Parallelized Visualization of N-body Cosmological Dark Matter Simulations

Greg Dooley
MIT 18.337 Parallel Programming

Dec 17, 2012

Abstract

300,000 years after the Big Bang, the universe consisted of a nearly perfectly homogenouse distribution of baryonic matter and dark matter. Over the next 13.7 billion years, gravitational forces caused overdense regions to collapse into the galaxies we observe and live in today. Astrophysicists like to further understand and test this process with massive gravitational simulations of the universe. One natural thing to do is to visualize the output. Since datasets can range from 4 to 30 GB, computing projected density fields can be a long computational task. Computing the density in parallel can make dramatic speed improvements and hence is a very useful thing to do. Having access to several $\sim$ 4 GB datasets, I found and implemented a visualization technique and made it parallel using julia code. The parallelization reduces what is almost a 2 hour serial problem into 3 minute computation when 50 processors are used.

1 Background

The universe consists of on the order of 100 billion galaxies. Each galaxy is composed of stars at its center in a long flattened shape, like a disk, surrounded by a roughly spherically symmetric distribution of dark matter. Dark matter is a nearly collosionless, non self-interacting substance that interacts graviationally, but not with light. It makes up $\sim$ 84% of the matter content of the universe. Evidence of dark matter surrounding galaxies, called a halo, exists from observations and simulations. These structures can be simulatd by modelling the distribution of dark matter in the early universe, assuming a certain cosmology, assuming a temperature of the dark matter, approximating the density distribution with

particles, then evolving that distribution under its own gravitational dynamics. Simulations output the position and velocity of every particle, $134,217,728$ particles of the same mass in the data set used here. With this information it is possible to visualize the density field and see the overdense halos that would surround galaxies like the Milky Way. The volume of the simulation is a cube of specified size with periodic boundary conditions.

2 Simulation Code

The N-body code used to produce the simulations was GADGET 2 written by Volker Springel [1]. GADGET 2 is a smoothed particle hydrodynamics code that evolves collisionless N-body particles gravitationally. At each adaptive timestep it approximately solves Poisson's equation for gravity: $\nabla^2\Phi = 4\pi G\rho$. In order to make the computation tractible, it computes local gravitaional forces with a hierarchical multipole expansions, and long range forces with faster, but less accurate fourier techniques. For visualization purposes, this process is only important in that it defines a proscription for computing density fields, and the density field is what I aim to visualize.

Gravitating objects in the simulation are represented as massive particles with a precise location. The real matter distribution of the universe however is much more smooth. Simulated orbits around point sources leads to unphysical paths where energy and entropy conservation is violated. To avoid these issues, the mass of a particle is smoothed out in a spherically symmetric distribution with radius h, the smoothing length. The smoothing length is chosen such that the estimated mass in the volume $\frac{4}{3}\pi h^3\rho$ is a constant. Thus low density regions will have a large smoothing length, and high density regions a small smoothing length. Particles are smoothed out according to the Monaghan kernal $W(r,h)$ given by equation 1 and plotted in Figure 1. Note that no density contributions are made beyond a radius $r > h$.

$$W(r,h) = \frac{8}{\pi h^3}\begin{cases} 1 - 6\left(\frac{r}{h}\right)^2 + 6\left(\frac{r}{h}\right)^3, & 0 \leqslant \frac{r}{h} \leqslant \frac{1}{2}, \\ 2\left(1 - \frac{r}{h}\right)^3, & \frac{1}{2} < \frac{r}{h} \leqslant 1, \\ 0, & \frac{r}{h} > 1. \end{cases} \tag{1}$$

3 Computing Density

In order to produce a two dimensional density projection of the simulation, the volume of interest and the dimensions in pixels of the output image must be specified. Each pixel stores

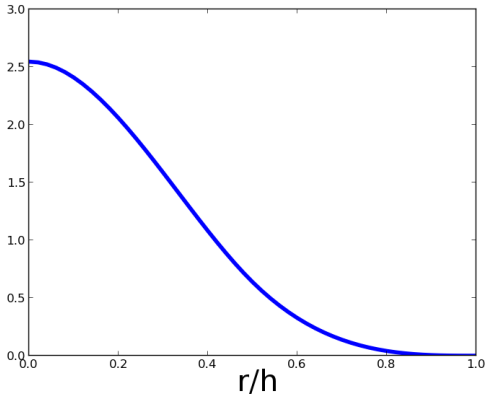

Figure 1: Shape of Monaghan cubic spline used to smooth out the density of particles.

the sum of all density conributions made by nearby particles summed in a column over the entire depth (3^{rd}) axis. That is, if ρ_i is the density of pixel i, m_j is the mass of particle j, h_j is the smoothing length of particle j, and r_{ij} is the distance between particle j and the center of pixel i, then

$$\rho_i = \sum_{j=1}^{N} m_j W(|r_{ij}|, h_j) \tag{2}$$

where N is the total number of particles and W is the kernal function defined in Eq. 1. The overall density is computed by looping over every particle and adding density values to each pixel whose center falls within a radius h from the particle center. This is visualized in figure 2. Two minor complications occur when the smoothing radius is too small for any contributions to be made, see Figure 3, and when the region of interest to draw encompasses the entire size of the box. A minimum h is enforced to ensure each particle contributes to at least one pixels. When the entire width, height, or depth is visualized, periodic boundary conditions are applied, but otherwise are not.

4 Data Sets Used

Two types of data sets were used for visualizations. One was a set run by a recent MIT PhD graduate in astrophysics on a computer cluster at Harvard. Each simulation contained $134,217,728$ particles, had 3.3 GB of position data, and 1.1 GB of smoothing data. The boxsize was 25 megaparsecs, which is enough to contain $\sim 100,000$ galaxies. There were six different simulations each run with slightly different input cosmologies. The effect is

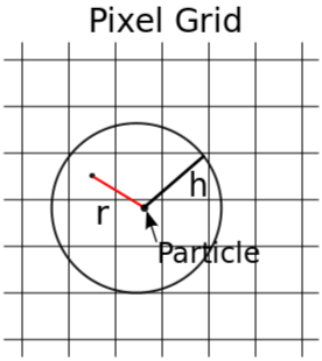

Figure 2: The pixel at the end of the red line will gain a density contribution of $W(r, h)$ from the particle drawn

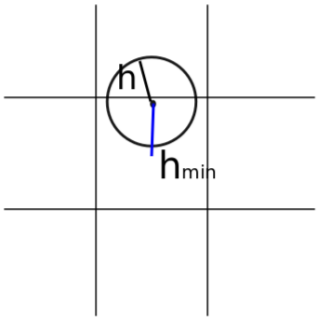

Figure 3: Smoothing radius too small for particle to contribute anywhere. h will be adjusted to contribute to nearest pixel.

simulations with the same structures, but slight deviations from each other. See Figure 6 for these visualizations. A second data set was run by the Aquarius Consortium [2] in 2008. It only contains a single galaxy, with a box size of 100 kiloparsecs, and $606,866,170$ particles. It has 16 GB of position data, and I do not have any smoothing data for the box. I assumed a value of .25 Kpc for the smoothing length of all particles.

5 Parallelizing the Code

All code to produce the density grid is written in Julia. A user can specify an arbitrary number of processors, up to the machine's limit, on the command line to run the code (i.e. ¿¿ julia -p 5 parallelVis.jl). The code splits up its data set accordingly and combines its result at the last step, following the general flow listed below.

1. Read file headers to determine number of particles
2. Allocate space in distributed arrays to store position and smoothing length data
3. Read all data in parallel into distributed arrays
4. Wait for reading tasks to finish
5. Calculate density projection onto pixel grid on each processor's own set of data
6. Sum the results

The following parallel Julia commands were used for the specified reasons:

- *remote_call_wait* - load reading and density computation functions to each processor
- *darray* - used to allocate distributed array space for positions and smoothing lengths
- *@spawnat* processor ID - used to spawn reading and computing tasks
- *RemoteRef* - to keep track of when tasks finish
- *wait* - wait for reading tasks to finish
- *fetch* - combine computed results into one density field.

6 Parallelization Performance

The particle data is stored in multiple files, and I previously had a python program to read the data one file at a time. Translating this code into Julia led most naturally to either reading in data in serial, or on at most X processors, where X is the number of files. Reading the data in serial, then using the call *distribute* to distribute the data into a distributed array on 5 processors took 13.9 seconds. In order to take advantage of parallel reading, I rewrote the read-in files to read any specified range of contiguous particles, whether they stretch over many files or not. I then also allocated space in a distributed array for each processor, and read equal segments of data in parallel. With 5 processors, the read time was reduced to 4.6

seconds. The performance gain by reading the data in parallel is quickly saturated by 12 processors as seen by the red line in Figure 4.

An oddity in performance I noticed was in the order in which I spawned processes on each processor. If I spawned a computational task on the master processor first, there would be a long delay of about 1 second before the next task was spawned. Thereafter each task was spawned in quick succession, about .001 seconds per spawn. By reversing the order, and spawning a task on the master processor last, I avoided that extra second of wait time. However, the the first spawned task still always took longer at ~ 0.055 seconds. I interpreted this as the master node is slowed down on its other operations by spawning a task, and that spawing a task of a particular operation once puts something in local memory than is used to spawn the same task again faster.

The performance gain on parallelizing the density calculation is much more dramatic. I ran timing tests on $\frac{1}{4}$ of the data set from 1 processor to 50 processors, and plotted the compute time vs. number of processors on a log-log scale in Figure 4. The ideal case of time $\propto n_{proc}^{-}1$ is also plotted. As can be seen, the actual performance is almost ideal, deviating noticeably only after 20 processors. Thus this visualization compuation is an excellent program to take advantage of parallelization.

As the size of the data set is increased, more time is spent moving data across processors and the performance gain is less ideal, but still excellent. For example, with 30 proccessors on 1/4 of the data set, it took 63.24 seconds to compute. With 30 processors on the full data set, it took 284.6 seconds, or 31 seconds longer than the anticipated scaling if only computing, not data movement, was a factor. As the number of pixels specified increases, the number of pixels each particle contributes to increases as N^2, so the total compute time increases as N^2. With the dataset held fixed, parallelization becomes more ideal as the number of pixels is increased.

7 Visualization

A user specifes the size of the grid in pixels to produce. He also must specify the physical dimensions of the region of interest to visualize, the center position of the visualized region, the side of the box to view (e.g. X-Y, Z-X, Z-Y), and whether or not each dimension is periodic. Julia writes the output data to file, where I then read it back in with a Python script. The values of $\log_{10}(\rho)$ are normalized into a $0-1$ scale. I meticulously selected colors at different thresholds between 0 and 1 to color the image. The color of densities in between the chosen thresholds is interpolated between the two bounding colors. Results of

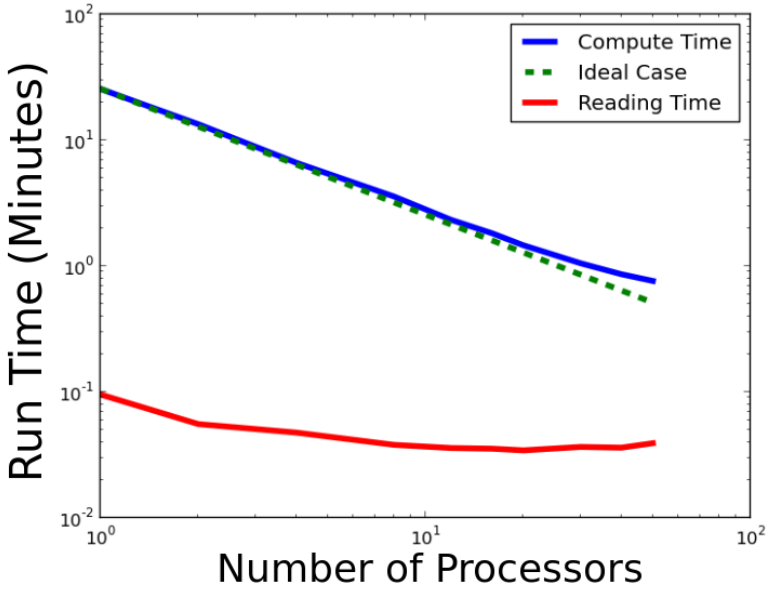

Figure 4: Run time on 1.1 GB of position and smoothing data, $\frac{1}{4}$ of data set. Parallel performance gained on reading in data quickly saturates after $N \approx 5$ processors. Parallel performance on density computations close to ideal performance up to 50 processors.

this process are shown in Fig 6 and 5. I additionally wrote a code that computes the density field in three dimensions, but was not able to install a reasonalbe 3D visualization program in time to view the 3D images.

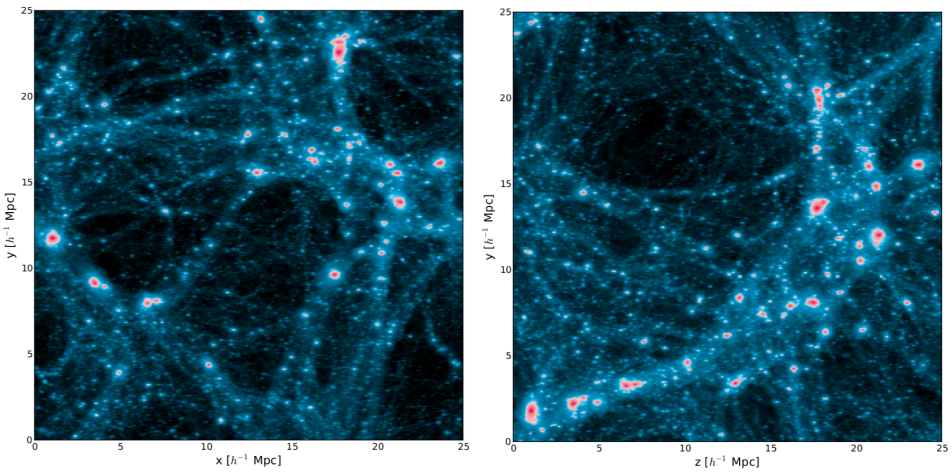

Figure 5: X-Y axes view of simulation on left and Z-Y axes view right.

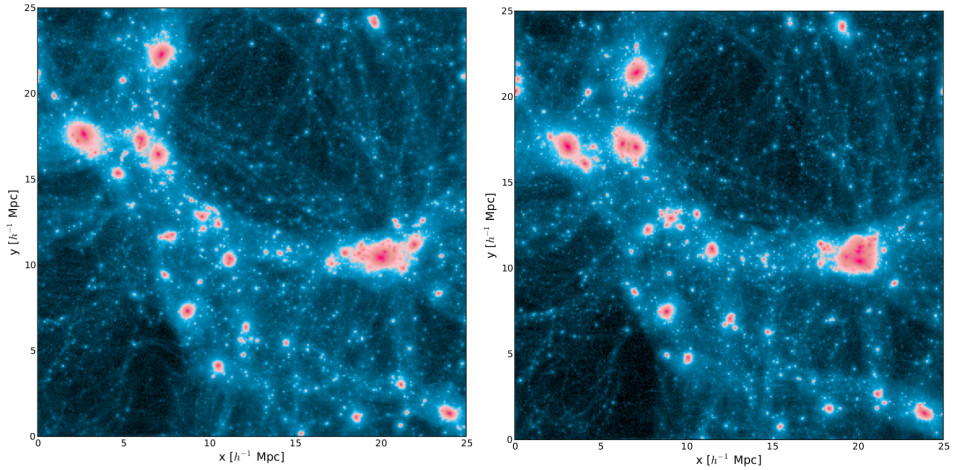

Figure 6: Visualization of the same region of space with two different cosmologies.

References

[1] V. Springel. The cosmological simulation code GADGET-2. *mnras*, 364:1105–1134, December 2005.

[2] J. F. Navarro, A. Ludlow, V. Springel, J. Wang, M. Vogelsberger, S. D. M. White, A. Jenkins, C. S. Frenk, and A. Helmi. The diversity and similarity of simulated cold dark matter haloes. *mnras*, 402:21–34, February 2010.