COLATUS: an alternative GPGPU/CUDA N-body simulator for cosmology

D. H. Stalder[a,*], R. R. Rosa[a], R. R. de Carvalho[b], P. Barchi[a], A. Bomfin Jr.[a], R. R. Ruiz[a], H.F.C. Velho[a], J. R. Silva[c], E. Clua[c]

[a] *Lab for Computing and Applied Mathematics - LAC*
National Institute for Space Research - INPE
São José dos Campos, SP, Brazil
[b] *Astrophysics Division - DAS*
National Institute for Space Research - INPE
São José dos Campos - Brazil
[c] *Institute of Computing - IC*
Universidade Federal Fluminese - UFF
Niteroi, RJ, Brazil
version Mar 07, 2017

Abstract

Gravitational N-body problem in cosmology is traditionally addressed with HPC simulations involving massive nonlinear calculations, which demand technological solutions optimizing the physical representation, the algorithm complexity and the hardware acceleration. Then, the more efficient these simulations are, the more accurate the large scale surveys will be in addressing the main unsolved cosmological challenge which is the interpretation of the cosmic dark content. Dark matter, in particular, presents a strong constraint with the characteristics of the cosmological model adopted to feed a structure formation simulation based on N-bodies. In this context, the present work involves the application of GPGPU/CUDA technology for simulation of gravitational large-scale structure formation ($8Mpc/h < L < 128Mpc/h$), incorporating new simulation features, as lagrangian turbulent tracers and topological potential deformations, that allow testing alternative theoretical proposals as well as physical refinements to the standard cosmological model ΛCDM. First simulations comprising a number of elements in the order of 10^6 are set to the same ranges of redshift, boundary conditions and initial conditions considered in the usual simulations that do not use the GPU technology. Comparative studies involving the velocity dispersion, calculated also on results obtained from the Virgo consortium, evaluate positively the performance of this GPGPU simulator ($COLATUS$) for alternative applications and reduced preliminary tests in computational cosmology.

Program summary

Title of program: COLATUS (*COsmic LAgrangian TUrbulence Simulator*)
Program obtainable from: `https://bitbucket.org/diegostalder/colatus-public`
Workstation for which the program is designed and others on which it has been tested:
(i) Jaburu: processor Intel Core I7 950 Quad-Core + Nvidia GeForce GTX 580 1.5GB GDDR5.
(ii) Parayba: processor Intel Sandy Bridge-E Core I7 3930K Six-Core + two Nvidia GeForce GTX 680 2GB GDDR5.
(iii) LAC-Hybrid-CLuster: 2 CPU processor Intel 10-Core + 2 GPUs Nvidia K20, (4 Nodes).
Operating systems or monitors under which the program has been tested: Ubuntu 14.04 and Bright Cluster System with the NVIDIA SDK COMPUTING 6.5
Programming language used: C, C++, CUDA
Has the code been vectorized or parallelized?: Parallelized in threads by using GPUs (Graphic Processor Units)
Number of thread used: Every thread take care about one particle. The user can select the number of particles.
Keywords: many-core arquitecture, hybrid computing, GPGPU coprocessor, particle-particle comoving N-body code for cosmological simulations.
Nature of physical problem: hybrid collisionless N-body code for the calculation of the gravitational force in 3D galaxies distribution traking, in a machine learning environment, the nonlinear path of single particles.
Method of solution: It is based on the particle-particle scheme for the force calculations and the leap-frog with variable time-step for the integration.
Restrictions on the complexity of the program: The program integrator scheme could be changed by the user.
Typical running time: 65 seconds for each time-step, running a 80^3 particles simulation, on a NVIDIA K20 with 2496 CUDA cores.
Unusual features of the program: COLATUS uses particle-particle scheme for tracking the particles paths during the simulations; allows to insert topological deformation potential for simulating nonhomogeneuos matter distribution; and also incorporates a machine learning resource for each experiment.
Restrictions: Nvidia Cuda Tooklit version 6.5 or later should be installed. To use GPU realizations, Nvidia GPU supporting CUDA and the corresponding Nvidia driver should be installed.

Keywords: Gravitational N-body simulation, Large-scale structures, Cosmology, GPGPU-CUDA computing technology
PACS: 98.80.-k, 98.65.At, Cw, Dx, 95.35.+d, 95.75.Pq

1. Introduction

The gravitational N-body simulations have become a powerful tool for testing the theories of structure formation in astrophysical and cosmological systems [1, 2, 3]. Over the past twenty years, great progress has been made in developing computer codes and hardware that are capable of simulating the evolution imposed by the standard cosmological model ΛCDM (incluir duas referencias), which in its most elaborate versions, constrained by observational data, may include the gravitational interaction, hydrodynamics of gas and radiative processes which allow us to consider even the star formation [4] (mais ref Yoshida CCP).

The first generation of simulators of N-bodies to astrophysics and cosmology was developed during the 70s. The main feature of the first generation, is the use of the direct sum of interaction known as PP scheme whose complexity is $\approx O(N^2)$. Within this paradigm DSPP (*Direct Sumation Particle-Particle*), were developed nearly a dozen simulators for computational cosmology [5, 6, 7].The second generation of N-body simulations sought to introduce simplifications capable of decreasing the computational complexity of algorithms for $\approx O(NlogN)$, then have been developed the hierarchical tree algorithm [8, 9], the particle-mesh (PM) method and the hybrid ones P^3M, Tree Particle-Mesh (TreePM), etc [2]. The first code of the second generation for computational cosmology fully based on massively parallel architecture of CPUs (Central Processing Units), to be categorically available and widely used by the international scientific community was GADGET (*Galaxies with Dark matter and Gas interact*)[10, 11].

In recent years, there has been a rapid development of the coprocessors technologies based on many-cores architecture, specially using General Purpose Graphics Processing Unit (GPGPU), making them attractive for hybrid high-performance computing (HHPC). The HHPC gave rise to the gravitational N-body simulations from where theoretical peak performance is now greater than five teraflops [12, 13]. These fact motivate the develop of the third generation of N-bodies simulators which are based on HHPC. The HHPC started in the early 2000s by adopting DSPP algorithm with a 2nd order integrator [14]. A few years later the N-Body implementations in NVIDIA GPGPU/CUDA was improved to include individual time steps, adapting the hierarchical tree algorithms and higher order integrators [15, 16, 13].

With the advent of Machine and Deep Learning techniques a fourth generation is already on the horizon. Machine learning-based analytics (MLBA) of very large data sets could uncover correlations that are too complex for the ordinary human analytics. Intrinsic spatio-temporal and parametric difficulties in analyzing very large data sets from massive N-body Hubble volume simulation will be technically faced. For example, it is know that a particle is involved in a complex and unique Lagrangian pathway during the gravitational collapse due to the gravitational instability plus the rich environment effects. From the simulation it is know its end state, but not the entire Lagrangian pathway. Trying to figure out by chance which trajectory is correct involve a large set of time-intensive and expensive experiments. However, applying a machine learning algorithm to the data set, find correlations between variables involving specific particles can bring a kind of knowledge hypothesis. Then, a MLBA can set up and run a simulation that tests such hypothesis.

Whatever the generation of simulators used for the study of large-scale structure formation, it must provide a set of output data that can be analyzed in an increasingly refined way by exploiting as much as possible the intelligent data mining to compare key features of the simulation with those from the astronomical observation. In addition, it should allow, through the phenomenological scope of the algorithm, to test and validate, in addition to the ΛCDM, alternative processes, theories and models that allow a deeper understanding of the nature and relationship between dark matter and dark energy.

In the above context, we present here a computer program based on HHPC, using GPGPU / CUDA technology, which incorporates new simulation features where we highlight the following: an explicit lagrangean particle tracer, a Machine Learning module for automatic detection of snapshots, and a parameter adjustment scheme that allows to substitute the dark matter for the topological deformation potential predicted in some alternative models (eg cusp ref and backreaction).

In section 2 we describe the COLATUS scheme including, where necessary, the motivations that lead us to incorporate novelties into a program of this type. The practical operation of the program is demonstrated in sections 3 and 4. In section 5, we present a summary and perspectives. The formalism of the equations of motion and integration scheme are presented in the Appendix.

2. The COLATUS Scheme

The current COLATUS scheme is an improved version of the prototype developed five years ago when at that time it was designed to run on a HHPC containing only one GTX580 graphics card from NVIDIA. Many details of the current scheme are redundant with that of the prototype and therefore should be consulted in Stalder et al.

The expansion of the Universe is considered using the comoving coordinates for monitoring expansion using the scale factor $a(t)$ considering $\mathbf{x} = \frac{\mathbf{r}}{\mathbf{a(t)}}$ and $\mathbf{v} = \mathbf{u} + \mathbf{H}\,\mathbf{r}$, where usually the $(t_f) = 1$(today = current period of the Universe), $H(t) = \frac{\dot{a}}{a}$ is the Hubble parameter (now H_0) and $\mathbf{u} := \mathbf{a}\dot{\mathbf{x}}$ is the peculiar velocity. The equations of motion in comoving coordinates for the system with N particles are:

$$\frac{d\mathbf{x_j}}{dt} = \mathbf{u_j}, \qquad \frac{\mathbf{du_j}}{\mathbf{dt}} + \mathbf{H u_j} = -\nabla\mathbf{\Phi_j}, \tag{1}$$

$$\nabla\mathbf{\Phi_j} = \frac{F_j}{a^2 m} = \frac{G}{a^2}\sum_{k\neq j}^{N}\frac{m(\mathbf{x_k}-\mathbf{x_j})}{\left(\mid \mathbf{x_k}-\mathbf{x_j}\mid^2 + (\frac{\epsilon}{\mathbf{a}})^2\right)^{3/2}}. \tag{2}$$

Explicar opcao pela GPGPU, e referenciar On the next section we discuss the updated computational bottleneck on

*Correponding Authors
Email address: diego.stalder@inpe.br (D. H. Stalder)

the algorithms to solve the equations and obtain the particles position and velocities. The initial and the boundary conditions used are described on the next sub-sections.

2.1. The bottleneck: Force Calculations

The calculation of the interaction forces for each particle is one of the tasks that demand more computational resources, hence in recent years have developed a wide variety very efficient algorithms for this purpose [1, 2]. But given that our aim is to extract information of the dynamics of an individual particle we consider the simplest possible algorithm method direct sum, it evaluates the particle-particle interactions resulting in a computational complexity of $O(N^2)$.

So let F_{jk} the force that the particle exerts on k particle j, it is given by the following equation:

$$F_{jk} = G\frac{m_j m_k(\mathbf{x_j} - \mathbf{x_k})}{\left(\mid \mathbf{x_j} - \mathbf{x_k} \mid^2 + (\frac{\epsilon}{\mathbf{a}})^2\right)^{3/2}}, \tag{3}$$

therefore the sum of the forces acting on the particle j is:

$$F_j = G\sum_{k\neq j}^{N} \frac{m_j m_k(\mathbf{x_j} - \mathbf{x_k})}{\left(\mid \mathbf{x_j} - \mathbf{x_k} \mid^2 + (\frac{\epsilon}{\mathbf{a}})^2\right)^{3/2}}, \tag{4}$$

in equation 3 is observed that $F_{jk} = -F_{kj}$, then at least are calculated sums of $N(N-1)/2$, however the complexity is $N(N-1)/2 \propto O(N^2)$.

2.2. Integration scheme

We consider the approach introduced by [17], on the CPU we can integrate the Friedmann Equation using the Taylor approximations as follows:

$$a^{n+1} = a^n + \dot{a}^n dt + \ddot{a}^n \frac{dt^2}{2} + O(dt^3). \tag{5}$$

here it is appropriate to change the time by a parameter p that power is the scale factor, ie $p = a^\alpha$ [18]. The size of the time step Δp is calculated from the norm of the maximum velocity and acceleration of the particles ([19]) as follow:

$$\Delta p \leq \eta \sqrt{\frac{\epsilon_x}{a_{max}}} \quad \text{and} \quad \Delta p \leq \eta \frac{\epsilon_x}{u_{max}} \tag{6}$$

where by default the value $\eta = 0.2$ is generally used [20, 19], for each iteration the time step if necessary is subdivide into multiples of two. The Layzer-Irvine conditions (**incluir ref.**) are used as a dynamical test of the accuracy of the solution.

2.3. Boundary Conditions

A general presumption that simulations carried in cosmology are the periodic boundary conditions (PBC), these conditions are appropriate to suppress boundary effects. To this end, Ewald summation has been employed to handle long ranged interactions in PBC [21]. In the context, two types of truncation is often used to reduce the cost of the real-space sum, namely the minimum image convention and spherical cutoffs [22].

For this first implementations we use the minimum image scheme, that consider 26 boxes replicas of a central box during the simulation(on each face and each edge). The positions and velocities of the particles are the same replicas of the central box. If a particle moves from the central box, the particles of the replicas perform the same movement. If a particle leaves the central box it comes to the opposite side with the same speed, ie the number of particles is conserved. Each particle interacts with exactly $N - 1$ particles that are inside a fictitious box of size L centered on the particle. The total number of interactions is therefore $\frac{1}{2}N(N-1)$.

2.4. Initial Conditions

The observations analyzed by [23] indicate that the spectrum of primordial fluctuations is Gaussian with scale-invariant, also called Harrison-Zeldovich spectrum, ie $P_{initial}(k) \propto k^n$, where $n \approx 0.9646$ and k is the wave number. The density of primordial fluctuations is modified during the evolution of the Universe until reionization. **To calculate the final spectrum are considered plasma, dark matter , the density of neutrinos and photons. ??** Finally , the initial conditions are obtained from the final spectrum of fluctuations . To represent the density fluctuations, the positions and velocities are disturbed from a grid of particles distributed in the cosmological box, using linear perturbation theory and the code N-GENIC [24]. **We adapted (?)** N-GENIC code that has been written several years back (2003) by Volker Springel N-GENIC and was used for [10].

2.4.1. Homogeneous Cosmological Constraints

The mass of each particle is defined from the matter density , the critical density ρ_c , the volume of box L^3 and the number of particles N^3 , as follows :

$$m = m_j = \Omega_M \frac{\rho_c L^3}{N^3} \quad \forall j \tag{7}$$

where Ω_M is the density of matter (more baryonic dark).

2.4.2. Nonhomogeneous Cosmological Constraints

The design of optimal ways to measure and extract information from the large scale structure of the Universe. The simulation of alternative cosmologies to LCDM, such as quintessence dark energy. In those systems, as in fluid turbulence, the PSD should be scaling invariant. Therefore, the alternative resources of COLATUS works as a computational basis for testing backreactions due to inhomogeneous matter distribution, as addressed, for example, in *timescape cosmology* (ref Wiltshire, PRD, 2009; Saulder et al., 2012), *cusp cosmology* (ref.Rosa et al, 2012) which take into account the role of primordial topological deformation potential, and similar theories.

2.4.3. The Supervised Lagrangian Pathway

The statistical characterization of dark matter distribution has been shown that is an important ingredient in the investigation of large-scale structure formation in the Hubble volume simulated from the GADGET algorithm [11].

Recently, an established statistical method was used to demonstrate the importance of considering chaotic advection

(or Lagrange Turbulence) [25, 26] in combination with gravitational instabilities in the ΛCDM simulations performed from the Virgo Consortium (VC) [27]. However, the GADGET-VC algorithm does not allow the computation of the kinematics of a single particle, information that is necessary for the investigation of the chaotic advection. The *COsmic LAgrangian TUrbulence Simulator* (COLATUS) is developed to perform gravitational N-body simulations allowing the computation of the position and the velocity of a single particle at every time-step and using the Graphics Processing Units (GPUs) and the DSPP algorithm.

Recently, an (**?**) was used to demonstrate the importance of considering chaotic advection (or Lagrange Turbulence) [25, 26] in combination with gravitational instabilities in the ΛCDM simulations performed from the Virgo Consortium (VC) [27]. However, the GADGET-VC algorithm does not allow the computation of the kinematics of a single particle, information that is necessary for the investigation of the chaotic advection. The *COsmic LAgrangian TUrbulence Simulator* (COLATUS) is developed to perform gravitational N-body simulations allowing the computation of the position and the velocity of a single particle at every time-step and using the Graphics Processing Units(GPUs) and the DSPP algorithm.

For example, knowing the initial conditions, the extreme value statistics of dark matter clumps for $z < 50$ is an important complementary approach in the investigation of nonlinear processes involved in the ΛCDM large-scale structure formation [25, 26].

In this context, we introduce ongoing effort in machine learning to enrich the output knowledge about each simulation from COLATUS. With data from previous hundred(s) simulations, the machine learning module acts to predict the average outsnap in which the system of current simulation will enter in relaxation, before collapsing. To illustrate, slice projections of the particles dispersion from the simulation hypercube are represented in differnet moments of the simulation: at the beginning (Figure 1a), at the intermediate moment between the beginning and the start of relaxation 1b; at the moment the system enters in relaxation (Figure 1c); and, at the collapse moment (Figure 1d).

We use only slope as input feature (red line represented in Figures 1a, 1b, 1c and 1d) which is the Power Spectral Density (PSD) value. Figure 2 shows the slope variation over time: in the beginning of the simulation, slope $\simeq 0,001$; it almost reaches 0 and then ascends to $\simeq 0,010$; after reaching the global maximum, it decreases to relaxation (where the vertical gray line cuts the graph) and approximately stabilizes till collapse moment.

To perform this specific problem-driven task, a heuristic method searches for the approximate slope value of relaxation, followed by estabilized slope values till collapse. This is done by *COsmic LAgrangian TUrbulence Simulator - Relaxation Search* (COLATUS-RS) algorithm described as follows.

The search algorithm starts by setting the initial outsnap as the candidate for solution. In the following steps of the method, it verifies if the next outsnap is a better candidate for the slope value, considering the standard deviation: if so, this current slope value is the current candidate, and the search continues; else, the solution candidate is kept and the search continues. If COLATUS-RS finds a slope value that is the desired solution considering the standard deviation, it checks if this value is in the graph ascension, i.e., if the value of the previous outsnap/candidate is lower (situation shown in Figure 2 around outsnap 25). If so, this current value is the current candidate, and the search goes on. Else, the previous outsnap has a higher slope value than the current one, and so we have come to our solution (represented by the dashed gray line on the graph of Figure 2). The search continues till the end of the outsnaps sequence to ensure that the system slope does not change much and this is in fact the relaxation of the system, before collapsing.

This algorithm is always applied in 10 slices separated regularly, to get the mean and standard deviation of the hypercube simulation. Before predicting, COLATUS-RS must be applied to hundreds simulations to get the slope mean and standard deviation for the relaxation outsnap which is our ideal fitness to be found as solution in the next simulations. Thus, machine learning module indicates the relaxation moment by verifying in real-time the slope from the simulation iterations.

2.5. GPU/CUDA Implementation

a) mais obre o CUDA (ver Changa). b) ha um Library como no chamomile? c) e o pipeline?

A CUDA program consists of one or more steps that are executed on the host(CPU) or on the device(GPU). The part that present data parallelism is implemented on the device and the part that requires less processing implemented and the host. To exploit the processing power of the GPU kernels the functions should instantiated in large number of threads, some of them which are executed at the same time. In our implementation, the calculation of force and numerical integration are performed on the GPU, only the time-step, and the scale factor are computed on the CPU.

The modifications necessary to perform a simulation of COLATUS using the CPU (host) and GPU (device) are shown on the **FigureXX**. In this case, to initialize the simulation, in addition to reading the input files and allocate memory in the host, we must also allocate memory in the device, and after loading the initial conditions in the host memory, we need to copy them to the device. The work is distributed between the CPU and GPU. The execution is initialized and controlled by the host that invokes the kernel function **ComputeAcceleration** to generate a one-dimensional grid of N threads in device, where each thread will be responsible for calculating the acceleration of a particle in an interaction. The particle i in which each thread is responsible is determined using the identifiers (*blockIdx* and *threadIdx*) and the equation 8:

$$ID = blockIdx \times \textbf{nthreadpblock} + threadIdx, \tag{8}$$

where **nthreadpblock** is the number of *threads* of each block. This quantity is defined from the specifications of the GPU used to maximize the occupation of each core multiprocessors. So dividing the number of particles N the number of threads per

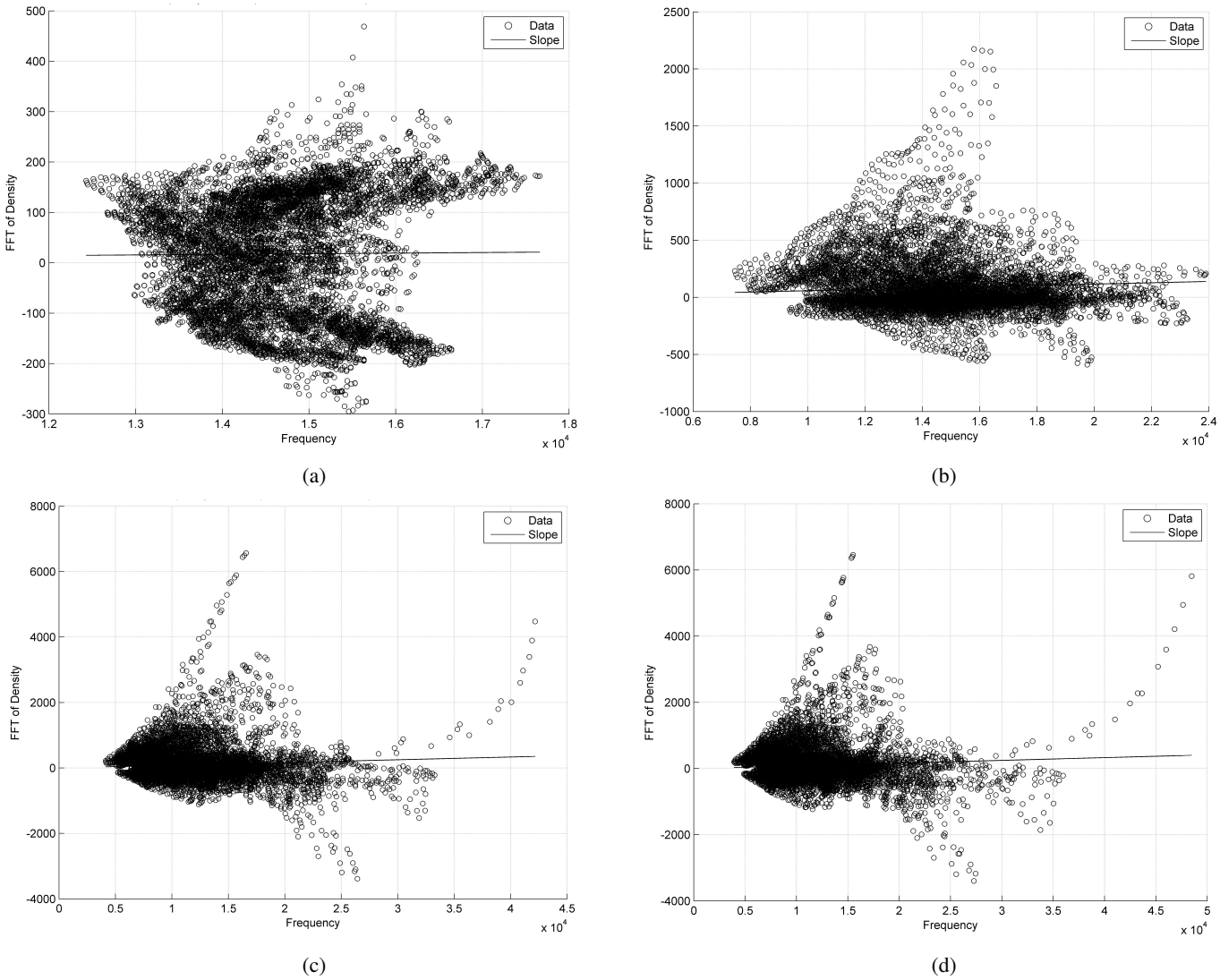

Figure 1: Projections of hypercube slice from a COLATUS simulation at the beginning (1a), intermediate moment between beginning and relaxation 1b, relaxation (1c), and collapse (1d) moment.

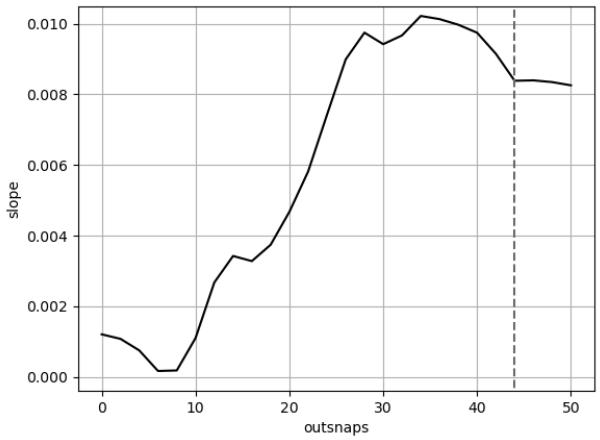

Figure 2: Graph of lope variation over time in this simulation slice. The vertical dashed gray line marks the beginning of relaxation.

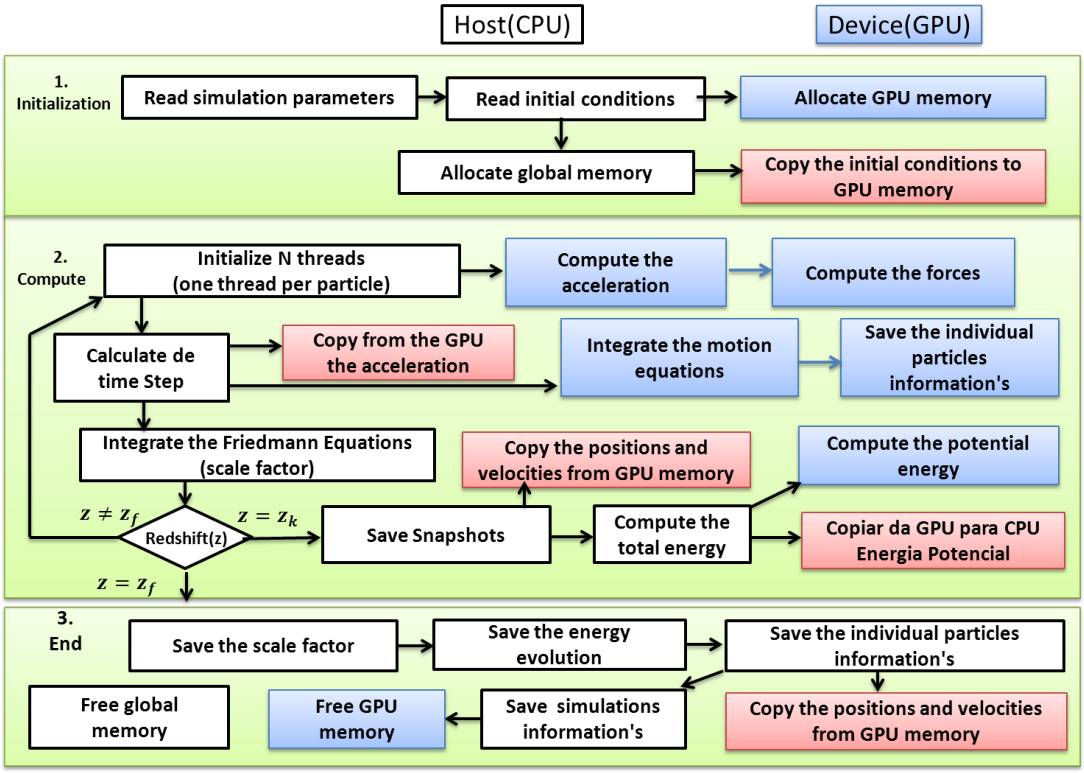

Figure 3: Scheme of N-body simulation on the GPU

block nthreadpblock, and rounding the result to the entire subsequent know how many blocks will be generated on each *kernel* invocation.

Each kernel function **ComputeAcceleration** need to sum the $N-1$ forces acting on the particle for which he is responsible. For calculating each force the kernel we use the **bodyBodyInteraction** device function. The acceleration is computed from the sum of the forces acting on each particle and the maximum values of the accelerations and velocities are searched by the host function **Computetimestep** for computing the step appropriate time.

Then the host invokes another kernel function named **Integrate**, that receives the value of the time step computed by the function **Computetimestep**, and from accelerations that are already in memory overall the device integrates the equations of motion for the new positions and velocities of each particle. Similarly to the case of function **ComputeAcceleration** is invoked N threads, one for each particle. The **Integrate** function should consider periodic boundary conditions as explained in last subsection, plus she is responsible for saving the position and velocity of the particle selected for analysis in each step, data that is passed for the memory of host to end the simulation.

When the function **Integrate** terminates the calculations, the host integrates the Friedmann equations to modify the scale factor and calculate the new age of the universe, the redshift z. If the value of z belongs to the set $z_k := \{z_0, z_1,\ cdots, z_{maxSnap}\}$, the iterative process is stopped to copy the positions and velocities of all the particles from the device to the host, to save them in to a data file, and then calculate the total energy of the box to assess the conservation of energy during the simulation. In this case, the gravitational potential energy is calculated in the device by using the **gEnergyT** that generates N threads, each responsible for calculating the total potential energy of a particle. The simulation ends when it reaches the maximum number of iterations is reached or it redshiftend z_f (user configurable). The data for analysis of selected particle must first be copied to the memory of host to save them in the file. Then release the global device memory to make it available to other applications.

3. How to run COLATUS: input file

aqui? The N-body simulator was developed in C, C + + and CUDA. He reads a configuration file and the initial conditions generated by N-GENIC. The configuration file contains information that indicates where the simulation will be performed (GPU or CPU), how many snapshots will saved, the folder and file name of the initial conditions, etc. The COLATUS performs a simulation of N-body cosmological scale with comoving coordinates and periodic boundary conditions in both CPU and GPU. It iteratively computes the interaction forces, integrating the equations of motion and the Friedmann equations (scale factor). Finally the results are saved in various files for later analysis and visualization. The general scheme of the simulator is shown on the Figure **??**.

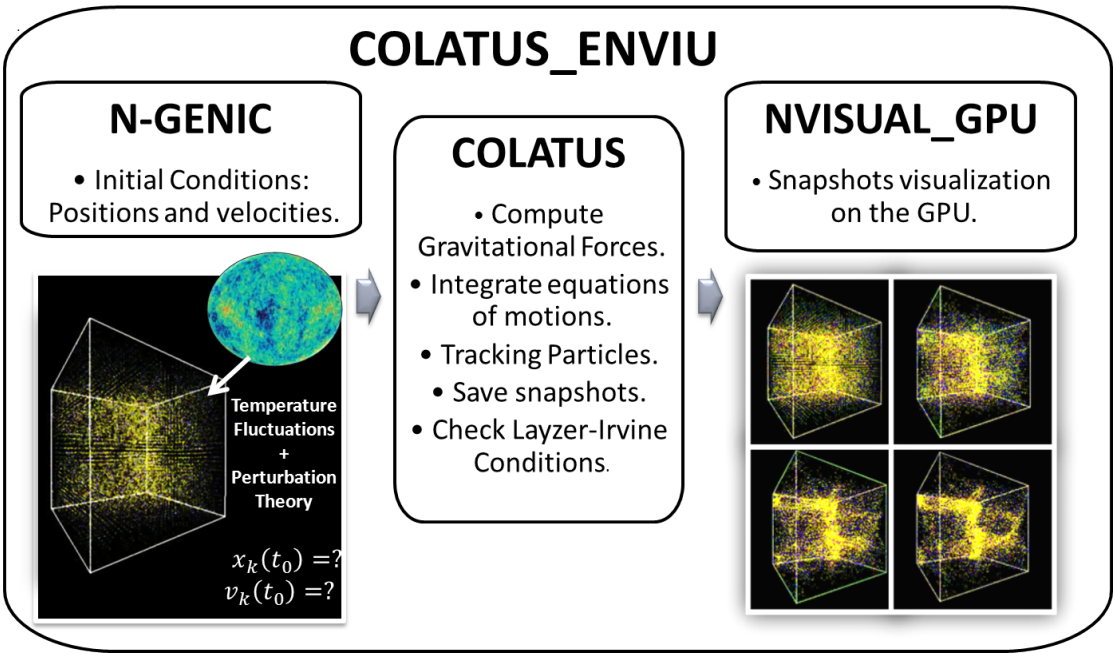

Figure 4: Cosmic Lagrangian Turbulence Simulator: COLATUS

3.1. Initial Conditions

The initial conditions generator was developed using C and MPI (*Message Passing Interface*) libraries and FFTW (*Fastest Fourier Transform* for calculating the fast Fourier transforms) and GSL (GNU *Scientific Library*, the numerical computation libraries). The N-GENIC code is available at `http://www.mpa-garching.mpg.de/gadget/`.

This application reads an input file where the cosmological parameters are defined, such as the size of the box, the initial redshift, the amount of baryonic and dark matter, the amount of dark energy, the Hubble parameter, the type of power spectrum and the measurement units. After setting the parameters of the simulations we run the inicial conditions generator by using the following command:

```
mpiexec -np 2  ./N-GenIC  ics.param
```

The output of the original code are several binary files, but new features that which adapt the output to the COLATUS read the initial conditions of a single file. We also generate a data description file by using the libxml2 library that contains the parameters with which the initial conditions were generated.

3.2. Simulations Parameters

In the COLATUS configuration file, are defined the simulation parameters that are the following:

- *Device to run* = [GPU—CPU]: defines the simulation will run on the CPU or GPU.
- Compute the energy during the simulation *Compute energy* = [Yes—No].
- *Save snapshots* = [Yes—No]: enable to save the box *snapshots* during the simulations and defines the maximum quantity of *snapshots* that it will save *Max Snapshots*.
- *Use Comoving Coordinates* = [Yes—No].
- *Use Variable Time step* = [Yes—No] enable the variable time-step, and *Eta* .
- *Softering epsilon*: The softening factor avoids singularity in the calculation of forces..
- The *Redshift final* or the maximum time steps (*Max time steps*).
- *Tracking ID*: selects the particle which information (position and velocity) at each time step will be extracted.
- *Binary output* = [Yes—No] : defines whether the *snapshots* will be saved in binary or ascii files, and the parameter *output* sets the base name of the output files.
- *IC file*: Contains the name of the file that describes the initial conditions *ics.xml*.

After setting the parameters of the simulations and the inicial conditions we run COLATUS by using the following command:

```
./COLATUS  simCFG.xml
```

4. Results of test run: output files

The results presented in this section are derived from simulations carried out using the model ΛCDM, ie based on the same parameters of the simulations of the Virgo Consortium [27] for later comparisons. However, other parameters can be simulated

by changing only the configuration file and generating the corresponding initial conditions. The parameters considered in the simulations are $N = 16^3, 24^3, 32^3, 48^3, 64^3, 80^3$. For all cases we used $\Omega_m = 0.3$, $\Omega_\Lambda = 0.7$, $\Omega_K = 0$, $\Omega_R = 0.0$, $h = 0.72$, $m = 34.1\ 10^{10} M_\odot/h$, $\epsilon = 20Kpc/h$,$\sigma_8 = 0.9$ and $z_{init} = 30$.

4.1. Simulations

As explained earlier, we use high-resolution simulations in small boxes, available at `http://www.mpagarching.mpg.de/Virgo/data_download.html`. Here we present a COLATUS simulation by using the initial conditions consistent with the *redshift* $z = 30$, and the variable time step, after 145 hours, 11 minutes and 8027 iterations we obtain the sequence of snapshots shown in Figure 5.

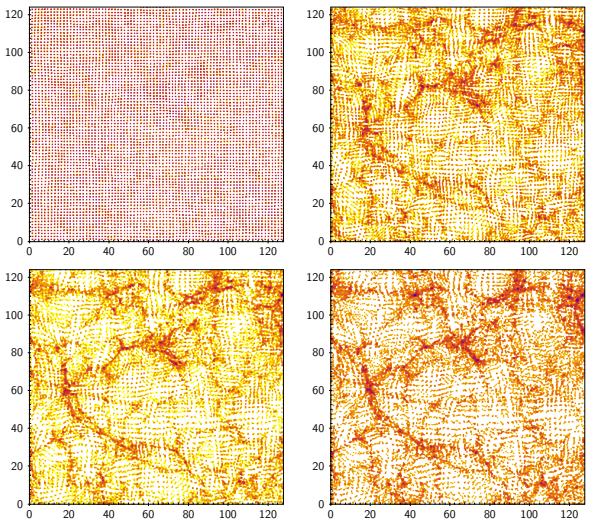

Figure 5: COLATUS simulations snapshots $z = \{30.0, 1.45, 0.53, 0.0\}$ (8x128x128 $(Mpc/h)^2$)

4.2. RMS differences between the GPU vs CPU

The simulations are performed using double precision variables COLATUS thus appear rounding errors and differences when comparing the results of CPU and GPU. As [28, 29] this occurs for several reasons: (i) the GPUs architecture are different from CPUs, eg, GPUs have multiply-add units and CPUs commonly not, (ii) the parallelization of the code can change the order of operations, (iii) the implementation of the IEEE 754 standard by the GPUs can not guarantee that the results are identical, even among GPUs. The mean quadratic differences between the corresponding positions and velocities of the particles show that the differences in outcome between CPU and GPU does not significantly change the distribution of matter, or the coordinates of a particle along the simulations.

Table 1 presents the results of the comparisons given the same initial conditions and parameters of the simulations.

4.3. Perfomance

Various simulations were performed in order to evaluate quantitatively the gains from the use of the GPUs. Due to the complexity of the algorithm used, the computation time increases proportionally to N^2. Experiments were done to compare the performance of the algorithm on GPUs, compared to CPU performance. In addition to checking the result of the calculations performed, these experiments aimed to study the behavior of processing time as compared to the amount of particles. According to the compute capability of the GPUs used GTX 580, GTX 680 and K20, was the adopted:

- threads numbers per block n_thread_block = 512 (GTX 580) and 1024 (for GTX 680 and K20).
- blocks numbers = $\lfloor N/\text{n\_thread\_block} \rfloor$ (where N is the total number of particles).

To evaluate the performance, we considered 100 iterations of the algorithm, the time was measured from the twentieth iteration to consider interactions where the data stream is already fully established.

Table 1: RMS differences between the GPU vs CPU($N = 16^3$)

Redshifs	10.0	5.0	1.0	0.0
Δx^{rms}	$5.59\ 10^{-7}$	$1.02\ 10^{-4}$	$7.87\ 10^{-3}$	$5.48\ 10^{-2}$
Δv_{rms}	$9.14\ 10^{-9}$	$1.05\ 10^{-4}$	$2.38\ 10^{-3}$	$5.39\ 10^{-2}$

Table 2: This table presents the simulator runtime in minutes and considering 100 time-steps

Particles	CPU Intel I7	GTX580	GTX680	K20
16^3	2.46	0.05	0.04	0.03
24^3	26.92	0.31	0.44	0.10
32^3	151.23	1.48	2.05	0.45
48^3	1722.57	15.29	24.18	5.83
64^3	9678.51	81.44	127.60	28.50
80^3	36920.58	326.89	486.14	108.04

The Figures 2 and 6 shown the times and speedup obtained respectively. The speed up (S) is obtained by dividing the time spent in the serial execution(only CPU) and the time spent in parallel execution, ie, $S = T_s/T_p$, where T_s is the sequential time and T_p is the parallel time.

On the Figures 2 and 6 we can observe that the performance gain is up to 118.7 times for the GTX 580 compared to serial implementation and up to 75.85 times for the GTX 680. However experiments on the GTX 680 (Kepler, newest) had lower performance than the GTX 580 (Fermi) because all calculations were performed in double precision and the Kepler architecture of the GTX 680 card does not implement this functionality. In this architecture, double precision is only emulated, which considerably reduces its performance compared to the performance of the Fermi architecture.

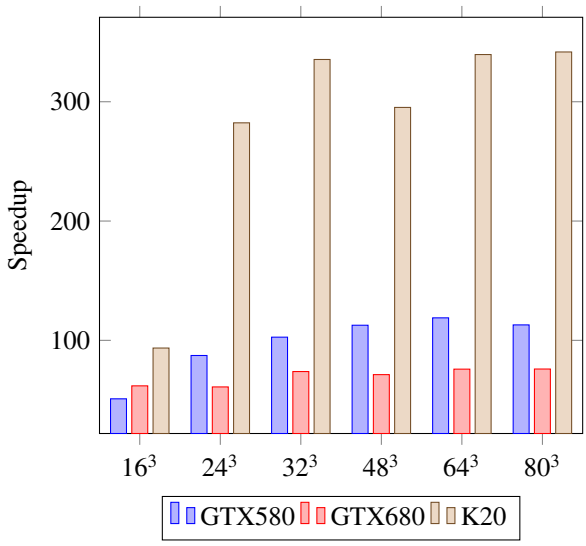

Figure 6: This Figure illustrates the speedup we consider different GPUs.

4.4. Virialized Halos

To identify the structures that formed, we used the FoF algorithm implemented by [30]. This application also calculates the properties of the halos identified, that is, the mass, position, velocity and average velocity dispersion halos. The total mass M_j of each halo is determined by summing the individual mass m_i of each particle contained in the halo, ie $M_j = \sum_{i=0}^{K_j} m_i$ where K_j is the number of particles contained in each halo. The center of mass of the halo is obtained from the average of the coordinates of the particles of the halo. For the mean velocity $\overline{v}$, add up the speeds of all the particles and divided by the total number of particles in the halo, as $\overline{v}_j = \frac{1}{K_j} \sum_{i=0}^{K_j} v_i$. Finally, the velocity dispersion is obtained by the standard deviation of the particle velocity for each halo. The standard deviation is given by $S = \sqrt{\frac{1}{K_j - 1} \sum_{i=0}^{K_j} (v_i - \overline{v}_j)^2}$, where $\overline{v_j}$ is the average speed.

To identify halos of galaxy clusters, usually uses the value of $b = 0.2$ (Mpc/h) and a minimum number of particles [31]. To identify the halos of galaxies, we used the FoF algorithm with $b = 0.1$ with the criterion of boundedness to select only virializaded halos, ie, those whose velocity dispersion is proportional to the square root of the mass (number of particles of each halo) [25].

The Figure 7 shows the velocity dispersion of the galaxy halos as a function of halo particle number. We consider only the virialized halos, ie those whose velocities are between the orange and yellow curves. These curves were obtained by making a least squares fit to a function of type $\overline{v} = a\,K + b$. With the resolution of particle mass ($1.4\ 10^{10} M\odot$) used in the simulations, a halo of dark matter in a galaxy like the Milky Way ($M \approx 10^{12} h^{-1} M_\odot$), for example, is obtained with approximately 100 particles. These results are consistent with those presented in [32], therefore we can say that COLATUS could simulate the gravitational collapse structures generating virializadas similarly those generated by simulations of the VIRGO consortium that used the GADGET [27, 11] .

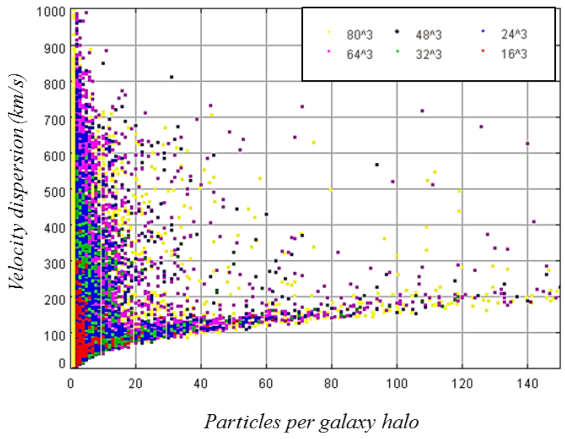

Figure 7: Velocity dispersion of galaxies halos identified in our simulations

4.5. Tracking Particles

The present status of the simulator [1] allows to extract information (position and velocity) of a group of particles in each time step of the simulation as shown in Figure **??**. This approach aim to identify patterns and similar to turbulence from a statistical analysis of the dynamics of particle processes.

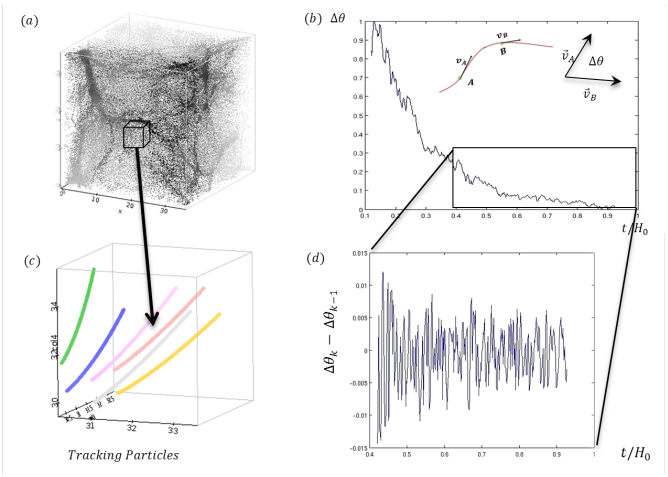

Figure 8: (a) Selection of the analyzed particles, (b) Angular variation of particle velocities respect to the time., (c) Evolution of the analyzed particle positions, (d) Diferences time series obtained from the angular variation.

4.5.1. Velocities angular variation

The variability pattern seen in the example of Figure **??**,(b,d), highlights the importance of including the Lagrangian tracer for fine study of instabilities present in the dynamics of each particle along the formation of filaments. In particular, this result opens a new perspective for the study of processes of type Chaotic advection (or Lagrangian turbulence) predicted from the analysis of data generated with GADGET2 [26]. This new analytical proposal may provide a new method for validating

[1] The first prototype is called COLATUS ENVIU 1.0 developed in the institutional Environment Inpe-Uff.

models in cosmology. Mainly for comparative analysis between models and observational data of high resolution planned for the coming years.

5. Summary and outlook

Our new implementation of the COLATUS in GPU/CUDA is an important step towards the study of cosmological large-scale structure formation by tracking the nonlinear path of the single particles during the gravitational collapse(*melhorar*).

In its current version, COLATUS ENVIU 1.0, the program serves as a prototype for pretesting new experiments in their minimized configuration before ordering such experiments to the major cosmology frameworks (eg Millenium and Ilustris) whose demand for sharing is very high.

Acknowledgments

We thank CAPES, FAPESP(Proc.2007/54133-0) and CNPQ for the finaltial support.

References

[1] E. Bertschinger, Annual Review Astronomy Astrophysics 36 (1998) 599–654.
[2] J. S. Bagla, Current Science 88 (2008) 1088.
[3] R. S. Somerville, R. Dav, Annual Review of Astronomy and Astrophysics 53 (2015) 67.
[4] M. VOGELSBERGER, S. GENEL, V. SPRINGEL, P. TORREY, D. SIL-JACKI, D. XU, G. SNYDER, S. BIRD, D. NELSON, L. HERNQUIST, Nature 509 (2014) 177–182.
[5] P. J. E. Peebles, Astrophysical Journal 75 (1970) 13.
[6] W. H. Press, P. Schechter, Astrophysical Journal 187 (1974) 425–438.
[7] S. D. M. White, Monthly Notices of the Royal Astronomical Society 177 (1976) 717–733.
[8] A. Appel, SIAM J. Sci. Stat. Comput. 6 (1985) 85.
[9] J. Barnes, P. Hut, Nature 324 (1986) 446–449.
[10] V. Springel, N. Yoshida, S. D. M. White, New Astronomy 6 (2001) 79–117.
[11] V. Springel, Mon. Not. R. Astron. Soc. 3 (2005) 31.
[12] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik1, O. O. Storaasli, Scientific Programming 18 (2010) 1–33.
[13] J. Bédorf, E. Gaburov, M. S. Fujii, K. Nitadori, T. Ishiyama, S. P. Zwart, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 54–65.
[14] L. Nyland, M. Harris, J. Prins, in: H. Nguyen (Ed.), GPU Gems 3, Addison Wesley Professional, 2007.
[15] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, M. Taiji, in: Proceedings of the Conference on High Performance Computing Networking,Storage and Analysis, SC 09, ACM, New York, NY, USA, 2009, pp. 62:1–12.
[16] E. Gaburov, S. Harfst, S. P. Zwart, Instrumentation and Methods for Astrophysics (2009).
[17] R. W. Hockney, J. W. Eastwood, Computer Simulation Using Particules, Adam Hilger, Bistol and New York, New York, USA, 1988.
[18] M. W. S. D. M. F. C. S. Efstathiou, G.; Davis, Astrophysical Journal Supplement Series 57 (1985) 241–260.
[19] P. Bodnheimer, G. P. Laughlin, H. W. Yorke, Numerical Methods in Astrophysics, Taylor and Francis, New York, USA, 2007.
[20] J. M. B. C. M. Zemp, Marcel; Stadel, Mon. Not. R. Astron. Soc. 376 (2007) 273–286.
[21] F. R. Hernquist, L.and Bouchet, Y. Suto, Astrophysical Journal Supplement Series 75 (1991) 231–240.
[22] A. Y. Toukmaji, J. A. Jr Board, Computer Physics Communication 95 (1996) 73–92.
[23] G. Hinshaw, D. Larson, E. Komatsu, D. N. Spergel, J. D. C. L. Bennett, M. R. Nolta, M. Halpern, R. S. Hill, N. Odegard, N. J. L. Page, K. M. Smithand J. L. Weilandand B. Gold, A. Kogut, M. Limon, S. S. Meyer, G. S. Tucker, E. Wollack, E. L. Wright, The Astrophysical Journal Supplement Series, Submited (2012) 31.
[24] Y. B. Zel'dovich, Astronomy and Astrophysics 5 (1970) 84–89.
[25] C. A. Caretta, R. R. Rosa, H. F. C. Velho, F. M. Ramos, M. Makler., Astron. Astrophys 48 (2008).
[26] R. R. Rosa, F. M. Ramos, C. A. Caretta, H. F. C. Velho, Computer Physics Communications 4 (2009) 180:621–624.
[27] A. D. S. W. G. Kauffmann, J.M. Colberg, Mon. Not. R. Astron. Soc. 303 (1999) 188–206.
[28] J. Delgado, J. Gazolla, E. Clua, S. M. Sadjadi 2 (2011) 3–11.
[29] N. Whitehead, A. FitFlorea (2011) 7.
[30] J. P. Huchra, M. J. Geller, Astrophysical Journal 257 (1982) 423–437.
[31] C. Lacey, S. Cole, Monthly Notices of the Royal Astronomical Society, 271 (1994) 676.
[32] R. S. d. R. Ruiz, Turbulência em cosmologia: análise de dados simulados e observacionais usando computação de alto desempenho, Ph.D. thesis, Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011.
[33] S. J. Aarseth, in: J. U. Brackbill, B. I. Cohen (Eds.), Multiple time scales, p. 377 - 418, pp. 377–418.
[34] S. J. Aarseth, Gravitation N-Body Simulation, Cambridge Unversity Press, New York, USA, 2003.
[35] H. Mo, F. vandenBosch, S. White, Galaxy Formation and Struture, Cambridge University Press, New York, USA, 2010.
[36] A. D. Green, Cosmological Application of Multigrid Methods, Phd thesis, University of Oxford, St. Peterś College, 2000.

6. Appendix

6.1. Equations of motion

If we consider N particles, each with mass m, distributed in a cosmological box (we assume a cube with sides of lenght L), the equations of motion [33] result in a system of equations differential of $6N$ variables, where the state of each particle is defined by the position vector $\mathbf{r_j}$ and velocity $\mathbf{v_j}$ as follows:

$$\frac{d\mathbf{r_j}}{dt} = \mathbf{v_j}, \qquad \frac{\mathbf{dv_j}}{\mathbf{dt}} = -\nabla\mathbf{\Phi_j}, \tag{9}$$

where

$$\nabla\Phi_j = \frac{F_j}{m_j} = G\sum_{k\neq j}^{N}\frac{m_k(\mathbf{r_k}-\mathbf{r_j})}{|\ \mathbf{r_k}-\mathbf{r_j}\ |^3}, \tag{10}$$

here the evaluation of force (10) and solving the equation of motion (9) in a comoving coordinates are two key components of these simulations [2].

Smoothing factor

The N-body simulations " cosmological " consider a very large number of particles ($N > 10^5$) [1] acolisional way of interacting. A smoothing factor ϵ usually associated with the *half- mass radius of a standard galaxy* [34] is introduced to avoid the singularity when the distance between two particles approaches zero and ensure accurate integration methods.

$$\nabla\Phi_j = \frac{F_j}{m} = G\sum_{k\neq j}^{N}\frac{m(\mathbf{r_k}-\mathbf{r_j})}{\left(|\ \mathbf{r_k}-\mathbf{r_j}\ |^2+\epsilon^2\right)^{3/2}} \tag{11}$$

Scale Fator

The variation of the scale factor is modeled by the Friedmann equations:

$$\frac{\dot{a}}{a} = H_0 \sqrt{\Omega_R\, a^{-4} + \Omega_M\, a^{-3} + \Omega_K\, a^{-2} + \Omega_\Lambda} \quad (12)$$

$$\frac{\ddot{a}\, a}{\dot{a}} = \Omega_M/2 - \Omega_\Lambda - \Omega_R \quad (13)$$

where Ω_R is the density of radiation, Ω_M is the density of matter (more baryonic dark) ,Ω_K is the density of the spatial curvature and Ω_Λ is the cosmological constant or vacuum energy density [35].

6.2. *Details of the Integration scheme*

We consider the approach introduced by [17], where the equations are:

$$\frac{d\mathbf{x_j}}{dp} = \frac{\mathbf{u'_j}}{a}, \quad \frac{d\mathbf{u'_j}}{dp} + 2\,A(p)\mathbf{u'_j} = -\mathbf{B(p)}\nabla_\mathbf{x}\phi(\mathbf{x_j}), \quad (14)$$

$$\nabla_x\phi(\mathbf{x_j}) = G\sum_{k\neq j}\frac{m(\mathbf{x_j}-\mathbf{x_k})}{\left(\mid \mathbf{x_j}-\mathbf{x_k}\mid^2 + (\epsilon_\mathbf{x})^2\right)^{3/2}}. \quad (15)$$

where $\epsilon_x = \epsilon/a, u'_j = \frac{d\mathbf{x_j}}{dp}$, $A(p) = \frac{\alpha + \frac{\ddot{a}a}{a^2}}{2\alpha\, p}$ e $B(p) = \frac{1}{(\alpha\, p\dot{a})^2}$.

Them we change the derived position and velocity differences in the respective equations to numerically integrating the equations of motion of each particle by using the integration scheme called Leap-Frog *j*, ie:

$$\mathbf{u_j^{n+1}} = \mathbf{u^n}\frac{\mathbf{1}-\mathbf{A(p_n)\Delta p}}{\mathbf{1}+\mathbf{A(p_n)\Delta p}} + \mathbf{B(p_n)}\frac{\nabla_\mathbf{x}\phi(\mathbf{x_j^{n+1/2}})\mathbf{\Delta p}}{(\mathbf{1}+\mathbf{A(p_n)\Delta p}} \quad (16)$$

$$\mathbf{x^{n+3/2}} = \mathbf{x^{n+1/2}} + \mathbf{\Delta p}\,\mathbf{u^{n+1}}. \quad (17)$$

Note 1: the first and last step are calculated by the following equations:

$$\mathbf{u_j^{1/4}} = \mathbf{u^0}\{\mathbf{1} - \mathbf{2}\,\mathbf{A(p_0)}\frac{\mathbf{\Delta p}}{\mathbf{4}}\} + \mathbf{B(p_0)}\nabla_\mathbf{x}\phi(\mathbf{x_j^0})\frac{\mathbf{\Delta p}}{\mathbf{4}} \quad (18)$$

$$\mathbf{x^{1/2}} = \mathbf{x^0} + \frac{\mathbf{\Delta p}}{\mathbf{2}}.\,\mathbf{u^{1/4}} \quad (19)$$

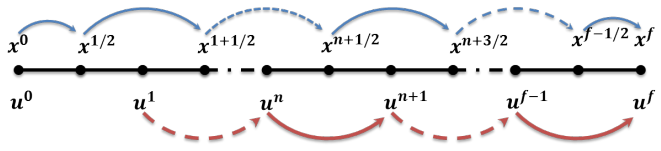

Figure 9: Leap Frog scheme

The Leap-Frog method is consistent, stable and truncation errors are $O(\Delta p^3)$. To implement methods of higher order would need to calculate and save the additional accelerations, which would increase considerably the memory requirements. Apart from this the Leap-Frog method allows energy is conserved during the simulation [36, 34].