barnes-Hut idea

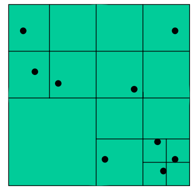
- Dynamic divide and conquer approach:
  - Each region (cube) of space divided into 8 subcubes
  - If subcube contains more than 1 body, it is recursively subdivided
  - If subcube contains no bodies, it is removed from consideration
- 2D example on right – each 2D region divided into 4 subregions

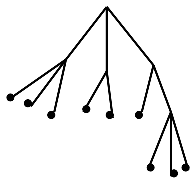




Barnes-Hut Algorithm

- For 2D decomposition, result is a quadtree, pictured below.
- For 3D decomposition, result is an octtree.





Barnes Hut 3D Problem Pseudo-code

- For (t=0; t< tmax; t++) {
  Build octtree;
  Compute total mass and center;
  Traverse the tree, computing the forces
  Update the position and velocity of all bodies
  }

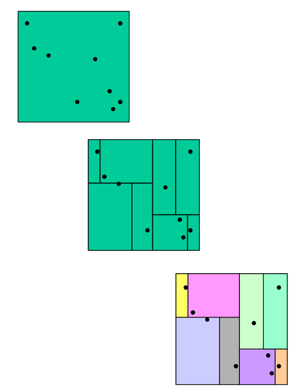
- Notes:
  - Total mass and center of mass of each sub-cube stored at its root.
  - Tree traversal stops at a node when the clustering approximation can be used for a particular body.
    - Need criteria for determining when bodies are in the same cluster.

Complexity of Barnes-Hut Algorithm

- Partitioning is dynamic: Whole octtree must be reconstructed for each time step because bodies will have moved.
- Constructing tree can be done in O(nlog n).
- Computing forces can be done in O(nlog n).
- One iteration of Barnes-Hut is O(nlog n) versus $O(n^2)$ with the exact solution.

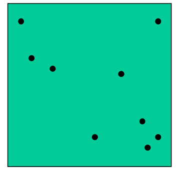
Generalizing the Barnes-Hut Approach

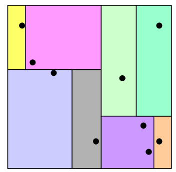
- Approach can be used for applications which repeatedly perform some calculation on particles/bodies/data indexed by position.
- Recursive Bisection:
  - Divide region in half so that particles are balanced each time.
  - Map rectangular regions onto processors so that load is balanced.



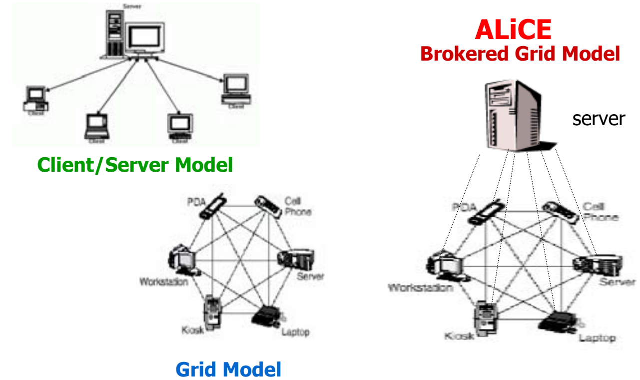
Recursive Bisection Programming Issues

- How do we keep track of the regions mapped to each processor?
- What should the density of each region be? [granularity!]
- What is the complexity of performing the partitioning? How often should we repartition to optimize the load balance?
- How can locality of communication or processor configuration be leveraged?

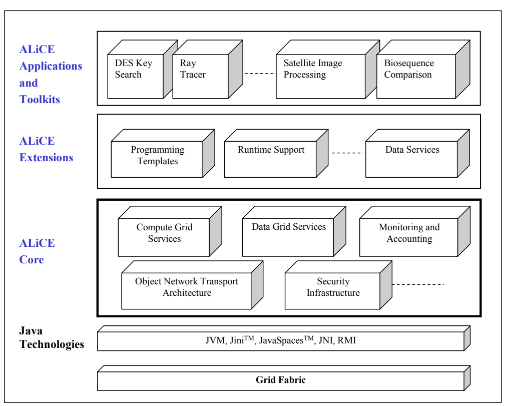




What is ALiCE (Adaptive and scaLable Internet-based Computing Engine)?

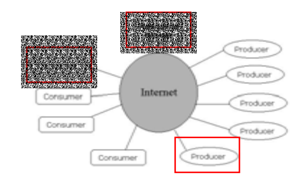


ALiCE Three-Layer Architecture



ALiCE Producer-Consumer Model

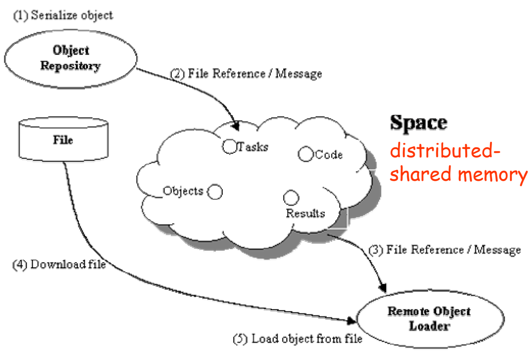
Consumers (C)
- interface to users
- launch point for applications
- collection point for results (visualization)



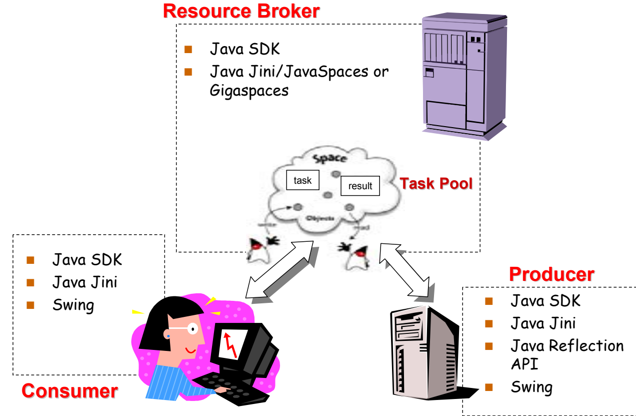
Resource Broker (RB)
- authentication
- application execution control
- resource management
  - scheduling
  - load balancing
- …

Producers (P)
- provide computing power
- executes tasks

Object Network Transfer Architecture



ALiCE Implementation



Programming in ALiCE

Types of Application

1. **Sequential Jobs**
   - parametric computation
   - supports single-tasking programs with well-defined methods like main() or run()

2. **Parallel Job - Object-level Parallelism**
   - exploits object-level parallelism through ALiCE Object Programming Template (AOPT)
   - main motivation is to hide complexities of parallel programming

ALiCE Template-based Programming

Template	Function
TaskGenerator	• Invoked at resource broker • Method to send tasks to producer
ResultCollector	• Visualizer to be invoke at consumer • Method to retrieve results
Task	• Specify functions to execute at producer • Return a Result object
Result	• Interface for producer to instantiate and return result

Java Programming Template

Task Generator Template

```
import alice.consumer.*;
import alice.data.*;
public class TASKGEN_CLASSNAME extends TaskGenerator {
  public TASKGEN_CLASSNAME() {}
  public void init() {
    //Place your initialisation code here
  }

  /* Main method - entry point */
  public void main(String args[]) {
   // This is where the tasks are generated, usually in a loop

   // This should be called for each task
   TASK_CLASSNAME t = new TASK_CLASSNAME();
   process(t);

   // To open a data file, read and write from/to it
   DataFile f = Data.openFile("file_name",this);
   READ_BUFF = f.read(POSITION, LENGTH);
  f.write( WRITE_BUFF, POSITION, LENGTH);

   // To send/receive an object
   OBJECT_CLASSNAME obj = new OBJECT_CLASSNAME();
   sendObject(obj, "snd_str_id");
   OBJECT_CLASSNAME rcvObj = (OBJECT_CLASSNAME)
                              requestObject("rcv_str_id");

   // To receive a string message from the result collector:
  String msg = getStringMessage();
  }
}
```

Task Template

```
import alice.consumer.*;
import java.io.*;
public class TASK_CLASSNAME extends Task {
  // Place variables here
  public TASK_CLASSNAME () {
  }

  public Object execute () {
    // This is where you do your computations. The results can be any kind
    // of objects

    // You can generate and send a new task to be produced
    O_TASK_CLASSNAME t = new O_TASK_CLASSNAME();
    process(t);

    // To open a data file, read and write from/to it
    DataFile f = Data.openFile("file_name",this);
    READ_BUFF = f.read(POSITION, LENGTH);
    f.write( WRITE_BUFF, POSITION, LENGTH);

    // To send/receive an object
    OBJECT_CLASSNAME obj = new OBJECT_CLASSNAME();
    sendObject(obj, "snd_str_id");
    OBJECT_CLASSNAME rcvObj =(OBJECT_CLASSNAME)
                              requestObject("rcv_str_id");
  }
}
```

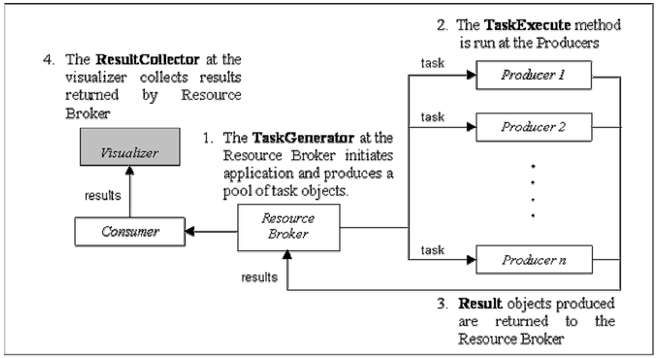
Result Template

```
import java.io.*;

public class MyResult implements Serializable {
  public DATA_TYPE var;
  public MyResult() {
    var=NULL;
  }
}
```

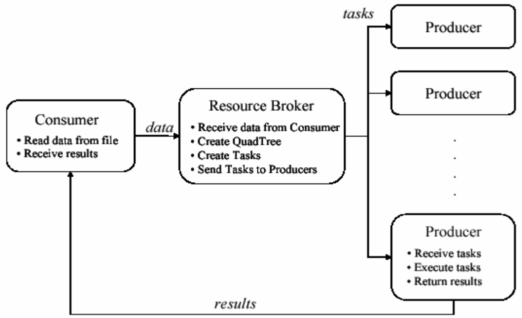
ResultCollector Template

```
import alice.result.*;
public class RESCOL_CLASSNAME extends ResultCollector {
  // Place Variables Here
  public RESCOL_CLASSNAME() {
  }

  public void collect() {
    // Place here the result collection and processing code to obtain
    // number of results ready call
    int resReady = getResultsNoReady()

     // To get a new result call
    RES_CLASSNAME res = (RES_CLASSNAME)collectResult();
  }
}
```


Job (Tasks) Execution



Application Architecture



TASK GENERATOR

A ← new Tree
Initialize (A)
for i in 1 to N/M
  T ← new TASK containing (Tree A, NodeID body[M])
  send T to Resource Broker
endfor

RESULT COLLECTOR

for i in 1 to N
  RESULT R ← incoming Result from Resource Broker
  Write R to the file
endfor

TASK EXECUTE (Tree A, Node i[])

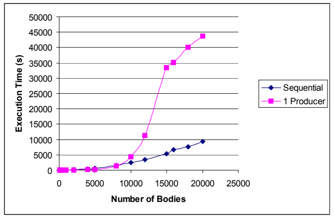
Calculate the total force of all bodies to node i
Calculate the new position of M bodies in array body[M]
Result R ← new Result
Insert new positions into R
Return R

Experiments

- 8 nodes : Intel Pentium III 866MHz with 256MB of RAM.
- 16 nodes : Intel Pentium II 400MHz with 256MB of RAM
- Heterogeneous nodes are inter-connected via a 100Mbps switch.

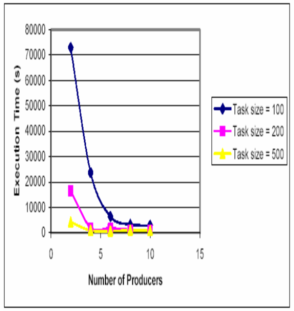
Sequential and ALiCE (one Producer)

#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



Varying Task Sizes and the Number of Tasks

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



Thank you.

Questions

Acknowledgements

Current ALiCE Team:
Johan Prawira Gozali, Ng Yew Kwong, Zheng Yudong, Verdi March, Ameya Virkar, Aditya, Chia Eileen, Wong Keng Choon, Erik Knave (Sweden), Erik Stackenland (Sweden), Lee Yih

Collaborators:
Sun Microsystems, Centre for Remote Imaging, Sensing and Processing (CRISP), Bioinformatics Institute, Nanyang Poly (School of Life Sciences).