Interacting Galaxies: N-Body Cosmological Simulation

Reddy, Prabhat M

ABSTRACT

What is Gravity? If we are to fully understand gravity, we must study it in a number of situations. Tidal disruption of interacting galaxies usually involves some of the largest gravitational forces ever observed in the cosmos. For this reason, it becomes crucial to study how gravity affects these objects. Unfortunately, due to the time it takes for these events to take place, it is not possible to study them by observation. In this paper, we will explore one method of studying these events – simulation. We will detail the process used in producing a simulation of a 2000 object collision, and then explore how to extend this to get more accurate results. We will also explore how such simulations are performed on larger scales in practice. Finally, we conclude.

INTRODUCTION

The governing force in the Universe appears to be gravity. Ever since Newton attempted to mathematically write down the force of gravity, physicists have been working on fully understanding what this force is, and more importantly, its mechanics. Einstein's Theory of General Relativity was another seminal work that redefined our comprehension of gravity. It wrapped up inconsistencies in Newtonian Gravity, the most prominent of which is the inability to explain the precession of the perihelion of Mercury's orbit. To astronomers trying to understand the cosmos, this was a giant leap. It led to a much better understanding of the mechanics of gravity, and dealt with observational discrepancies that classical physics could not explain.

But even as General Relativity cleared up many questions, it raised many more. The one question that currently is considered the Holy Grail of Physics still remains unanswered. "Is there a Theory of Everything? If so, what is it?" So far, three of the four forces – Electromagnetic forces, Strong Nuclear forces and Weak Nuclear forces – have been unified. But Gravity remains separate. It is much weaker and acts over nearly unfathomable distances. Why it is so different from the remaining forces remains a mystery to even the brightest and most dedicated minds.

Additionally, Gravity is believed to be the force that shapes our Universe's fate. Anything with mass feels this force. Even the lightest particles separated by billions of light years still feel the force exerted by the others. And with the immense masses in the Universe, the forces felt are sometimes of inconceivable magnitude. Whether we end up as a really hot, dense, tiny ball, or

expand into infinity until the space between two atoms is extremely large, or anything intermediate, gravity will have a big hand in determining our fate.

In an attempt to understand this force, we turn to interacting galaxies. Billions of stars spread over thousands of light years come together to either merge or destroy each other, all because of gravity. These galaxies pull each other closer for millions of years, if not billions, until they inevitably collide, wreaking havoc. Stars fly in all directions. Some shoot off into space. Some are ripped apart from the conflicting forces. Most stick around and become part of a new larger galaxy. But all this occurs over the course of time spans that cannot be observed. In order to study these interactions, we must speed up time. This is achieved through simulations. By modelling the force of gravity, we can study these collisions of immense proportions from the safety of our planet within a day.

THEORY

While Newtonian Gravity is not exactly right, it is a very good approximation in this case, as none of the objects move at relativistic speeds. We can treat every star as a point mass, as at the distances involved, this approximation is very nearly the truth. Newton's Universal Law of Gravitation tells us that the force due to gravity between any two massive objects is given by

$$F = \frac{GM_1M_2}{r^2} \qquad (1)$$

Where G is the Universal Gravitational constant, M_1 and M_2 are the masses of the two interacting objects, and r is the distance between them.

Now we can calculate the forces between every pair of objects, and use this to find their accelerations by Newton's Second Law of Kinematics

$$F = ma \equiv a = \frac{F}{m} \qquad (2)$$

From the acceleration, we find the average velocity over a small time period dt by the following relation

$$a = \frac{dv}{dt} \equiv v = \int a\, dt \qquad (3)$$

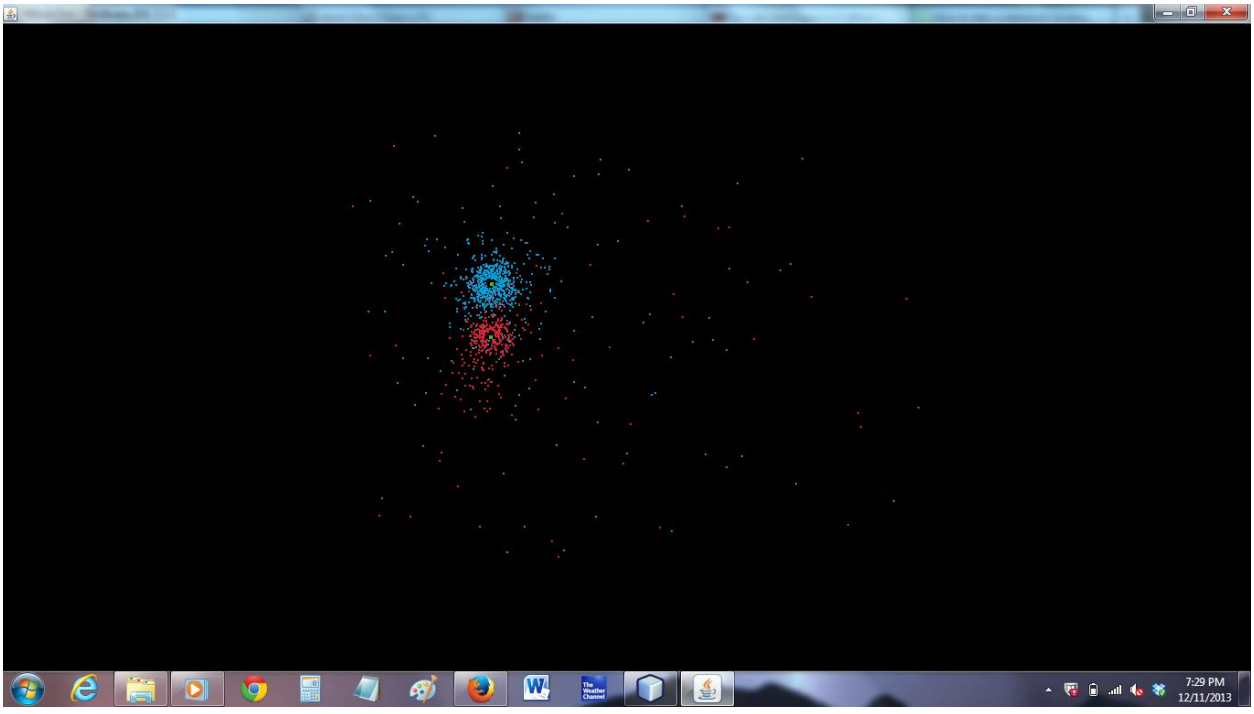
Now we represent each point mass as a dot. For every time step, we calculate the force on each object by equation (1) due to every other object. Then from equations (2) and (3), we can calculate each objects velocity. We then update the positions and recalculate. In this manner, over a range of times, we can create a representation of the particles and their positions. We can collate these images and make a movie out of them, which allows us to see how these galaxies are affecting each other in real time.

IMPLEMENTATION

The code for this simulation was executed in Java. The complete code for this program can be found in Appendix I (Page 8). This program simulates 2000 stars belonging to two galaxies set on a collision course. A star is saved as an object with parameters including mass, position and velocity. The program initializes all 2000 stars with a mass of one solar mass (1 solar mass = 2 x 10^{30} kg). It also sets initial positions and velocities as follows.

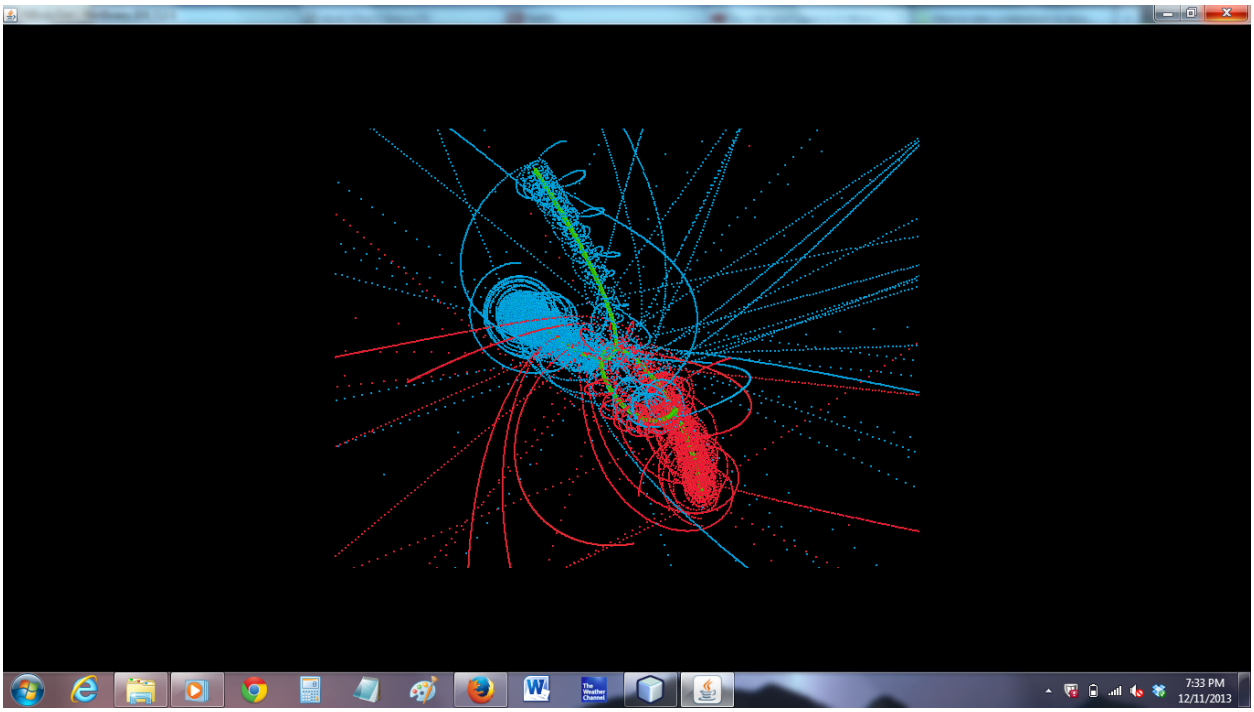
Two-thirds of the stars are in circular orbit around a central mass of 1 million solar masses (hereafter referred to as Galaxy 1). The remaining one-third are in circular orbit around a similar mass of 1 million solar masses (hereafter referred to as Galaxy 2). The larger 'galactic cores' are colored green, and are set at explicitly stated positions with explicitly stated velocities. The stars in Galaxy 1 are colored blue. Their distance from the central core is determined randomly, and the initial velocities are calculated to achieve circular orbit. Similarly, stars in Galaxy 2 are colored red and also follow the same procedure as their Galaxy 1 counterparts.

After the initial conditions have been determined and assigned, a nested pair of loops begin that calculate the forces between every object pairwise and update their positions. The simulation runs for 1000 iterations, and after each iteration the stars are drawn to a screen based on their positions. Each screen is captured and saved as an image. By assembling the images, one can make a movie which represents the interactions of the two galaxies.



This screenshot shows the program running. A video of the simulation is included in the supplemental materials, called GalaxySim.

In addition to this, I implemented a version of this simulation where the orbit of each object is traced. This is quite informative, as one can follow each objects trajectory, learning why it takes the path it does. This also allows us to extend these trajectories to make predictions for where each object might end up. This is achieved by overlaying each frame over the already existent composite image. Below is a screenshot of this program in progress.



A video of this simulation is available in the supplemental materials, labelled as GalaxySimTrace.

DISCUSSION

While these simulations are informative as a general overview, they are not exact. They cannot be relied on for making predictions about observed objects. Those must be simulated with care. In order to recreate the actual collision, one would have to simulate billions of objects.

However, the algorithm used above would not be suitable for this. While it is good for a small number of objects (<10,000), it scales as n^2 where n is the number of objects simulated. To recreate this simulation on a personal computer with a billion objects is problematic. We will explore this below.

Let the number of objects n = 10^9

Let the number of iterations m = 10^3

Let time required per calculation t = 10^{-9} seconds

(The time required per calculation is an optimistic expectation, but roughly constant across most computers as it is imposed by hardware limitations.)

Now the total running time of this algorithm is

$$T(n) = mn^2t$$

$$T(n) = 10^{12}s = 31710\ years$$

More importantly, the heap space (memory used for storing the numbers calculations take place on) available is insufficient for this program. With an 8GB RAM, which is what most PCs have at this time, it is impossible to calculate the positions of each star at all.

However, there is a way out of these problems, allowing us to perform highly detailed simulations with large amounts of objects. It requires a different algorithm and very expensive equipment, but can give results in a reasonable amount of time. We will explore this algorithm, and then re-evaluate our estimate for the expected running time.

BARNES-HUT ALGORITHM

The Barnes-Hut algorithm drastically reduces the number of calculations required by only evaluating the forces nearby individually. For the remaining objects, it groups them into cells, which it treats as a single particle with a collective mass of the total mass in the cell. It does this by implementing recursion trees. This lends it a running time algorithm of O(n logn)[1]. Now let us re-evaluate the expected running time.

Let the number of objects n = 10^9

Let the number of iterations m = 10^3

Let time required per calculation t = 10^{-9} seconds

Now the total running time of this algorithm is

$$T(n) = mn\log(n)t$$

$$T(n) = 9000s = 2.5\ hours$$

This is significant improvement. With this algorithm, simulations with billions of objects can be created and available for study within a few hours, which is feasible. However, memory remains an issue. This is not something that can be improved through code. However, there are other options.

HARDWARE IMPROVEMENTS

One option to deal with the lack of memory is to use supercomputer arrays. These would be able to parallelize the calculations, since each force calculation is independent of the other. The limitations on memory are no longer applicable, and even without the optimistic estimate of time per calculation, we can expect a result within a few hours.

But supercomputer arrays are expensive to construct, and are in high demand. For example, Blue Waters cost almost $400,000,000 to build[2], and took about five years to complete[3]. And while it could give quick results, the waiting time to receive computing time is much larger. It takes two weeks just to process the application.

A cheaper and easier option would be a cluster of PCs, which are now viable options. PCs have vastly improved over the last 10 years, with each having more computing power than some of the more impressive servers in the past. Using Hadoop, or a similar FDS, we can distribute the jobs across hundreds or thousands of PCs, which are much cheaper, but just as effective as a powerful supercomputer. At about $2,000 each, a hundred PC network would cost only $200,000. This is 2000 times cheaper than Blue Waters.

Crowdsourcing is another option which in conjunction with using a cluster of PCs makes it much cheaper. Here, volunteers offer their computers for calculations. While they are not using their PCs, the program runs in the background. This is cheap if not free, and easy to implement. And results are equally good when compared to a supercomputer.

CONCLUSION

In this paper, we evaluated the merits of using N-body cosmological simulations to model interactions between two galaxies. We motivated the need to study such systems, and provided a method to do so. We then followed up with methods to make these simulations even more useful by lending them predictive power. We discussed the limitations of trying to run these simulations on one's own. We then provided options that are more feasible in these situations. Large scale simulations like the Millennium Run by VIRGO Consortium or Eris utilized similar methods[4]. In the future, as computers get better, faster and cheaper, it may become possible to start doing such large scale simulations regularly. Perhaps the invention of Quantum Computer algorithms can vastly speed up the running time. In this way, we can gather as much

data about how gravity behaves theoretically to compare with observed effects. Hopefully, these simulations will assist the quest to understand gravity.

REFERENCES AND CREDITS

Lawrence Angrave – Zen.java

[1] – Josh Barnes & Piet Hut: A hierarchical *O(N log N)* force-calculation algorithm

[2][3] – Paul Wood (November 14, 2011): Cray Inc. replacing IBM to build UI supercomputer

[4] - Powerful supercomputers allow first simulation of Milky Way-like galaxy: The Indian Express

APPENDIX I

```
import java.awt.Dimension;

import java.awt.Rectangle;

import java.awt.Robot;

import java.awt.Toolkit;

import java.awt.image.BufferedImage;

import java.io.File;

import javax.imageio.ImageIO;


class params{


    public static int t = 0;


}

class Star{

  private static final double G = 6.673e-11;

  private static final double solarmass=1.98892e30;


  public double rx, ry;

  public double vx, vy;

  public double fx, fy;

  public double mass;

  public String img;
```

```
public Star(double rx, double ry, double vx, double vy, double mass, String img) {
  this.rx    = rx;
  this.ry    = ry;
  this.vx    = vx;
  this.vy    = vy;
  this.mass  = mass;
  this.img = "star.gif";
}

public void update(double dt) {
  vx += dt * fx / mass;
  vy += dt * fy / mass;
  rx += dt * vx;
  ry += dt * vy;
}

public double distanceTo(Star s) {
  double dx = rx - s.rx;
  double dy = ry - s.ry;
  return Math.sqrt(dx*dx + dy*dy);
}

public void resetForce() {
  fx = 0.0;
```

```
    fy = 0.0;
  }

  public void addForce(Star s) {
    Star a = this;
    double EPS = 3E4;
    double dx = s.rx - a.rx;
    double dy = s.ry - a.ry;
    double dist = Math.sqrt(dx*dx + dy*dy);
    double F = (G * a.mass * s.mass) / (dist*dist + EPS*EPS);
    a.fx += F * dx / dist;
    a.fy += F * dy / dist;
  }

  public String toString() {
    return "" + rx + ", "+ ry+ ", "+  vx+ ", "+ vy+ ", "+ mass+ " "+img;
  }
}

class StarTest{
    public int N = 100;
    public Star stars[]= new Star[1000000];

  public static double circlev(double rx, double ry) {
```

```
  double solarmass=1.98892e30;
  double rx2 = rx*rx;
  double ry2 = ry*ry;
  double r2=Math.sqrt(rx2+ry2);
  double numerator=(6.67e-11)*1e6*solarmass;
  return Math.sqrt(numerator/r2);
}

public void startthebodies1(int N) {
  double radius = 1e18;
  double solarmass=1.98892e30;
  for (int i = 0; i < N; i++) {
    double px = 1e17*exp(-1.8)*(.5-Math.random());
    double py = 1e17*exp(-1.8)*(.5-Math.random());
    double magv = circlev(px,py);

    double absangle = Math.atan(Math.abs(py/px));
    double thetav= Math.PI/2-absangle;
    double phiv = Math.random()*Math.PI;
    double vx   = -1*Math.signum(py)*Math.cos(thetav)*magv;
    double vy   = Math.signum(px)*Math.sin(thetav)*magv;
        if (Math.random() <=.5) {
          vx=-vx;
          vy=-vy;
```

```
        }
    double mass = Math.random()*solarmass*10+1e20;
    stars[i]   = new Star(px, py, vx, vy, mass, "star.gif");
     System.out.println(stars[i].toString());
    if(i>2*N/3){
     stars[i]   = new Star(px+4e17, py+4e17, vx, vy, mass, "star.gif");
    }
  }
  stars[0]= new Star(0,0,1000,0,1e6*solarmass,"star2.gif");
  stars[2*N/3] = new Star(4e17,4e17,-1000,-10000,1e6*solarmass,"star2.gif");
 }


 public void combine(int n){
    for(int i = 0; i < n;i++){
       for(int j = 0; j < n; j++){
         if(i!=j && stars[i].mass != 0 && stars[j].mass !=0){
          if(Math.floor(stars[i].rx/(Math.pow(10,12))) ==
Math.floor(stars[j].rx/(Math.pow(10,12))) && Math.floor(stars[i].ry/(Math.pow(10,12))) ==
Math.floor(stars[j].ry/(Math.pow(10,12)))){
              double m = stars[i].mass + stars[j].mass;
              stars[i].vx = (stars[i].mass*stars[i].vx + stars[j].mass*stars[j].vx)/m;
              stars[i].vy = (stars[i].mass*stars[i].vy + stars[j].mass*stars[j].vy)/m;
              stars[i].mass = m;
              stars[j].mass = 0;
         }
```

```
        }
      }
    }
}

public void addforces(int N) {
  for (int i = 0; i < N; i++) {
    stars[i].resetForce();
    for (int j = 0; j < N; j++) {
      if (i != j) {
          stars[i].addForce(stars[j]);
      }
    }
  }
  for (int i = 0; i < N; i++) {
    stars[i].update(1e11);
  }
}
 public static double exp(double lambda) {
     return -Math.log(1 - Math.random()) / lambda;
  }

 public void captureScreen(String fileName) throws Exception {
```

```
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   Rectangle screenRectangle = new Rectangle(screenSize);
   Robot robot = new Robot();
   BufferedImage image = robot.createScreenCapture(screenRectangle);
   ImageIO.write(image, "png", new File(fileName));

}

  }

class Main{
    public static void main(String[] args) throws Exception {
        StarTest st = new StarTest();
        int n = 2001;
        st.startthebodies1(n);
        int t = 0;
        while(t < 10000){
        for(int i = 0; i <n; i++){
              if(i == 0 || i == 2*n/3 && st.stars[i].mass != 0){
                 Zen.drawImage("star2.gif", (int) (200+(st.stars[i].rx/(2*Math.pow(10,15)))), (int)
(200+(st.stars[i].ry/(2*Math.pow(10,15)))), 4, 4);
              }
              else if(i<2*n/3 && st.stars[i].mass != 0){
                 Zen.drawImage("star.gif", (int) (200+(st.stars[i].rx/(2*Math.pow(10,15)))), (int)
(200+(st.stars[i].ry/(2*Math.pow(10,15)))), 2, 2);
```

```
            }
            else if(st.stars[i].mass != 0){
                Zen.drawImage("star1.gif", (int) (200+(st.stars[i].rx/(2*Math.pow(10,15)))), (int)
(200+(st.stars[i].ry/(2*Math.pow(10,15)))), 2, 2);
            }
          }
       st.addforces(n);
       t++;
       st.captureScreen("starsimagetrace" + t);
       Zen.flipBuffer();
     }
   }
}
```