sid.inpe.br/mtc-m19/2013/02.08.17.58-TDI

UM NOVO SIMULADOR DE N-CORPOS PARA COSMOLOGIA COMPUTACIONAL UTILIZANDO GPUS

Diego Herbin Stalder Díaz

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Reinaldo Roberto Rosa, e Renata Sampaio da Rocha Ruiz, aprovada em 22 de fevereiro de 2013.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3DGNTNH>

INPE
São José dos Campos
2013

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

sid.inpe.br/mtc-m19/2013/02.08.17.58-TDI

UM NOVO SIMULADOR DE N-CORPOS PARA COSMOLOGIA COMPUTACIONAL UTILIZANDO GPUS

Diego Herbin Stalder Díaz

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Reinaldo Roberto Rosa, e Renata Sampaio da Rocha Ruiz, aprovada em 22 de fevereiro de 2013.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3DGNTNH>

INPE
São José dos Campos
2013

Dados Internacionais de Catalogação na Publicação (CIP)

Stalder Díaz, Diego Herbin.
St16n Um novo simulador de n-corpos para cosmologia computacional utilizando GPUs / Diego Herbin Stalder Díaz. – São José dos Campos : INPE, 2013.
xxvi + 106 p. ; (sid.inpe.br/mtc-m19/2013/02.08.17.58-TDI)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2013.
Orientadores : Drs. Reinaldo Roberto Rosa, e Renata Sampaio da Rocha Ruiz.

1. cosmologia computacional 2. GPU 3. ncorpos 4. formação de estruturas em grande escala . I.Título.

CDU 004:524.8

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em

Computação Aplicada

Dr. Solon Venâncio de Carvalho

Presidente / INPE / SJCampos - SP

Dr. Reinaldo Roberto Rosa

Orientador(a) / INPE / SJCampos - SP

Dra. Renata Sampaio da Rocha Ruiz

Orientador(a) / INPE / São José dos Campos - SP

Dr. Haroldo Fraga de Campos Velho

Membro da Banca / INPE / São José dos Campos - SP

Dr. Stephan Stephany

Membro da Banca / INPE / SJCampos - SP

Dr. Hugo Vicente Capelato

Membro da Banca / INPE / SJCampos - SP

Dr. Esteban Walter Gonzalez Clua

Convidado(a) / UFF / Rio de Janeiro - RJ

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

Aluno (a): *Diego Herbin Stalder Diaz*

São José dos Campos, 22 de Fevereiro de 2013

"El cosmos es todo lo que es, todo lo que fue y todo lo que será. Nuestras más ligeras contemplaciones del cosmos nos hacen estremecer: Sentimos como un cosquilleo nos llena los nervios, una voz muda, una ligera sensación como de un recuerdo lejano o como si cayéramos desde gran altura. Sabemos que nos aproximamos al más grande de los misterios.".

Carl Sagan
em *"Cosmos"*, 1980

A meus pais ***Nélida*** *e* ***Herbin****,*
e ao amor da minha vida ***Mariam***

AGRADECIMENTOS

Agradeço em especial, aos meus orientadores Reinaldo Roberto Rosa e Renata Sampaio da Rocha Ruiz, por ter acreditado em mim, pela sua competência, atenção e disponibilidade em sempre ajudar.

Aos professores do LAC-INPE, que contribuíram para minha formação acadêmica e amadurecimento pessoal. Em especial aos professores Haroldo, Fernando, Stephan e Solon que acompanharam com incentivo o desenvolvimento deste projeto.

Ao professor Christian Schaerer pela motivação e apoio constante.

Agradeço aos colegas que participaram, de alguma forma, no estudo e desenvolvimento das versões do algoritmo que resultou no simulador $COLATUS\_ENVIU$: Amarísio da Silva Araújo, Vitor Conrado F. Gomes, José Ricardo da Silva Júnior, Daniel Madeira e Prof. Esteban Clua.

Aos meus colegas de turma, pelos agradáveis momentos de convivência, apoio, incentivo e troca de ideias e experiências.

Aos amigos Saymon, Luis e Rudinei, pela paciência em ouvir, pela amizade e pela companhia.

À CAPES pela bolsa.

E finalmente a minha família e namorada pelo inestimável apoio no âmbito pessoal.

RESUMO

Nas últimas duas décadas a pesquisa sobre a origem e a evolução do Universo atingiu status de grande ciência, principalmente devido aos importantes resultados experimentais obtidos tanto a partir de grandes missões espaciais (satélite COBE em 1992), como também a partir de avançados experimentos numéricos desenvolvidos no âmbito da supercomputação (Consórcio Virgo em 1994). A cada dia, dezenas de artigos são publicados com novos resultados observacionais e teóricos lançando novos desafios científicos e tecnológicos. Um dos destaques é a demanda por novos simuladores de interação gravitacional em grandes escalas baseados em tecnologias com maior eficiência computacional. Nesse contexto, a presente pesquisa de mestrado apresenta, como principal resultado, um novo simulador de N-corpos, o $COLATUS\_ENVIU$, desenvolvido por iniciativa nacional entre o LAC-INPE e o IC-UFF para aplicações em cosmologia computacional. O estudo envolve a aplicação da tecnologia GPU/-CUDA na simulação da formação de estruturas gravitacionais em grandes escalas ($16Mpc/h < L < 128Mpc/h$), incorporando características que permitem testar propostas teóricas alternativas para o modelo cosmológico ΛCDM. Simulações do volume de Hubble compreendendo um número de elementos da ordem de 10^6 são apresentadas para as mesmas faixas de *redshift*, condições de contorno e condições iniciais consideradas em simulações que não utilizam a tecnologia GPU. Estudos comparativos, envolvendo a função de correlação de dois pontos e a dispersão de velocidades também calculadas sobre resultados obtidos a partir do Consórcio VIRGO, comprovam o desempenho do simulador $COLATUS\_ENVIU$ para aplicações em cosmologia computacional.

A NEW SIMULATOR FOR COMPUTATIONAL COSMOLOGY USING THE GPUS

ABSTRACT

In the last two decades the study of the origin and evolution of the Universe achieved the status of big science mainly due to significant experimental results not only from major space missions (COBE satellite in 1992), but also from advanced numerical experiments performed by supercomputing (Virgo Consortium in 1994). Every day, dozens of articles are published with new observational and theoretical results bringing new scientific and technological challenges. One of the highlights is the demand for new simulators for gravitational interaction on large scales based on technologies with greater computational efficiency. In this context, this master's research presents, as main result, a new N-body simulation environment, named $COLATUS\_ENVIU$, developed from a brazilian initiative between LAC-INPE and IC-UFF, for applications in computational cosmology. The study involves the application of GPU/CUDA technology for simulation of gravitational large-scale structure formation ($8Mpc/h < L < 128Mpc/h$), incorporating new features that allow one to test alternative theoretical proposals to the standard cosmological model ΛCDM. Simulations of the Hubble volume 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 simulations that do not use the GPU technology. Comparative studies involving the two-point correlation function and the velocity dispersion, calculated also on results obtained from the Virgo consortium, prove the performance of the simulator $COLATUS\_ENVIU$ for applications in computational cosmology.

LISTA DE FIGURAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

COLATUS	–	Cosmic Lagrangian Turbulence Simulator
LAC	–	Laboratório Associado de Computação e Matemática Aplicada
GPU	–	Unidade de Processamento Gráfico
CPU	–	Unidade de processamento
CUDA	–	Compute Unified Device Architecture
APM	–	Adaptative Particule Mesh
ARM	–	Adaptative Mesh Refinement
TreePM	–	Tree Particule Mesh
TreePP	–	Tree Particule Particule
FoF	–	Algoritmo Amigos dos Amigos(*Friends o Friends*)
2PCF	–	Função correlação de dois pontos
SM	–	Streaming Multiprocessor
SP	–	Streaming Processor
TRG	–	Teoria Geral da Relatividade
DSPP	–	Soma direta - Partícula Partícula
DSP	–	Digital Signal Processor
FPGA	–	Field Programamble Gate Array
SDSS	–	Sloan Digital Sky Survey
FIRAS	–	Far InfraRed Absolute Spectrometer
COBE	–	COsmic Background Explorer
UFF	–	Universidade Federal Fluminense
SF	–	Singularidade de Friedmann
SEP	–	Strong Equivalence Principle

LISTA DE SÍMBOLOS

z	–	desvio para o vermelho ou *redshift*
Ω_M	–	parâmetro da densidade densidade de matéria (escura mais bariônica)
Ω_R	–	parâmetro da densidade da radiação
Ω_Λ	–	constante cosmológica ou parâmetro de densidade de energia do vácuo
Ω_K	–	parâmetro da densidade da curvatura espacial
σ_8	–	amplitude do espetro de potência linear na escala de 8Mpc/h
P	–	pressão
r	–	vetor de posição
x	–	vetor de posição em coordenada comóveis
v	–	velocidade física
u	–	velocidade peculiar
$H(t)$	–	constante de Hubble
H_0	–	constante de Hubble atual
a	–	fator de escala
ρ	–	densidade de massa
ρ_c	–	densidade de massa crítica
m, M	–	massa
G	–	constante gravitacional
F	–	força
ϵ	–	fator de suavizado
η	–	parâmetro de ajuste do passo de tempo
Λ	–	constante cosmológica
λ_o	–	comprimento de onda emitido
λ_e	–	comprimento de onda observado
λ_e	–	comprimento de onda observado
ϵ	–	fator de suavizado
ϕ	–	potencial
U	–	Energia Potencial total
T	–	Energia Cinética total
N	–	Número de partículas
L	–	Tamanho da caixa
R	–	Raio de percolação do algoritmo FoF

SUMÁRIO

Pág.

1 INTRODUÇÃO

A Cosmologia contemporânea é um arranjo complexo de teorias minuciosamente elaboradas baseadas em observações e experimentos numéricos que compreendem alta tecnologia, além de hipóteses que em geral são formuladas no escopo dos maiores paradigmas da física: a teoria da relatividade geral e a teoria quântica. A cosmologia, como ciência moderna bem estabelecida, está determinada a encontrar respostas sobre a origem e evolução do universo físico que sejam consistentes com dados observacionais de alta resolução provenientes desde o universo em seus estágios iniciais (ex.: radiação cósmica de fundo) como também em seus estágios mais evoluídos (distribuição de galáxias e de seus aglomerados).

Além da realidade imposta pelos dados observacionais da astronomia moderna, bem como pelas teorias consolidadas da física, os mais importantes fundamentos da cosmologia contemporânea têm sido elaborados também a partir de experimentos numéricos utilizando supercomputação. De fato, a computação aplicada à cosmologia tem sido importante em quase todos os seus segmentos: (i) nos projetos e testes de qualidade dos instrumentos observacionais de última geração, (ii) no processamento, visualização e análise de dados gerados com alta resolução nos domínios temporal, espacial e espectral; (iii) na simulação de mapas da radiação cósmica de fundo; e (iv) nas simulações da formação de estruturas gravitacionais em grandes escalas.

No escopo desta dissertação, a "Cosmologia Computacional" tratará apenas da modelagem da formação da estruturas no Universo por meio de simulações numéricas. Estas simulações podem ser consideradas como "experimentos" capazes de verificar teorias da cosmologia moderna onde se destaca o modelo padrão denominado ΛCDM (ver Apêndice A).

Ao longo dos últimos vinte anos, grandes progressos foram feitos no desenvolvimento de códigos de computador que são capazes de simular a evolução imposta pelo modelo ΛCDM, os quais em suas versões mais elaboradas podem incluir a interação gravitacional, hidrodinâmica dos gases e processos radiativos os quais permitem considerar até mesmo a formação de estrelas (DEKEL A.AND OSTRIKER, 1999). Neste contexto, este projeto de mestrado foi elaborado a partir de uma colaboração entre pesquisadores do Laboratório Associado de Computação e Matemática Aplicada (LAC) do INPE e do Instituto de Computação da Universidade Federal Fluminense (UFF). A proposta de colaboração salientou o estudo e implementação de um simulador de N-corpos, baseado na tecnologia GPU, para a linha de pesquisa em computação aplicada à cosmologia do LAC. Esta linha, ativa no INPE desde 2002,

iniciou-se a partir da simulação e análise de dados da radiação cósmica de fundo, progredindo para a análise de dados de aglomerados de galáxias, estudo de algoritmos para simulação da formação de estruturas em grandes escalas, e finalmente na concepção de modelos alternativos para interpretar a natureza da matéria e energia "escuras" (ver Apêndice B).

1.1 Contexto e Motivação

A primeira geração de simuladores de N-corpos para astrofísica e cosmologia foi desenvolvida durante a década de 70. Esses simuladores em geral envolvem um código mestre para simulação de N-corpos em interação através do campo gravitacional. A principal característica da primeira geração, denominada DSPP, é o uso da soma direta (*direct sumation*) no esquema de interação conhecido como PP (*Particle-Particle*) cuja complexidade é $\approx O(N^2)$. Dentro desse paradigma DSPP (*Direct Sumation Particle-Particle*), foram desenvolvidos quase uma dezena de simuladores para a cosmologia computacional (HOLMBERG, 1941; PRESS; SCHECHTER, 1974; WHITE, 1976; AARSETH, 1985). A segunda geração de simuladores N-corpos procurou introduzir esquemas de interação capazes de diminuir a complexidade computacional dos algoritmos para $\approx O(NlogN)$. O método mais comumente utilizado para interação das partículas é conhecido como TreePM ("Tree Particle-Mesh"), onde grupos de partículas são definidos a partir de um esquema de árvore. Esse método foi criado no início da década de 80 por Andrew Appel em sua dissertação de mestrado desenvolvida em Princeton (APPEL, 1985), sendo que o algoritmo de árvore mais aplicado em cosmologia computacional foi introduzido por Barnes e Hut em 1986 (BARNES; HUT, 1986). A partir do início da década de 90, tendo como base o novo paradigma, vários grupos de pesquisadores em diferentes instituições no cenário internacional, deram início a grandes projetos que visavam a elaboração de simuladores de segunda geração para a cosmologia computacional. Após duas décadas de pesquisa e desenvolvimento de vários trabalhos continuados nessa área, é possível identificar mais de uma dezena de simuladores numéricos de segunda geração dedicados à cosmologia computacional. Podemos considerar que uma terceira geração começou a surgir no início da última década com o advento da tecnologia das placas gráficas programáveis para a execução de cálculos numéricos simples. A tecnologia conhecida como GPU/CUDA (Graphic Processing Unit/ Compute Unified Device Architecture), por exemplo, proporcionou aos cientistas uma nova concepção de computação científica incluindo com destaque a partir de 2006, a simulação N-corpos (ELSEN et al., 2006; NYLAND et al., 2007) inclusive direcionados para aplicações em astrofísica e cosmologia (NITADORI; AARSETH, 2012; STALDER et al., 2012).

O primeiro código da segunda geração para a cosmologia computacional, a ser categoricamente disponibilizado e amplamente utilizado pela comunidade científica internacional foi o GADGET[1], desenvolvido pelos pesquisadores Volker Springel, Naoki Yoshida e Simon White do Max Planck Institut für Astronomie (MPIA) em Garching na Alemanha (SPRINGEL et al., 2001). Esse simulador, baseado integralmente em arquitetura massivamente paralela de CPUs (Central Processing Units), além de considerar um esquema para interação das partículas TreePM incorporava um módulo SPH (Smoothed Particle Hydro-dynamics)(LUCY, 1977; GINGOLD; MONAGHAN, 1977) para considerar o gás intergaláctico. Outros simuladores foram desenvolvidos considerando novos métodos e algoritmos como por exemplo: APM (*Adptative Particle Mesh*), AMR (*Adaptative Mesh Refinement*) e ART (*Adaptative Refinement Tree*) ou P3M (BERTSCHINGER, 1998). Em geral, os simuladores de N-corpos para cosmologia computacional passam por três estágios durante a sua concepção: a primeira versão simula apenas a interação gravitacional entre N galáxias ($N > 10^4$), em versões posteriores são incorporados ingredientes para incluir o gás intergaláctico e posteriormente o plasma para o estudo de formação de estrelas durante o colapso gravitacional em escalas cosmológicas (MO et al., 2010). A inclusão do gás e dos processos radioativos nem sempre são necessários para um estudo da formação de estruturas em grandes escalas. O refinamento físico do simulador depende do propósito fenomenológico para o qual uma determinada teoria e/ou conjunto de dados observacionais estão associados ao problema. Nesse sentido, é necessário estudar e identificar que tipos de abordagens para a terceira geração podem ser úteis para a cosmologia moderna, principalmente quando consideramos propostas alternativas para interpretar a natureza da matéria e energia escuras (modelo ΛCDM) bem como os dados de muito alta resolução[2] que serão obtidos a partir da nova geração de instrumentos que entrarão em operação nesta e nas próximas décadas.

Alguns ambientes de simulação para cosmologia computacional, de segunda e terceira gerações, são identificados na Figura 1.1. Nesta tabela está incluído o simulador $COLATUS\_ENVIUv1$ estudado e concebido majoritariamente a partir deste projeto de mestrado.

As principais motivações para a elaboração do $COLATUS\_ENVIUv1$ foram as seguintes:

1) Inexistência de um simulador para cosmologia computacional, baseado em

[1] GAlaxies with Dark matter and Gas interact.

[2] *Spectrographs* 1000 *fibers of* $2''$ *entrance aperture* and *redshifts* $z > 2.2$, (`http://www.sdss3.og/dr9`)

computação híbrida incluindo GPU/CUDA, no cenário nacional;

2) Identificação no Brasil de um grupo consolidado de computação híbrida especializado na tecnologia GPU/CUDA, no Instituto de Computação da UFF, que propôs uma parceria com o LAC-INPE para aplicações em ciências espaciais;

3) Desde o início de 2002 a linha de pesquisa em cosmologia computacional do LAC-INPE trabalha com análise de dados e, mais recentemente, com modelos alternativos relacionados ao cenário proposto pelo modelo ΛCDM (Apêndice B). Com a concepção de um simulador próprio, que proporcionará ao grupo um completo domínio e entendimento do ambiente de simulação, abordagens alternativas poderão ser estudadas inserindo o grupo de pesquisa em patamares INPE-UFF concorrentes com grupos internacionais.

Simulador	Grupo Principal	Inicio Operacional	Processamento	Algoritmo	Gás	Formação de estrelas	Traçador Lagrangeano
HYDRA	Uwest.Onta.	1996	CPU	AP3M	X		
ART	UNewMexico	1997	CPU	ART	X		
GADGET1	MaxPlanckIA	2001	CPU	TreePM	X		
MLAPM	Potsdam	2001	CPU	ART	X		
GOTPM	Utoronto	2003	CPU	GOTreePM			
PKDGRAV2	UMcMaster	2003	CPU	TreePP	X	X	
AMIGA	U.Madrid	2004	CPU	ART	X		
GADGET2	MaxPlanckIA	2005	CPU	TreePM	X	X	
CUNBODY	CAL-JP	2007	CPU/GPU	DSPP-TreePP			
ENZO	UCalif._LCA	2010	CPU/GPU	ART	X		
DEUSS	Obs.Paris	2010	CPU	ART	X	X	
CHANGA2	UWashington	2010	CPU/GPU	TreePP	X	X	
BOLSHOI	Ucalif.-HIPACC	2010	CPU	ART	X		
NBODY6	U.Cambrige	2012	CPU/GPU	DSPP			
COLATUS_ENVIU	INPE/UFF	2012	CPU/GPU	DSPP			X

Figura 1.1 - Uma relação em ordem cronológica do simuladores N-Corpos de segunda e terceira gerações

1.2 Objetivos

Os objetivos desta dissertação são apresentados abaixo:

Objetivo Geral

Conceber um simulador híbrido utilizando GPU/CUDA que em sua primeira versão gere resultados onde o colapso gravitacional no contexto cosmológico seja observado. Isto é, que a formação de filamentos seja obtida ao longo de um colapso gravitacional que se desenvolva em faixas válidas para o *redshift* cosmológico e densidade de galáxias e que seja consistente com o modelo ΛCDM.

Objetivos Específicos

a) Estudar a simulação de N-corpos no ambiente GPU/CUDA e adaptá-la ao contexto da cosmologia computacional de interesse, com destaque para a pesquisa sobre condições iniciais, métodos para calcular as forças de interação gravitacional considerando $N > 10^5$ partículas;

b) Apresentar resultados provenientes de experimentos utilizando o simulador $COLATUS\_ENVIU1$ e validá-los por meio da análise comparada, da correlação de dois pontos e da dispersão de velocidades, com resultados obtidos via GADGET2 (Projeto Virgo) (SPRINGEL, 2005);

c) Explorar o esquema DSPP para inserção de ingredientes que sejam importantes nas abordagens alternativas estudadas pelo grupo de cosmologia computacional do LAC-INPE. Um exemplo: considerar a implementação do "Traçador Lagrangeano" de partículas, ausente nos simuladores da segunda e terceira geração. Esse traçador permite salvar a posição e velocidade instantâneas de uma ou um grupo de partículas ao longo da simulação. Cabe ressaltar que, embora o módulo SPH(*Smoothed Particle Hydrodynamics*) incorporado aos algoritmos N-corpos (ex. GADGET) permite determinar a densidade local de massa, o esquema TreePM inviabiliza o acompanhamento de uma ou um grupo de partículas (SPRINGEL et al., 2001; BINNEY J.; TREMAINE, 2003).

1.3 Estrutura da Dissertação

O Capítulo 2 apresenta os fundamentos de cosmologia computacional que são necessários para o estudo e desenvolvimento do simulador de N-corpos proposto nesta

dissertação e apresentado no Capítulo 3.

As primeiras simulações, resultados e validações são apresentadas e interpretadas no Capítulo 4.

A conclusão do trabalho, que compõe o Capítulo 5, inclui os trabalhos futuros que são apresentados como desdobramentos naturais da presente dissertação.

Material relacionados a tópicos estudados e desenvolvidos, porém com uma importância de segunda ordem no contexto deste trabalho, são apresentados na forma de Apêndices.

2 FUNDAMENTOS TEÓRICOS DE COSMOLOGIA COMPUTACIONAL

A Cosmologia moderna tem o grande desafio de descrever a natureza do Universo, tendo como base um imenso e complexo arsenal teórico e observacional (ROSA, 2012). As observações do espectro infravermelho a longas distâncias e da radiação cósmica de fundo por *Far Infrared Absolute Spectrophotometer* (FIRAS), *Cosmic Background Explorer* (COBE) e *Wilkinson Microwave Anisotropy Probe* (WMAP) mostram evidências de que o Universo é homogêneo e isotrópico em grandes escalas (superaglomerados de centenas de Megaparsecs) (LIDDLE, 2003). Entretanto, localmente, o Universo se apresenta como um sistema não homogêneo composto por filamentos e vazios (*voids*) que são gerados a partir da amplificação das flutuações primordiais durante a sua expansão. Portanto, dentro do escopo da Cosmologia Moderna (ver Apêndice A), a simulação da formação de estruturas em grandes escalas visa reproduzir a distribuição hierárquica de flutuações, filamentos e vazios observados a partir dos grandes telescópios espaciais tanto paro o Universo jovem (WMAP) como para o Universo em seus estágios mais recentes *Sloan Digital Sky Survey* (SDSS).

2.1 Formação de Estruturas em Grandes Escalas

A evolução do Universo na teoria cosmológica moderna pode ser dividida em várias etapas ou eras: (i) Era de Planck; (ii) Era da inflação; (iii) Era da radiação; (iv) Era da nucleossíntese; (v) Era da recombinação; (vi) Era da reionização; (vii) Era presente (MO et al., 2010). Os modelos cosmológicos consideram que as grandes estruturas do Universo começaram a se formar a partir da reionização (CHABRIER, 2009). Nessa era, a gravidade começa a dominar sobre as forças electromagnéticas e nucleares e, por isso, as simulações de N-Corpos que estudam esses processos podem considerar em uma de suas aproximações apenas as forças de interação gravitacional. As condições iniciais são geradas a partir das flutuações da radiação cósmica de fundo, considerando modelos do universo antes da reionização (MO et al., 2010). Para que os modelos sejam considerados válidos, as simulações devem gerar distribuições de densidade consistentes com as observações.

O movimento da matéria é tratado como um fluido cósmico e é modelado pela equação da continuidade, equação de Euler e equação de Poisson, dadas respectivamente

pelas seguintes equações:

$$\begin{cases} \frac{\partial \rho}{dt} + \nabla \cdot (\rho\, v) &= 0, \\ \frac{\partial v}{\partial t} + (v \cdot \nabla)v &= -\frac{1}{\rho}\nabla P - \nabla\Phi, \\ \nabla^2\Phi = \Delta\Phi &= 4\pi G\rho. \end{cases} \tag{2.1}$$

onde Φ é o potencial gravitacional, G a constante Gravitacional universal, ρ densidade de matéria e v é a velocidade local do fluido (MO et al., 2010; PEEBLES, 1980). A equação da continuidade impõe a lei da conservação da massa, a equação de Euler descreve o movimento a partir do potencial gravitacional e a equação de Poisson relaciona a densidade das flutuações com o potencial gravitacional.

Em seguida, por meio da identidade $\nabla \cdot (\rho\, v) = \rho \cdot (\nabla\, v) + v \cdot \nabla\rho$ e da definição de derivada total $\frac{d}{dt} = \frac{\partial}{\partial t} + v \cdot \nabla$, a Equação 2.1 pode ser escrita da seguinte forma:

$$\begin{cases} \frac{d\rho}{dt} + \rho\nabla \cdot v &= 0, \\ \frac{dv}{dt} = -\frac{1}{\rho}\nabla P &- \nabla\Phi, \\ \Delta\Phi &= 4\pi\rho, \end{cases} \tag{2.2}$$

Essa formulação que é Newtoniana[1], é valida se as escalas das pertubações são menores que o raio de Hubble $d_h = \frac{c}{H_0}$ (MCCREA; MILNE, 1934), onde c é a velocidade da luz e H_0 é a velocidade de expansão atual do Universo em $[km/(Mpc\, s)]$.

2.2 Simulação N-Corpos

As simulações N-Corpos cosmológicas que modelam a evolução gravitacional do Universo consideram dois tipos de distribuições de matéria: discretas (partículas) e contínuas (fluido) (MO et al., 2010). Nesta dissertação, consideramos que a densidade de massas de uma região do Universo é representada por uma distribuição de elementos de massa discretas iguais (partículas) (GREEN, 2000). Portanto, uma vez definidas as condições iniciais, o movimento de cada elemento (partícula) é calculado numericamente considerando as forças de interação gravitacional com todos os outros.

Se consideramos N partículas, cada uma com massa m, distribuídas numa caixa cosmológica (assumimos aqui um cubo com lados de comprimento L (Figura 2.1)), as equações do movimento 2.2 resultam num sistema de equações diferenciais de

[1] fluido incompressível e com viscosidade constante

$6N$ variáveis, onde o estado de cada partícula é definido pelo vetor posição $\mathbf{r_j}$ e velocidade $\mathbf{v_j}$ como segue:

$$\frac{d\mathbf{r_j}}{dt} = \mathbf{v_j}, \tag{2.3}$$

$$\frac{d\mathbf{v_j}}{dt} = -\nabla\Phi_j, \tag{2.4}$$

onde

$$\nabla\Phi_j = \frac{F_j}{m_j} = G\sum_{k\neq j}^{N}\frac{m_k(\mathbf{r_k}-\mathbf{r_j})}{\mid \mathbf{r_k}-\mathbf{r_j} \mid^{\mathbf{3}}}. \tag{2.5}$$

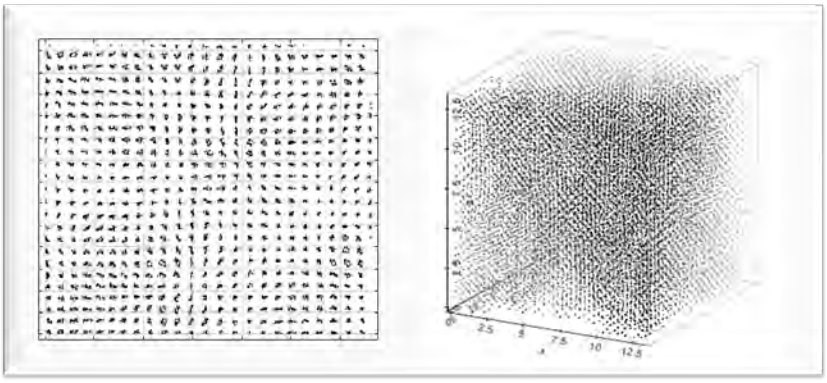


Figura 2.1 - Disposição inicial das partículas na caixa cosmológica L^3

As "simulações N-Corpos" cosmológicas consideram um número muito grande de partículas ($N > 10^5$) (BERTSCHINGER, 1998) que interagem de forma acolisional. Um fator de suavização ϵ é introduzido para evitar a singularidade quando a distância entre duas partículas se aproxima de zero e garantir a precisão dos métodos de integração:

$$\nabla\Phi_j = \frac{F_j}{m} = G\sum_{k\neq j}^{N}\frac{m(\mathbf{r_k}-\mathbf{r_j})}{(\mid \mathbf{r_k}-\mathbf{r_j} \mid^{\mathbf{2}} +\epsilon^{\mathbf{2}})^{3/2}} \tag{2.6}$$

o fator de suavização usualmente é associado ao *half-mass radius* de uma galáxia padrão (AARSETH, 2003), que é definido como sendo o raio que contém a metade da massa total da galáxia.

A expansão do Universo é considerada utilizando as coordenadas comóveis que permitem acompanhar a expansão utilizando o fator de escala $a(t)$ como ilustrado na

Figura 2.2, considerando:

$$\mathbf{x} = \frac{\mathbf{r}}{\mathbf{a(t)}}, \qquad \mathbf{v} = \mathbf{u} + \mathbf{Hr} \tag{2.7}$$

onde usualmente $a(t_f) = 1$ (hoje = período atual do Universo), $H(t)$ é o parâmetro de Hubble (hoje H_0) e $\mathbf{u} := \mathbf{a\dot{x}}$ é a velocidade peculiar. A variação do fator de escala é modelada pelas equações de Friedmann:

$$\frac{\dot{a}}{a} = H(t) = H_0\sqrt{\Omega_R\, a^{-4} + \Omega_M\, a^{-3} + \Omega_K\, a^{-2} + \Omega_\Lambda} \tag{2.8}$$

$$\frac{\ddot{a}\, a}{\dot{a}} = \Omega_M/2 - \Omega_\Lambda - \Omega_R \tag{2.9}$$

onde Ω_R é a densidade da radiação, Ω_M é a densidade de matéria (escura mais bariônica), Ω_K é a densidade da curvatura espacial e Ω_Λ é a constante cosmológica ou densidade de energia do vácuo (MO et al., 2010). No lado direito da Figura 2.2, observamos que a evolução do fator de escala se modifica com a composição do Universo, a curva vermelha corresponde ao modelo ΛCDM.

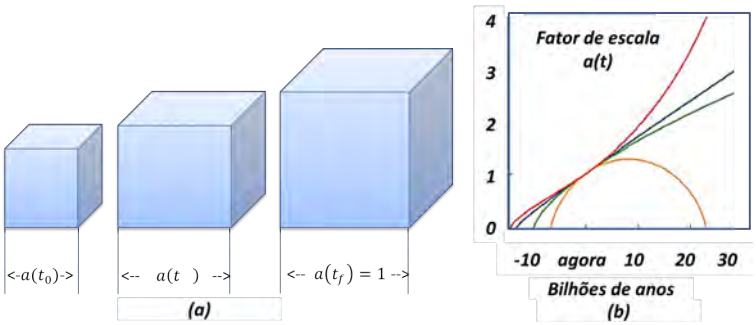


Figura 2.2 - (a) Efeito do fator de escala na simulação; (b) Evolução do Fator de Escala em relação as diferentes composições do Universo

Finalmente, utilizando as equações 2.7, 2.3 e 2.6, temos as equações do movimento em coordenadas comóveis para o sistema com N partículas:

$$\text{Para} \quad j = 1, \ldots, N,$$

$$\begin{cases} \frac{d\mathbf{x_j}}{dt} = \frac{\mathbf{u_j}}{a(t)}, \\ \frac{d\mathbf{u_j}}{dt} + H\mathbf{u_j} = -a(t)^{-2} G \sum_{k\neq j}^{N} \frac{m_k(\mathbf{x_k}-\mathbf{x_j})}{\left(|\mathbf{x_k}-\mathbf{x_j}|^2+(\epsilon/\mathbf{a(t)})^2\right)^{3/2}}. \end{cases} \tag{2.10}$$

2.2.1 Condições iniciais

As observações analisadas por Hinshaw et al. (2012) indicam que o espectro de flutuações primordiais é gaussiano e invariante com a escala, também denominado de espectro de Harrison-Zeldovich, isto é $P_{initial}(k) \propto k^n$, onde $n \approx 0.9646$ e k é o número de onda (HINSHAW et al., 2012). A densidade das flutuações primordiais (ver Figura 2.3) é modificada durante a evolução do Universo até a reionização. Para calcular o espectro final são considerados o plasma, a matéria escura, a densidade de neutrinos e fótons. Finalmente, as condições iniciais são obtidas a partir do espectro final das flutuações. Para representar as densidade das flutuações, as posições e as velocidades são perturbadas a partir de uma grade de partículas distribuídas na caixa cosmológica, utilizando a teoria linear da perturbação (ZEL'DOVICH, 1970).

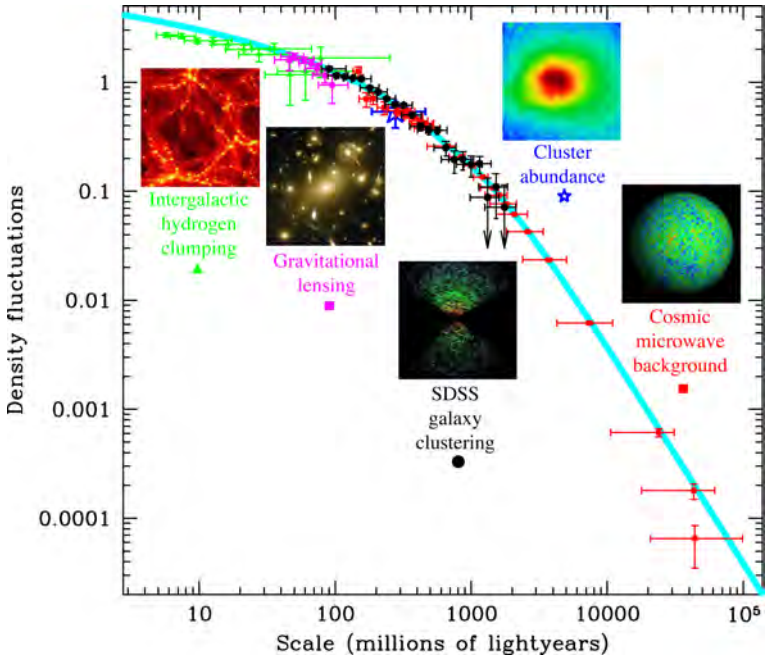


Figura 2.3 - Densidade das flutuações da radiação cósmica de fundo observada
Fonte: Hinshaw et al. (2012)

A massa de cada partícula é definida a partir da densidade de matéria, a densidade crítica ρ_c, o volume da caixa L^3 e o número de partículas N^3, como segue:

$$m = m_j = \Omega_M \frac{\rho_c L^3}{N^3} \quad \forall j \tag{2.11}$$

a implementação das condições iniciais serão discutidas na seção 3.2.2.

2.2.2 Condições de Contorno

As simulações cosmológicas são realizadas considerando um volume finito, consequentemente para representar o Universo todo e eliminar efeitos de borda utiliza-se condições de contorno periódicas, que também são consistentes com o princípio cosmológico (BERTSCHINGER, 1998; MO et al., 2010).

Em 2D, as condições de contorno periódicas são implementadas considerando a existência de 8 caixas que são réplicas de uma caixa central durante a simulação, consequentemente em 3D temos 26 réplicas (ver Figura 2.4). As posições e velocidades das partículas das réplicas são as mesmas da caixa central. Se uma partícula da caixa central se movimenta, as partículas das réplicas realizam o mesmo movimento. Se uma partícula sai da caixa central ela entra na face oposta com mesma velocidade, isto é, o número de partículas se conserva (AARSETH, 2003; HOCKNEY; EASTWOOD, 1988; BODNHEIMER et al., 2007). Vale destacar que, quando calculamos as forças de interação que agem sobre cada partícula, definimos uma nova caixa em torno dela, de maneira a somar a força de interação de cada partícula só uma vez, para isso o volume da nova caixa não deve ultrapassar o volume da caixa central (considera a partícula na Figura 2.4).

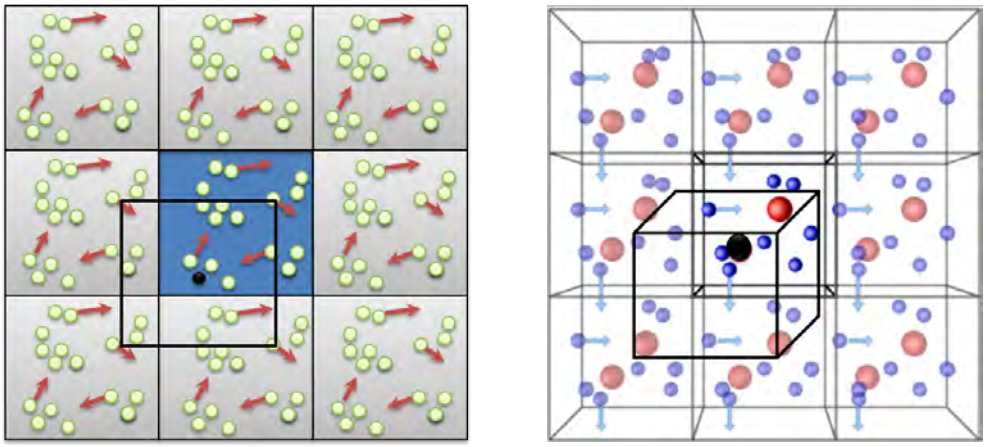


Figura 2.4 - Condições de contorno periódicas.
Fonte: http://isaacs.sourceforge.net/phys/pbc.html

2.2.3 Cálculo das Forças de Interação

O cálculo das forças de interação para cada partícula é o processo que demanda maior recurso computacional (principalmente tempo de processamento), consequentemente nos últimos anos foram desenvolvidos uma ampla variedade de algoritmos mais eficientes para este propósito, entre os quais se destacam (BERTSCHINGER, 1998; BAGLA, 2008):

- Soma Direta Partícula-Partícula (DSPP): ele é o mais realista porque considera todas as interações partícula-partícula, além de ser o único que permite facilmente definir um traçador Lagrangeano para cada partícula. Entretanto sua complexidade computacional é $\approx O(N^2)$.
- Algoritmos de árvore (*Tree Code*), desenvolvido por Barnes e Hut (1986), consegue reduzir a complexidade para $\approx O(N \log N)$ dividindo o volume em sub-caixas menores. Para calcular as forças que agem sobre uma partícula, este método considera individualmente todas as partículas que estão dentro da sub-caixa correspondente usando o esquema PP, as partículas que estão mais distantes são agrupadas em cada sub-caixa e após isso consideradas na somatória.
- Partícula-grade (PM, do inglês Particle-Mesh): Neste caso, o potencial gravitacional é definido sobre uma grade a partir da densidade das flutuações utilizando a equação de Poisson. A solução dessa equação utilizando a transformada rápida de Fourier permite calcular as forças de interação entre cada elemento da grade, reduzindo assim a complexidade computacional a $\approx O(N_g \log N_g)$, onde N_g é o número de nós da grade.
- Partícula-grade adaptativa (APM): esse método ajusta o tamanho da grade em função do contraste de densidade para aumentar a precisão nas regiões de maior densidade.
- Partícula-3-grade(P3M): esse esquema combina o método partícula-partícula e o método partícula-grade com o objetivo de aumentar a precisão.

Neste trabalho, utiliza-se o método DSPP, pois ele permite extrair diretamente informação da dinâmica de partículas individuais. Embora seu custo computacional seja elevado, a utilização da tecnologia GPU/CUDA permite considerá-lo em um abordagem para cosmologia computacional.

2.2.4 Integração das Equações do Movimento

Considerando a abordagem introduzida por Hockney e Eastwood (1988) podemos integrar as equações de Friedmann (2.8) usando as aproximações de Taylor como seguem:

$$a^{n+1} = a^n + \dot{a}^n dt + \ddot{a}^n \frac{dt^2}{2} + O(dt^3). \tag{2.12}$$

A soluções das equações de Friedmann (2.8) quando consideramos o modelo padrão mostram que o Universo se expandiu não linearmente (ver curva vermelha do fator de escala da Figura 2.2). Portanto, é conveniente substituir o tempo pelo parâmetro $p = a^\alpha$ definido como sendo a potência do fator de escala conforme descrito em (EFSTATHIOU G.; DAVIS, 1985). Quando p é introduzido na equação (2.10) obtém-se:

$$\begin{array}{l} \text{Para} \quad j = 1, \ldots, N. \\ \left\{ \begin{array}{rcl} \frac{d\mathbf{x_j}}{dp} & = & \frac{\mathbf{u'_j}}{a}, \\ \frac{d\mathbf{u'_j}}{dp} + 2\,A\mathbf{u'_j} & = & -B\nabla_x\phi(\mathbf{x_j}), \\ \nabla_x\phi(\mathbf{x_j}) & = & G\sum_{k\neq j} \frac{m(\mathbf{x_j}-\mathbf{x_k})}{\left(|\mathbf{x_j}-\mathbf{x_k}|^{\mathbf{2}}+(\epsilon/\mathbf{a})^{\mathbf{2}}\right)^{3/2}}. \end{array} \right. \end{array} \tag{2.13}$$

onde $u'_j = \frac{d\mathbf{x_j}}{dp}$, $A(p) = \frac{\alpha + \frac{\ddot{a}a}{a^2}}{2\alpha\, p}$ e $B(p) = \frac{1}{(\alpha\, p\dot{a})^2}$. Note que essa troca é opcional e para trabalhar explicitamente com o tempo considera-se $A(t) = \frac{H(t)}{2}$ e $B(t) = \frac{1}{a^2}$ trocando t por p.

Logo, trocamos as derivadas da posição e da velocidade pelas respectivas equações em diferenças para integrar numericamente as equações do movimento de cada partícula j, isto é :

$$\frac{d\mathbf{x_j^{n+1/2}}}{dp} \approx \frac{\mathbf{x_j^{n+1}} - \mathbf{x_j^n}}{\Delta p}, \tag{2.14}$$

$$\frac{d\mathbf{u_j^{n+1/2}}}{dp} \approx \frac{\mathbf{u_j^{n+1/2}} - \mathbf{u_j^n}}{\Delta p/2}. \tag{2.15}$$

$$\frac{d\mathbf{u_j^{n+1/2}}}{dp} \approx \frac{\mathbf{u_j^{n+1}} - \mathbf{u_j^{n+1/2}}}{\Delta p/2}, \tag{2.16}$$

e a partir das equações anteriores temos o esquema de integração denominado Leap-

Frog (ver Figura 2.5):

$$\mathbf{u_j^{n+1}} = \mathbf{u^n}\frac{\mathbf{1}-\mathbf{A(p_n)\Delta p}}{\mathbf{1}+\mathbf{A(p_n)\Delta p}} + \mathbf{B(p_n)}\frac{\nabla_{\mathbf{x}}\phi(\mathbf{x_j^{n+1/2}})\mathbf{\Delta p}}{(\mathbf{1}+\mathbf{A(p_n)\Delta p}} \quad (2.17)$$

$$\mathbf{x^{n+3/2}} = \mathbf{x^{n+1/2}} + \mathbf{\Delta p\, u^{n+1}}. \quad (2.18)$$

Vale destacar que o primeiro e o último passo são calculados pelas seguintes equações:

$$\mathbf{u_j^{1/4}} = \mathbf{u^0}(\mathbf{1}-\mathbf{2\,A(p_0)\Delta p}/4) + \mathbf{B(p_0)}\nabla_{\mathbf{x}}\phi(\mathbf{x_j^0})\mathbf{\Delta p}/4 \quad (2.19)$$

$$\mathbf{x^{1/2}} = \mathbf{x^0} + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{\Delta p\, u^{1/4}}, \quad (2.20)$$

e que, quando o parâmetro p é considerado, não é necessario integrar as equações de Friedmann pois o fator de escala depende diretamente do parâmetro, conforme:

$$p^{n+1} = p^n + dp^n \quad (2.21)$$

$$a^{n+1} = (p^n)^{1/\alpha}, \quad (2.22)$$

onde α pode ser considerado $2/3$ (EFSTATHIOU G.; DAVIS, 1985). O método Leap-Frog é consistente, estável e os erros de truncamento são de $O(\Delta p^3)$. Para implementar métodos de maior ordem, é necessário calcular e salvar as acelerações adicionais, o que aumentaria os requerimentos de memória (GREEN, 2000; AARSETH, 2003).

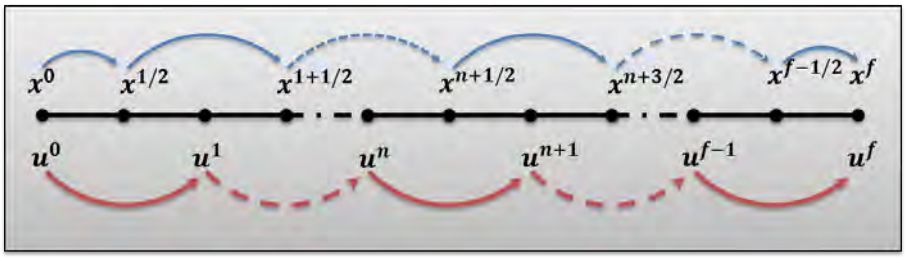


Figura 2.5 - Esquema Leap Frog

Conforme Bodnheimer et al. (2007) e Aarseth (2003), o tamanho do passo de tempo Δp pode ser calculado proporcionalmente à velocidade e à aceleração de cada partícula. Consequentemente, para utilizar um passo de tempo único para todas as partículas, utiliza-se a máxima aceleração e velocidade da caixa para calculá-lo, como segue:

$$\Delta p \leq \sqrt{\frac{\eta}{a_{max}}} \quad \text{e} \quad \Delta p \leq \frac{\eta}{v_{max}}, \quad (2.23)$$

onde η é uma constante adimensional que se ajusta de forma a manter a conservação da energia.

2.2.5 Equações da Energia: Condição de Layzer-Irvine

A lei da conservação da energia para uma distribuição de partículas em coordenadas comóveis é dada pela equação da energia de Layzer-Irvine,

$$\frac{da^2T}{dt} + a\frac{dU}{dt} = 0, \tag{2.24}$$

onde T é a energia cinética e U é a energia potencial da distribuição de partículas (HOCKNEY; EASTWOOD, 1988). Elas são calculadas como segue:

$$T = \frac{1}{2}\sum_{i=1}^{N} m\,\mathbf{u_i^2}. \tag{2.25}$$

$$U = \frac{1}{2}\sum_{i=1}^{N} \Phi(x_i), \tag{2.26}$$

onde $\Phi(x_i)$ é o potencial gravitacional comóvel na posição x_i da partícula i, isto é:

$$\Phi(x_i) = \sum_{j\neq i}^{N} \frac{m_i m_j\, G}{\mid x_i - x_j \mid}. \tag{2.27}$$

Logo, a equação (2.24) pode ser integrada da seguinte maneira:

$$a^2T + aU + \int_{a(t_0)}^{a(t)} U da = C, \tag{2.28}$$

ou também

$$a\,T + U + \int_{a(t_0)}^{a(t)} a^2 T da = C'. \tag{2.29}$$

A complexidade do cálculo da energia potencial é $O(N^2)$ e da energia cinética é $O(N)$. Em geral, utiliza-se a equação (2.29), que na forma discreta é:

$$a_n\,T_n + U_n + \sum_{k=1}^{n} a_k^2 T_k da = C', \tag{2.30}$$

dessa forma, essa última equação é utilizada como uma medida da precisão da integração numérica.

Considerando os formalismos apresentados até aqui os passos necessários para realizar uma simulação N-Corpos cosmológica estão resumidos na Figura 2.6.

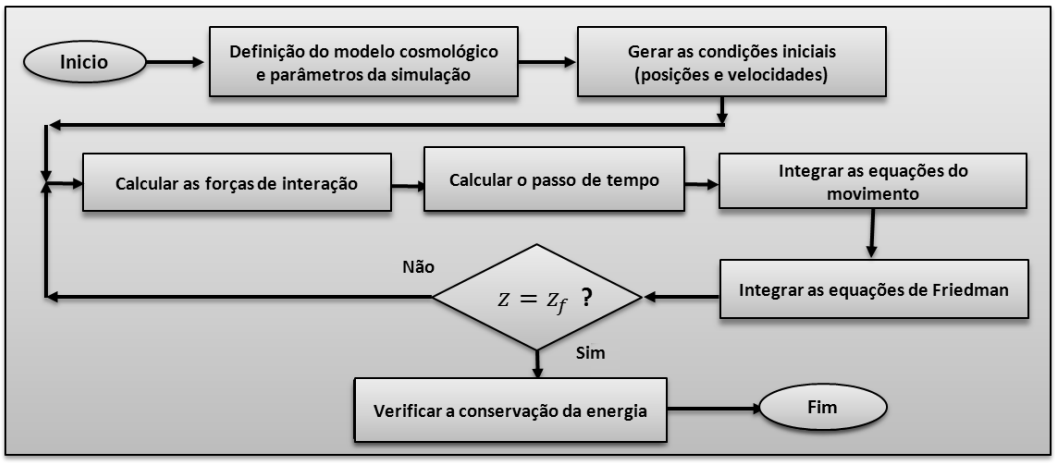


Figura 2.6 - Esquema de passos de uma simulação N-corpos usualmente adotado em cosmologia computacional

2.3 Ferramentas de Validação e Análise

Os resultados das simulações N-Corpos devem ser analisados e validados, comparando as propriedades físicas das estruturas geradas com as observadas. Primeiramente, para identificar essas estruturas (halos de galáxias, halos de grupos/aglomerados) utiliza-se o algoritmo *Friends of Friends* Huchra e Geller (1982) cujas aplicações em cosmologia computacional está bem estabelecida e testada (RUIZ, 2011; CARETTA et al., 2008; SPRINGEL et al., 2005). A função de correlação de dois pontos permite comparar a distribuição de matéria com uma distribuição aleatória em diferentes escalas (BERTSCHINGER, 1998). Nas seguintes seções descreveremos com mais detalhes cada uma delas.

2.3.1 Algoritmo *Friends of Friends* (FoF)

No caso de uma simulação N-Corpos de matéria escura, o algoritmo FoF permite identificar halos de estruturas conforme sua dimensão (galáxias, grupos/aglomerados ou superaglomerados), usando uma variável R denominada "raio de percolação". O raio de percolação permite ajustar o tamanho médio das estruturas a serem identi-

ficadas. De forma resumida o algoritmo FoF funciona da seguinte maneira: define-se uma esfera de raio R centrada em cada partícula do conjunto total. As partículas que estiverem dentro dessa esfera são chamadas "amigas" e são adicionadas a um determinado grupo (por exemplo, grupo 1). Para cada "amiga" encontrada o procedimento é repetido recursivamente buscando identificar as "amigas das amigas" e adicionando-as ao mesmo grupo. Quando nenhuma nova "amiga" pode ser adicionada ao grupo em questão, o processo é finalizado (como mostrado na Figura 2.7).

O resultado desse procedimento são grupos de partículas limitados por uma superficie de densidade aproximadamente constante $\frac{n}{\overline{n}} \propto \frac{1}{2R^3}$, onde n é o número total de objetos e $\overline{n}$ é o número médio no volume considerado. O valor de R é escolhido de forma que o contraste de densidade seja próximo ao valor esperado para um halo virializado conforme o modelo de colapso esférico. Usualmente, para halos de aglomerados, o valor estimado é $R = 0.2$ (LACEY; COLE, 1994).

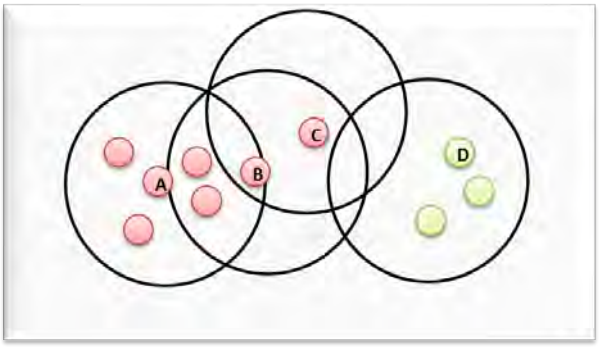


Figura 2.7 - Esquema esquema FoF: A,B,C são amigas, mas A e D não são

Para identificar estruturas menores, como halos de galáxias, pode-se utilizar uma combinação do algoritmo FoF, com $R = 0.1$ e do teorema do Virial. O teorema do Virial estabelece que se um sistema está relaxado dinamicamente, existe uma relação de equilíbrio entre as energias médias Potencial e Cinética, de modo que $2\,T = U$, logo:

$$2\,T + U = 0 \Rightarrow m\,V^2 + \frac{mM\,G}{r} \Rightarrow V = \sqrt{\frac{G\,M}{r}} \tag{2.31}$$

onde T é a energia cinética do halo, U é a energia potencial gravitacional do halo, e V é dispersão de velocidade das partículas de cada halo. Este teorema permite selecionar os halos gerados pelas simulações que estão em equilíbrio ou não, analisando a relação entre a dispersão de velocidade e massa dos halos identificados.

2.3.2 Função de Correlação de Dois Pontos

A função correlação de dois pontos permite caracterizar as não homogeneidades da distribuição em relação as diferentes escalas. O esperado é que em grandes escalas ela seja mais homogênea, e em escalas menores menos homogênea.

A função correlação de dois pontos $\xi(r)$ é definida por Peebles (1980) a partir da probabilidade dP de encontrar dois objetos separados por uma distância r, considerando duas amostras δV_1 e δV_2, com número de densidade n. Isto é:

$$dP = n^2(1 + \xi(r))\delta V_1 \delta V_2, \tag{2.32}$$

Na prática a função de correlação de dois pontos pode ser estimada contando os pares de objetos com diferentes separações (r). Na literatura, existem diferentes estimadores:

(i) Peebles e Hauser (1974):

$$\xi(r) = \frac{n_r}{nd}\left(\frac{DD(r)}{RR(r)} - 1\right) \tag{2.33}$$

(ii) Davis e Peebles (1983):

$$\xi(r) = \frac{2\,n_r}{(nd-1)}\left(\frac{DD(r)}{DR(r)} - 1\right) \tag{2.34}$$

(iii) Hamilton (1993):

$$\xi(r) = \frac{4\,n_r\,n_d}{(nd-1)(nr-1)}\left(\frac{DD(r)}{DR(r)}\frac{RR(r)}{DR(r)} - 1\right) \tag{2.35}$$

(iv) Landy e Szalay (1993):

$$\xi(r) = \frac{1}{RR}\left[DD\left(\frac{n_r}{nd}\right)^2 - 2\,DR\left(\frac{n_r}{nd}\right) + RR\right] \tag{2.36}$$

onde $DD(r)$, $DR(r)$ e $RR(r)$ são o número de pares separados até (r) considerando os pares dados-dados, dados-catálogo e catálogo-catálogo respetivamente. Os parâmetros n_d e n_r são os número de objetos nos dados e catálogos, respetivamente. Os catálogos são distribuições de objetos gerados aleatoriamente(distribuição de Poisson) e os dados são as distribuições geradas com as simulações. Entre os estimadores

citados, o de Landy e Szalay (1993) é o mais utilizado por ser superior em relação a precisão e estabilidade (KERSCHER et al., 2000).

3 SIMULAÇÃO N-CORPOS COM GPU-CUDA PARA COSMOLOGIA

O aumento contínuo da demanda computacional ao longo dos anos nas aplicações científicas e industriais impulsionou o desenvolvimento de novas arquiteturas de hardware e técnicas de computação paralela (ver Figura 3.1). A capacidade de processamento tem sido incrementada a partir de várias estratégias: (i) Agrupando ou interconectando várias unidades de processamento em *clusters*. As unidades de processamentos são máquinas de memória compartilhada, tipicamente com vários processadores multicore compondo os *clusters*, que são máquinas paralelas de memória distribuída; (ii) Aumentando a capacidade de processamento das CPUs por meio do aumento da velocidade do *clock* e uso da tecnologia *multi-core*; (iii) Combinando-se diferentes dispositivos de *hardware* com a CPU, por meio da abordagem conhecida como "computação híbrida". Entre os dispositivos de *hardware*, as Unidades de Processamento Gráficas (GPUs) têm se destacado, sobretudo devido a seu enorme poder computacional e baixo custo. Neste trabalho, utiliza-se a computação híbrida (CPU+GPU) para desenvolver um simulador N-Corpos considerando uma abordagem em CUDA (JUNIOR et al., 2012).

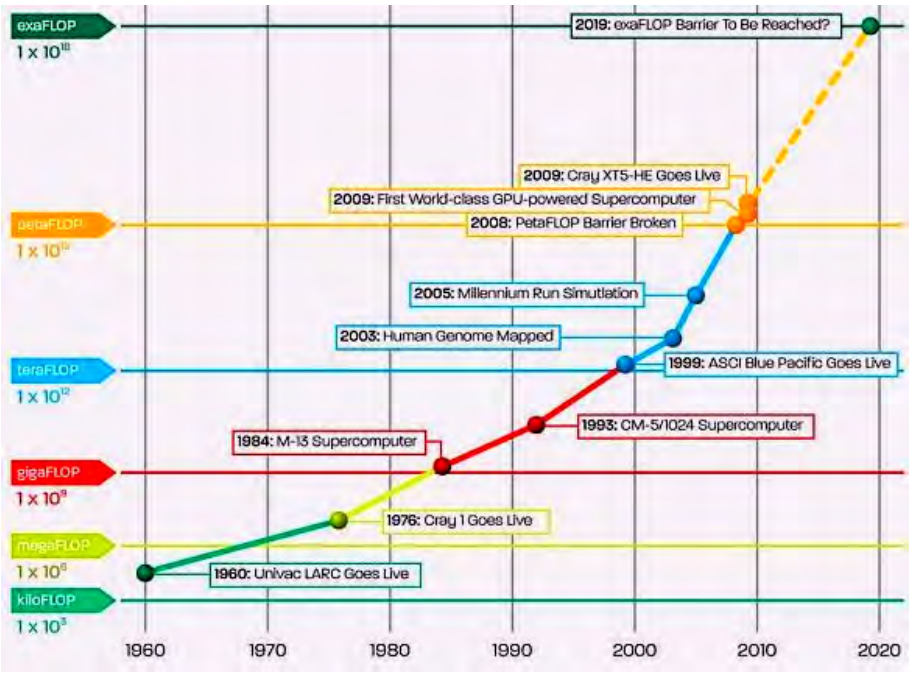


Figura 3.1 - Aumento da Capacidade de processamento dos *clusters*
Fonte: Disponível em: http://blogs.amd.com/

O nosso sistema de computação híbrida consiste em um *host* (hospedeiro), que é uma CPU de propósito geral, e um ou mais *devices* (dispositivos), que são processado-

res massivamente paralelos. Nessa arquitetura, os *devices* são usados para acelerar a execução das seções do programa que apresentam paralelismo de dados. O paralelismo de dados refere-se à propriedade do programa pela qual muitas operações aritméticas podem ser realizadas sobre um conjunto de dados de maneira simultânea (KIRK; HWU, 2011). Maiores detalhes sobre a tecnologia GPU/CUDA no contexto da computação híbrida são apresentados no Apêndice C.

3.1 Estrutura de um Programa em CUDA

Um programa CUDA consiste em uma ou mais etapas que são executadas no *host* ou no *device* (GPU). A parte que apresenta paralelismo de dados é implementada no *device* e a parte que necessita menos processamento é implementada no *host*. O programa pode ser escrito em código ANSI C estendido, que permite escrever funções denominadas *kernels* e que são executadas em um ou mais dispositivos (GPUs). O compilador C da NVIDIA(nvcc) separa o código *host* do código do *device* durante o processo de compilação (KIRK; HWU, 2011). Para explorar a capacidade de processamento da GPU as funções *kernels* devem gerar um grande número de *threads*, que são escalonadas de forma independente. Um esquema da execução de um programa CUDA é mostrado na Figura 3.2. Inicia-se a partir do código sequencial executado no *host* e quando uma função *kernel* é chamada, gera um número grande de *threads* que serão executados concorrentemente no *device* (GPU) . Um conjunto de *threads* é chamado de bloco e um conjunto de blocos é denominado *grid* de blocos. Quando todas as *threads* de um *kernel* completam sua execução, a grade correspondente termina e a execução continua no *host* até que outro *kernel* seja invocado. O conjunto de blocos pode ser executado em qualquer ordem, simultânea ou sequencialmente. As *threads* de um mesmo bloco são executadas de forma sincronizada, utilizando identificadores (*blockIdx* e *threadIdx*) empregados para seu escalonamento e para explorar o paralelismo de dados.

Os *devices* (GPUs) tem memória própria. Antes de executar uma função *kernel*, devemos então alocar memória no *device* e transferir os dados de entrada da memória principal da unidade de processamento para a memória do *device*. Uma vez finalizado o processamento dos dados no *device*, é preciso transferir os resultados do *device* para o *host* e liberar a memória alocada no *device*.

A memória do *device* tem uma organização hierárquica como é mostrado na Figura 3.3. Conforme mostra a Figura, o *host* pode transferir dados da memória do sistema para a memória global ou a memória constante e no sentido contrário também. No caso da memória constante, o acesso do *device* é possível só no modo leitura. A

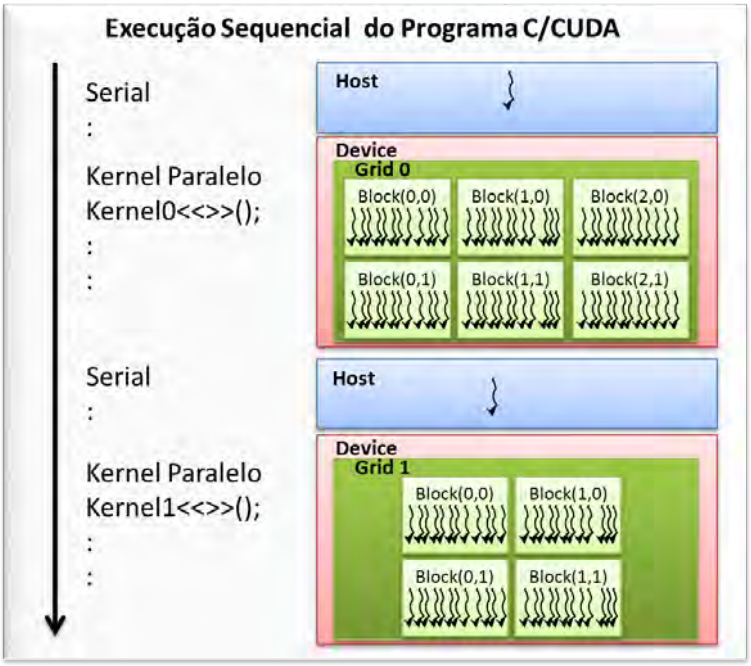


Figura 3.2 - Sequência de execução de um programa em CUDA
Fonte: Kirk e Hwu (2011)

memoria compartilhada é reservada para cada bloco, mas pode ser acessada só pelos *threads* que pertencem ao mesmo bloco.

3.2 Simulações de N-Corpos com Tecnologia GPU/CUDA

Conforme apresentado no Capítulo 2, uma simulação N-Corpos cosmológica demanda muito processamento. O cálculo das forças de interação gravitacional para cada partícula é a etapa de maior complexidade computacional $O(N^2)$. A comunidade da ciência da computação não tardou em aplicar as GPUs para melhorar o desempenho dos simuladores de N-corpos. Os primeiros resultados apresentados em 2006 (NYLAND et al., 2007; ELSEN et al., 2006), considerando 16384 partículas e o algoritmo DSPP, obtiveram $200Gflops$ [1] de desempenho. Para essas simulações, foi utilizada uma GeForce 8800 GTX, que tem 128 SPs.

Outras simulações considerando algoritmos mais sofisticados para o cálculo das forças obtiveram um pico de desempenho de $28.1Tflops$, considerado 1.6 bilhões de partículas e utilizando um supercomputador com 256 GPUs e algoritmos de árvore para calcular as forças (HAMADA et al., 2009) (ver Figura 3.4).

[1] *flops* do inglês, operações de ponto flutuante por segundo

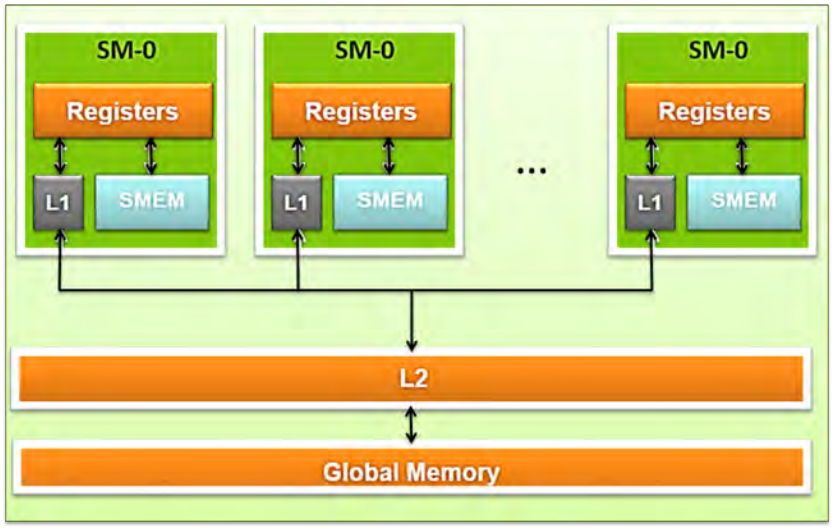


Figura 3.3 - Visão geral do modelo de memória de um *device* da arquitetura Fermi
Fonte: Adaptada de `http://www.nvidia.com`

Em Gaburov et al. (2009), é apresentado um algoritmo híbrido que realiza cálculos concorrentes em (CPU) e (GPUs) emulando o funcionamento de um hardware especificamente desenvolvido para realizar simulações N-Corpos. As simulações consideram 10^6 partículas com um pico desempenho de até $800 Gflops$, utilizando quatro GPUs GeForce 9800GX2.

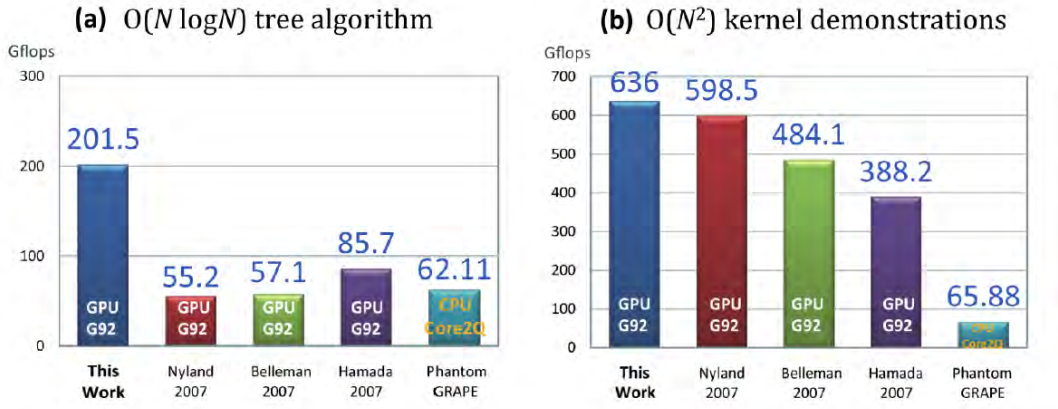


Figura 3.4 - Desempenho das simulações N-Corpos em GPU: (a) Considerando o esquema de árvore, (b) Considerando o esquema DSPP
Fonte: Hamada et al. (2009)

Esses resultados mostram que as simulações em GPUs conseguem melhorar o desem-

penho dos simuladores baseados em CPUs (ver Figura 3.4). O nosso trabalho foca-se em considerar mais física nas simulações e desenvolver um simulador N-Corpos em GPU para estudar o processo de formação de estruturas.

3.2.1 Esquema Geral de uma Simulação N-Corpos em CUDA

Aplicando os conceitos introduzidos nas seções anteriores o esquema que seguimos para realizar uma simulação N-Corpos é mostrado na Figura 3.5.

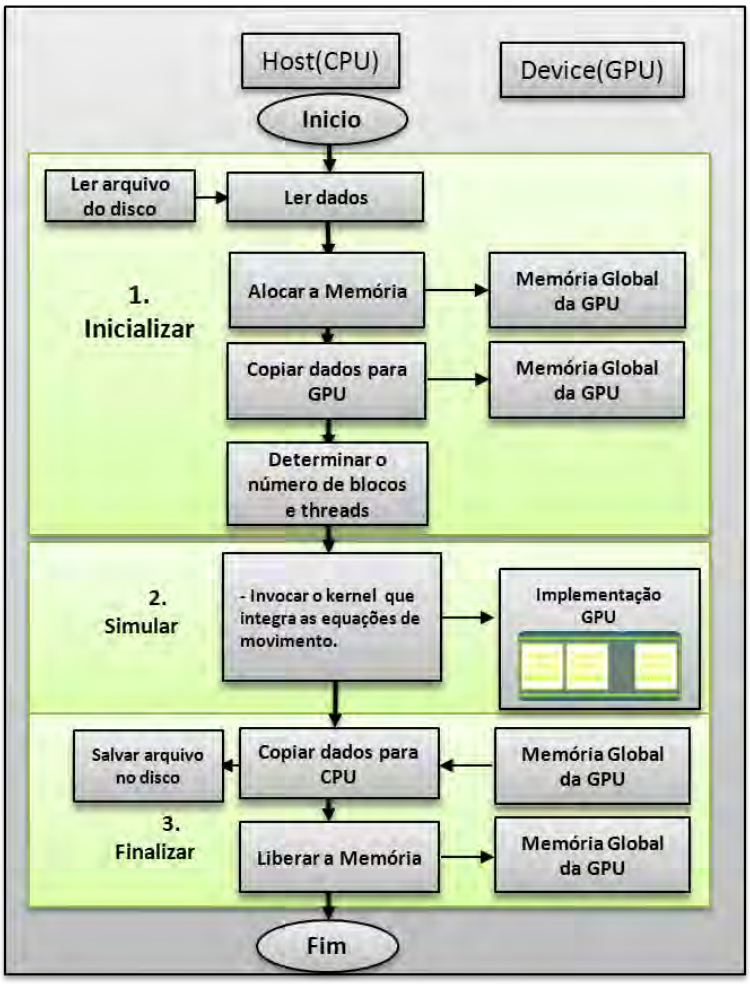


Figura 3.5 - Esquema da simulação N-Corpos em GPU

Neste trabalho, utilizou-se dois computadores:

a) Jaburu: processador Intel Core I7 950 3.06GHz Quad-Core LGA 1366, 4GB de memória RAM 1333MHz, 1 TB de hard disk e uma Nvidia GeForce GTX 580 1.5GB GDDR5

b) Parayba: processador Intel Sandy Bridge-E Core I7 3930K 3.2 GHz Six-Core, 4GB de memória RAM 1600MHz, 3 TB de hard disk e duas Nvidia GeForce GTX 680 2GB GDDR5 (ver Figura 3.6).

O Sistema Operacional adotado foi a versão Linux Ubuntu 10.10. A GTX 580 possui 32 núcleos por SM, 48 multiprocessadores SM, e um total de 1536 núcleos SP por chip. A GTX 680 possui 192 núcleos SP por cada SMX, 8 multiprocessadores SM e um total de 1536 núcleos SP por chip. Os principais recursos em cada GPU estão disponíveis através de um padrão chamado *compute capability*. Especificamente, a GTX 580 possui *compute capability* 2.0 e a GTX 680 possui *compute capability* 3.0, todas as suas especificações técnicas e recursos encontram-se, no Apêndice C.4.

Nosso objetivo foi desenvolver um simulador N-Corpos em linguagem C/C++ CUDA para reduzir o tempo das simulações utilizando computação híbrida com GPU. Para tal, foram desenvolvidas diversas versões de programas para realizar as mesmas tarefas: uma versão sequencial e outras paralelas.

Figura 3.6 - Computador LAC/INPE - SJC. Projeto atmosfera massiva II-Cnpq Proc.560178/2010-7

O simulador N-Corpos foi dividido em três etapas(ver figura 3.7):(1) Geração das condições iniciais (ii) COLATUS, e (iii) Visualização dos Resultados. A geração das condições iniciais foi realizada utilizando um adaptação do código desenvolvido por Volker Springel disponível em `http://www.mpa-garching.mpg.de/gadget/`.

Desenvolveram-se duas versões do COLATUS uma serial para uma CPU e outra paralela em CUDA (GPU), com o objetivo de comparar o desempenho das simulações. Para visualizar os resultados das simulações foi desenvolvida também uma aplicação em CUDA.

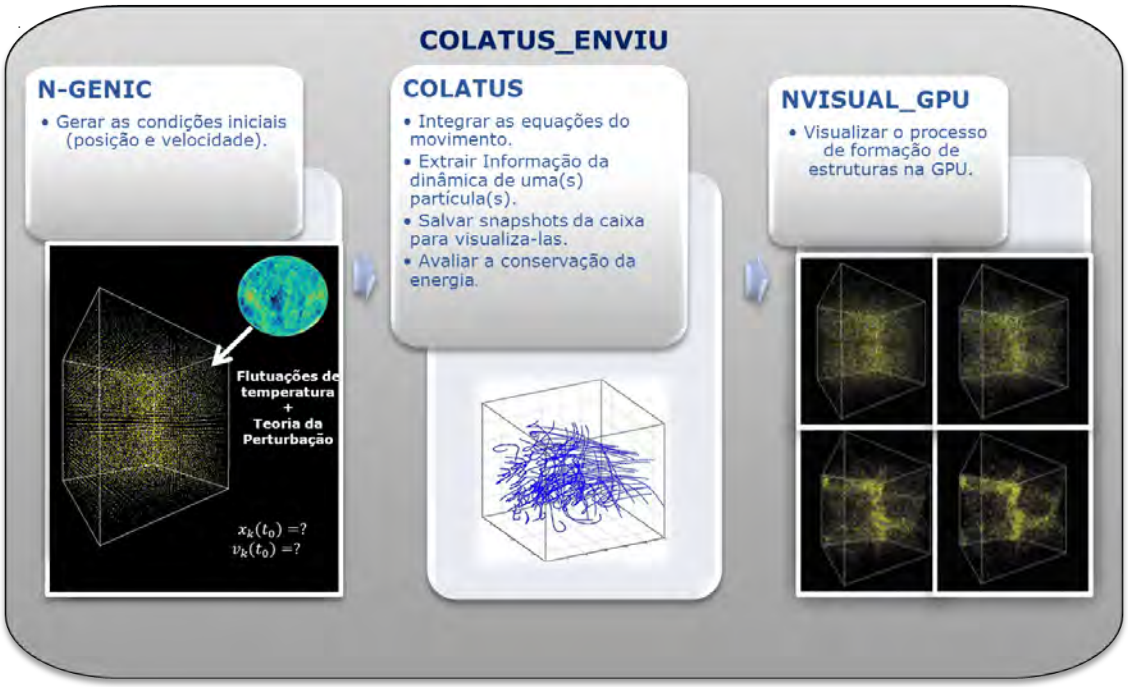


Figura 3.7 - Blocos do Simulador

Nas seguintes seções, descrevemos o código, as estruturas de dados, os arquivos de configuração de entrada e de saída de cada bloco.

3.2.2 Gerador de Condições Iniciais N-GENIC

O gerador de condições iniciais foi desenvolvido usando C e MPI (*Message Passing Interface*) e bibliotecas FFTW (*Fastest Fourier Transform* [2], para calcular as Transformadas de Rápidas de Fourier) e GSL (GNU *Scientific Library*[3], bibliotecas do calculo numérico). O código N-GENIC está disponível em `http://www.mpa-garching.mpg.de/gadget/`.

Esta aplicação lê um arquivo de entrada (um exemplo do arquivo de entrada se encontra no Apendice D.1.2) onde são definidos os parâmetros cosmológicos, como o

[2]Disponível em: http://www.fftw.org/
[3]Disponível em: http://www.gnu.org/software/gsl/

tamanho da caixa, *redshift* inicial, quantidade de matéria bariônica e escura, energia escura, parâmetro de Hubble, tipo de espectro de potência e as unidades de medidas. Os parâmetros de entrada são os seguintes:

- Número máximo do número de onda, (*Nsample*). Se n é o número total de partículas da simulação, $Nsample = n^{1/3}$.
- Tamanho da grade (*Nmesh*) usada para calcular as perturbações, deve ser maior ou igual a *Nsample*.
- Parâmetro de Hubble (h), considera-se $h = 0,70 \pm 2,2$ (HINSHAW et al., 2012).
- Tamanho da caixa (BOX), normalmente é escolhido sendo 100 [Mpc/h] (BERTSCHINGER, 1998; J.S.BAGLA; RAY, 2005; BAGLA, 2008).
- Nome do arquivo de saida (*FileBase*) e diretório (*OutputDir*) onde serão salvas as condições iniciais.
- Nome do arquivo que contém a grade de partículas $(n')^3$ homogênea e uniforme (*GlassFile*).
- Número de partículas: ela é fixada pelo parâmetro $TileFac = \dfrac{(n)^3}{(n')^3}$
- Densidade de matéria (bariônica + escura) $\Omega = 0.2793$ (HINSHAW et al., 2012).
- Densidade de energia escura $\Omega_\Lambda = 0.721$ (HINSHAW et al., 2012).
- Densidade de matéria bariônica $\Omega_b = 0.0463$ (HINSHAW et al., 2012).
- Parâmetro de Hubble $h = 0.700$, (HINSHAW et al., 2012).
- *Redshift* inicial $z < 100$.
- Normalização do espectro de potência $Sigma8 = 0.821$ (HINSHAW et al., 2012).
- Tipo de espectro de potência inicial *WhichSpectrum*. Pode ser ajustado conforme a formulação de (EISENSTEIN; HU, 1998) ou (EFSTATHIOU G.; DAVIS, 1985). As simulações do Consorcio Virgo (KAUFFMANN J.M. COLBERG, 1999) usam a formulação de Efstathiou com o parâmetro de forma $ShapeGamma = 0.21$.

- O espectro de potência primordial invariante com as escalas:

  $P_{initial}(k) \propto k^{PrimordialIndex}$, onde $PrimordialIndex = 1$.

- Unidade de comprimento ($UnitLength\ in\ cm$):$[Mpc/h]$

- Unidade de massa ($UnitLength\ in\ g$): $[10^{10} M_{\odot}/h]$, onde $M_{\odot} = 1,9885 \times 10^{33}$.

- Unidade de velocidade ($UnitLength\ in\ cm\ s$): $[10km/s]$.

Depois de definir os parâmetros da simulações executamos o código usando o seguinte comando:

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

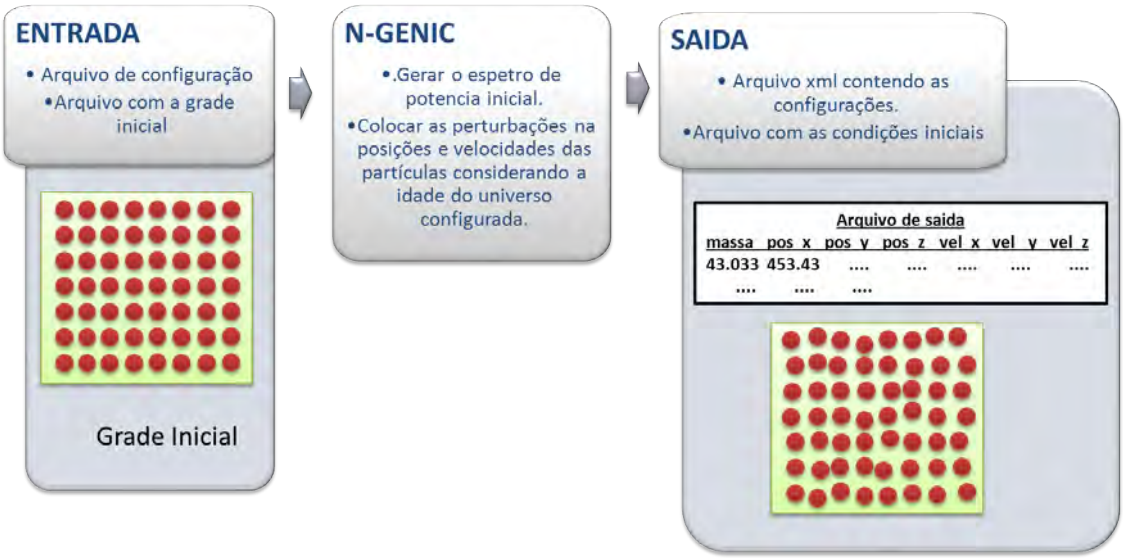


Figura 3.8 - Gerador de condições iniciais: N-GENIC

A saída do código original são vários arquivos binários, mas foram criadas novas funções que adaptam a saída para que o COLATUS leia as condições iniciais de um único arquivo. Além disso geramos um descritor dos dados usando a biblioteca LIBXML2 *.xml* que contém os parâmetros com os quais a condições iniciais foram geradas (ver exemplo em D.1.3). Um esquema geral do funcionamento do código é mostrado na Figura 3.8, onde temos um esquema que mostra os arquivos de entrada e saída do gerador de condições iniciais.

3.2.3 Simulador N-Corpos: COLATUS

O simulador N-Corpos foi desenvolvido em C,C++ e CUDA. Ele lê um arquivo de configuração e as condições inicias geradas pelo N-GENIC. O arquivo de configuração contém informações que indicam onde a simulação será realizada (GPU ou CPU), quantos *snapshots* serão salvos, a pasta e nome do arquivo das condições iniciais, etc, que serão detalhados na seguinte seção. O COLATUS realiza uma simulação de N-Corpos em escala cosmológica com coordenadas comoveis e condições de contorno periódicas tanto em CPU como em GPU. Ele calcula iterativamente as forças de interação, integra as equações de movimento e as equações de Friedmann (fator de escala). Finalmente os resultados são salvos em vários arquivos para sua posterior visualização e análise. O esquema geral do simulador é mostrado na Figura 3.9.

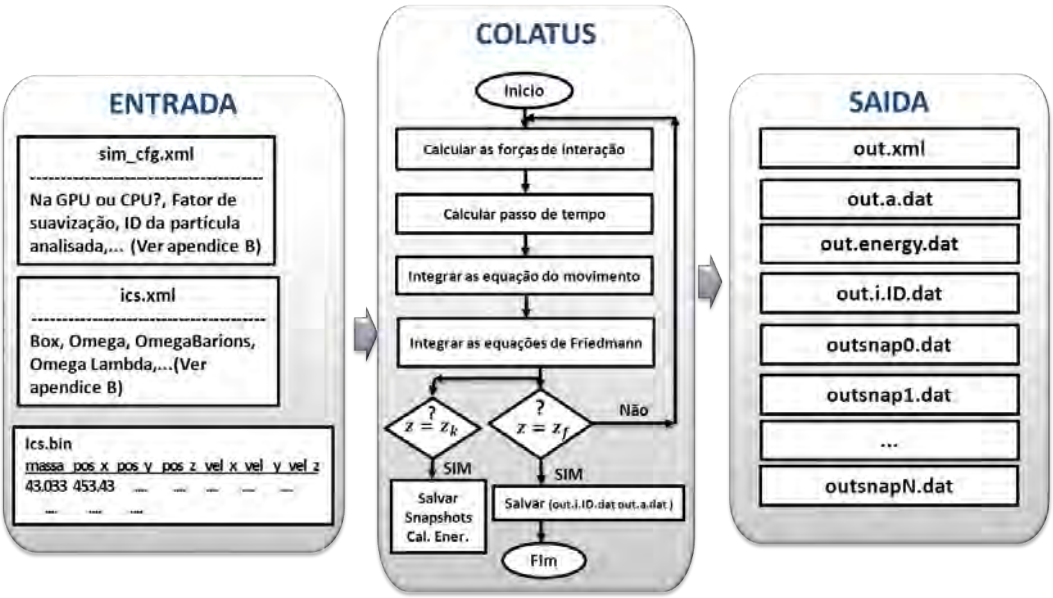


Figura 3.9 - Simulador de Turbulência Lagrangiana Cósmica: COLATUS

Na Figura 3.10, apresenta-se um esquema mais detalhado que mostra um diagrama de fluxo de uma simulação do COLATUS utilizando somente a CPU. Este esquema representa uma execução serial do COLATUS, e é utilizada para avaliar o ganho de desempenho ao utilizar a GPU. O COLATUS pode ser divido em três grandes etapas: (i) Inicialização; (ii) Computação e; (iii) Finalização. Inicializar o código consiste em ler os arquivos de entrada e alocar memória na CPU para salvar as posições, velocidades, fator de escala, etc. A etapa de Computação realiza o cálculo iterativo das forças de interação entre as partículas, as acelerações, o passo de tempo,

o fator de escala (equações de Friedmann). Em cada passo a posição de velocidade da partícula secionada para a análise é salva na memória. Quando integramos as equações que envolvem do fator de escala, o *redshift* se modifica até chegar ao valor desejado z_f (*Redshift Final* ou a quantidade máxima configurada no inicio). Antes de iniciar a simulação são definidos *redshift* $z_0, z_1, \ldots, z_k \ldots, z_{\text{maxsnap}}$ nos quais serão salvos o *snapshots* (posição e velocidade de todas a partículas). O processo iterativo é pausado para salvar os *snapshots*, em seguida é calculada a energia total do sistema que é utilizada posteriormente para validar os resultados. Ao finalizar a execução do COLATUS são gerados vários arquivos: (i) out.xml (tempo da simulação, quantidade de *snapshots* salvos, etc); (ii) out.i.ID.dat (posição e velocidade da partícula selecionada); (iii) out.energia.dat (a energia total de cada *snapshot*); (iv) out.a.dat(fator de escala); (v) os snapshots.

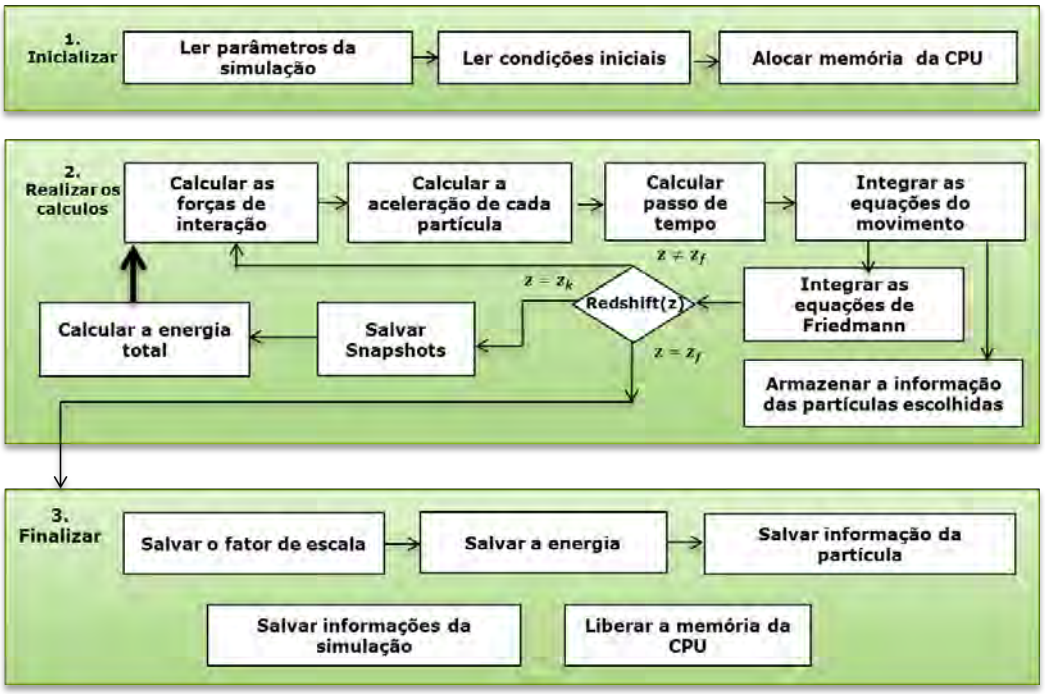


Figura 3.10 - Esquema da simulação N-Corpos em CPU

Na Figura 3.11, apresentamos as modificações necessárias para realizar uma simulação do COLATUS utilizando a CPU (*host*) e GPU (*device*). Neste caso, para inicializar a simulação, além de ler os arquivos de entrada e alocar memória no *host*, também devemos alocar memória no *device* e, logo após carregar as condições iniciais na memória do *host*, precisamos copiá-las para o *device*.

Na etapa de cálculos, o trabalho é distribuído entre a CPU e a GPU. A execução é inicializada e comandada pelo *host* que invoca a função *kernel* **ComputeAcceleration** para gerar uma grade unidimensional de N *threads* no *device*, onde cada *thread* será responsável pelo cálculo da aceleração de uma partícula em uma interação. O ID da partícula pela qual cada *thread* é responsável se determina usando os identificadores (*blockIdx* e *threadIdx*) e a equação 3.1:

$$ID = blockIdx\,\mathbf{nthreadpblock} + \mathbf{threadIdx}, \tag{3.1}$$

onde **nthreadpblock** é o número de *threads* de cada bloco. Esta quantidade é definida a partir das especificações da GPU utilizada para maximizar a ocupação de cada núcleo dos multiprocessadores. Então dividindo o número de partículas N pelo número de *threads* por bloco **nthreadpblock**, e arredondando o resultado ao inteiro subsequente sabemos quantos blocos serão gerados em cada chamada de um *kernel*.

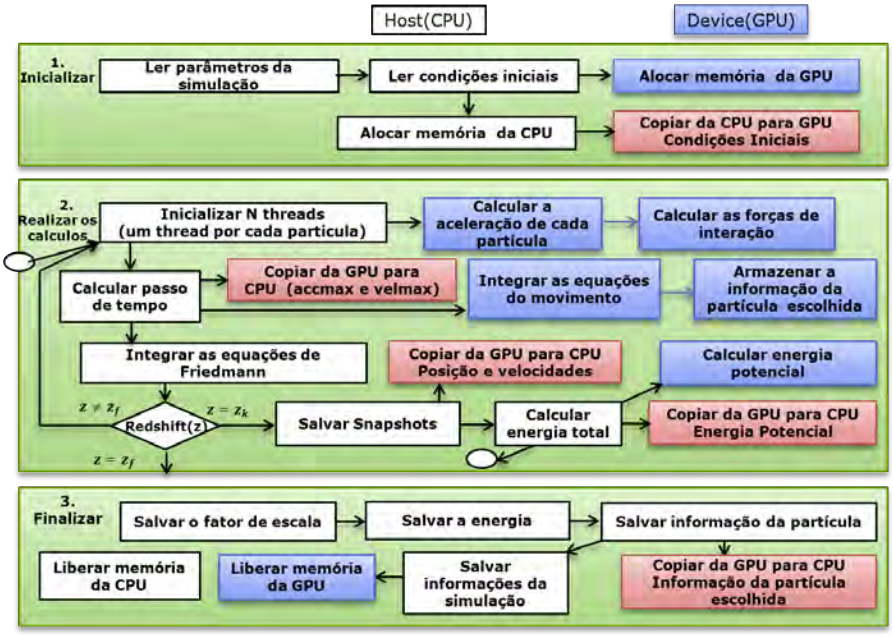


Figura 3.11 - Esquema da simulação N-Corpos em GPU

A função *kernel* **ComputeAcceleration** precisa calcular e somar as $N-1$ forças que agem sobre a partícula pela qual ele é responsável. Para isso é chamada $N-1$ vezes a função do *device* **bodyBodyInteraction**, gerando assim $N-1$ *threads*. Primeiramente devemos considerar partículas réplicas da caixa mencionadas na seção

2.2.2 para manter condição de homogeneidade (será explicado com detalhe na seção 3.2.3.3). A partir das somas das forças que agem sobre cada partícula é possível calcular a aceleração, a norma da aceleração máxima e da velocidade máxima, de tal forma que a seguinte função **Computetimestep** (no *host*) calcule o passo de tempo adequado usando as equações 2.23. Antes porém, os valores máximos da aceleração e da velocidade devem ser copiados da GPU para CPU. O passo de tempo é calculado na CPU para diminuir o erro introduzido pela função que calcula a raiz quadrada na GPU.

Em seguida, o *host* invoca outra função *kernel* denominada **Integrate**, que recebe o valor do passo de tempo calculado pela função **Computetimestep**, e a partir das acelerações que já estão na memoria global do *device* integra as equações de movimento para obter as novas posições e velocidades de cada partícula. Identicamente ao caso da função **ComputeAcceleration** são invocados N *threads*, um para cada partícula. A função **Integrate** deve considerar as condições de fronteira periódicas como foi explicado na seção 2.2.2, além disso ela é responsável por salvar a posição e velocidade da partícula selecionada para a análise em cada passo, dados que são passados para a memoria do *host* ao finalizar a simulação.

Quando a função **Integrate** finaliza os cálculos, o *host* integra as equações de Friedmann para modificar o fator de escala e calcular a nova idade do Universo, o *redshift* z. Se o valor de z pertence ao conjunto $z_k := \{z_0, z_1, \cdots, z_{maxSnap}\}$, o processo iterativo é interrompido para copiar as posições e velocidades de todas a partículas da memória do *device* para a memória do *host*, salvá-las num arquivo, e logo calcular a energia total da caixa para avaliar a conservação da energia durante a simulação. Neste caso, a energia potencial gravitacional é calculada no *device* pelo kernel **gEnergyT** que gera N *threads*, cada um deles responsável pelo cálculo da energia potencial total de uma partícula. A simulação finaliza quando se atinge o número máximo de iterações ou chega-se no *redshift* final z_f configurado pelo usuário. A finalização do código é idêntica ao caso serial, mas com a diferença que os dados para análise da partícula selecionada devem ser copiados primeiramente para a memória do *host* para salvá-las no arquivo. Libera-se então a memória global do *device* para que fique disponível para outras aplicações.

3.2.3.1 Arquivo de Configuração

No arquivo de configuração do COLATUS, são definidos os parâmetros da simulação que são os seguintes:

- *Device to run* = [GPU|CPU]: define se a simulação será executada na CPU ou na GPU.
- Será calculada a energia durante a simulação *Compute energy* = [Yes|No]. Observação: a energia é calculada só nas iterações nas quais é salvo o *snapshot*.
- *Save snapshots* = [Yes|No]: Define se salvaremos os *snapshots* durante a simulação e *Max Snapshots* a quantidade máxima de *snapshots*.

  Observação: *Max Snapshots* deve ser menor que o número máximo de interações porque é custoso copiar os dado para CPU e calcular a energia.
- Consideraremos as coordenadas comoveis *Use Comoving Coordinates* = [Yes|No].
- *Comov peculiar velocities= Yes|No*: Permite modificar a definição da velocidade salva em cada passo de tempo, se (Yes) $\mathbf{u} := \mathbf{a}\dot{\mathbf{x}}$, e se (No) $\mathbf{u} := \dot{\mathbf{x}}$.
- A variável *Use Variable Time step* = [Yes|Nao] habilita o cálculo de passo de tempo em cada iteração, e *Eta* é usado para calcular o passo de tempo usando a equação (2.23).
- *Softering epsilon*: O fator do suavização evita a singularidade no cálculo das forças.
- *Use Parametrization as Time*: Habilita o uso da parametrização usada na equação (2.13), e (*Alpha*) define relação entre a variável temporal p e o tempo.
- A simulação acaba quando chegamos ao *Redshift final* ou ao número máximo de iterações (*Max time steps*).
- *Tracking ID*: seleciona a partícula da qual serão extraídas as informações (posição e velocidade) em cada passo de tempo.
- *Binary output* = [Yes|No] : define se os *snapshots* serão salvos em arquivos binários ou ascii, e o parâmetro *output* define o nome base deles.
- *IC file*: Contém o nome do arquivo que descreve as condições iniciais *ics.xml*.

Um exemplo do arquivo de configuração se encontra no Apêndice D.1.2.

3.2.3.2 Alocação de Memória

A principais variáveis que são utilizadas na simulação são apresentadas na figura 3.12. Uma execução serial (na CPU) do COLATUS precisa de espaço na memória para colocar as posições (**pos**) e velocidades (**vel**) das partículas, o fator de escala (**a**), a energia cinética (**eT**), a energia potencial (**eU**), a energia total (**eTol**) e as informação da posição (**pos i**) e velocidade (**vel i**) da partícula selecionada para análise.

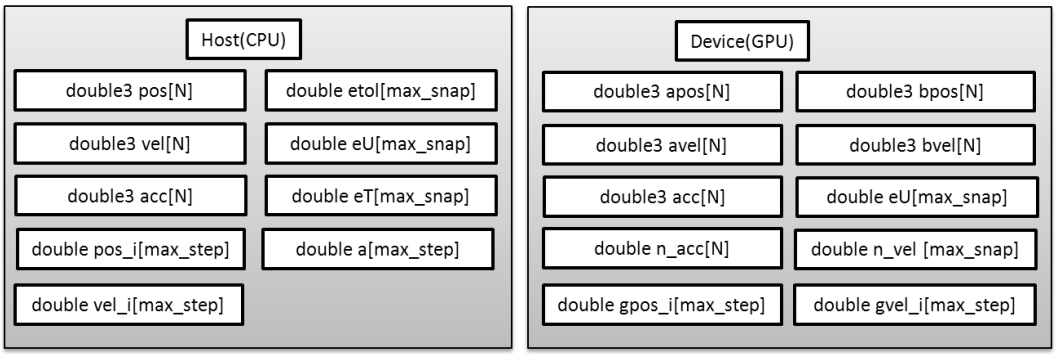


Figura 3.12 - Variáveis utilizadas pelo COLATUS na CPU e na GPU

Numa simulação hibrida CPU-GPU, também precisamos reservar espaço no *device*. Para eliminar a dependência de dados, foram declarados dois pares de vetores para salvar os valores das posições (**apos, bpos**) e velocidades (**avel, bvel**) respectivamente antes e depois da integração das equações de movimento (ver seção 3.2.3.4). Além dísso são considerados dois vetores para salvar as normas da aceleração (**nacc**) e da velocidade (**nvel**). Todas as variáveis são definidas como *double* para minimizar o erro de truncamento.

3.2.3.3 Integração das Equações do Movimento

A posição e a velocidade são calculadas integrando as equações 2.13, usando o esquema 2.17. Este processo é dividido em 4 etapas que serão detalhadas a seguir: (i) cálculo das forças de interação; (ii) cálculo da aceleração; (iii) cálculo do passo de tempo; (iv) cálculo das posições de velocidades. Considerando esta divisão reescrevemos as equações 2.17, mas introduzindo a aceleração $a_j^{n+1/2} = \frac{d\mathbf{u'_j}}{dp}^{n+1/2}$ como

segue:

$$\begin{aligned}
\text{Para} \quad j &= 1, \ldots, N, && (3.2)\\
\nabla_x \phi(\mathbf{x_j^{n+1/2}}) &= G \sum_{k \neq j} \frac{m(\mathbf{x_j^{n+1/2}} - \mathbf{x_k^{n+1/2}})}{\left(\mid \mathbf{x_j^{n+1/2}} - \mathbf{x_k^{n+1/2}} \mid^{\mathbf{2}} + (\epsilon/\mathbf{a})^{\mathbf{2}} \right)^{3/2}}. && (3.3)\\
\mathbf{a_j^{n+1/2}} &= -B(p_n) \nabla_x \phi(\mathbf{x_j^{n+1/2}}) - \mathbf{2}\, \mathbf{A(p_n) u^n} && (3.4)\\
\mathbf{u_j^{n+1}} &= \mathbf{u_j^n} + \frac{\mathbf{\Delta p}}{\mathbf{1} + \mathbf{A(p_n)}} \mathbf{a_j^{n+1/2}} && (3.5)\\
\mathbf{x_j}^{\mathbf{n+3/2}} &= \mathbf{x_j^{n+1/2}} + \mathbf{\Delta p}\, \mathbf{u_j^{n+1}}, && (3.6)
\end{aligned}$$

onde $A(p^n) = \frac{\alpha + \frac{\ddot{a}^n a^n}{(a^n)^2}}{2\alpha\, p^n}$ e $B(p^n) = \frac{1}{(\alpha\, p^n \dot{a}^n)^2}$.

Cálculo das forças

Nesta etapa calcula-se e soma-se as forças que agem sobre cada partícula usando o esquema partícula-partícula (PP). A implementação desta etapa é crítica por dois motivos:

(i) A complexidade computacional $O(N^2)$ do algoritmo (PP). Porém, as GPUs mostraram um bom ganho de desempenho em relação a CPU, embora a complexidade do algoritmo PP sejam proibitivamente elevada.

(ii) As condições de fronteira periódicas requerem considerar as caixas réplicas da original em torno da caixa inicial conforme foi explicado na seção 2.2.2.

Para considerar a existência das caixas réplicas no entorno, realizamos uma mudança do sistema de referência de tal forma que a partícula considerada fique no centro do sistema de referencia (χ) (ver Figura 3.13), isto é:

$$\begin{aligned}
\chi_{\mathbf{k}} &= \mathbf{x_k} - (\mathbf{x_j} + (\mathbf{L/2}, \mathbf{L/2}, \mathbf{L/2})) && (3.7)\\
\chi_{\mathbf{j}} &= (L/2, L/2, L/2)\,. && (3.8)
\end{aligned}$$

Colocando as mudanças de coordenadas na equação do potencial temos que:

$$\begin{aligned}
\text{Para} \quad j &= 1, \ldots, N, && (3.9)\\
\nabla_\chi \phi(\chi_{\mathbf{j}}^{\mathbf{n+1/2}}) &= G \sum_{k \neq j} \frac{m(\chi_{\mathbf{j}}^{\mathbf{n+1/2}} - \chi_{\mathbf{k}}^{\mathbf{n+1/2}})}{\left(\mid \chi_{\mathbf{j}}^{\mathbf{n+1/2}} - \chi_{\mathbf{k}}^{\mathbf{n+1/2}} \mid^{\mathbf{2}} + (\epsilon/\mathbf{a})^{\mathbf{2}} \right)^{3/2}}. && (3.10)
\end{aligned}$$

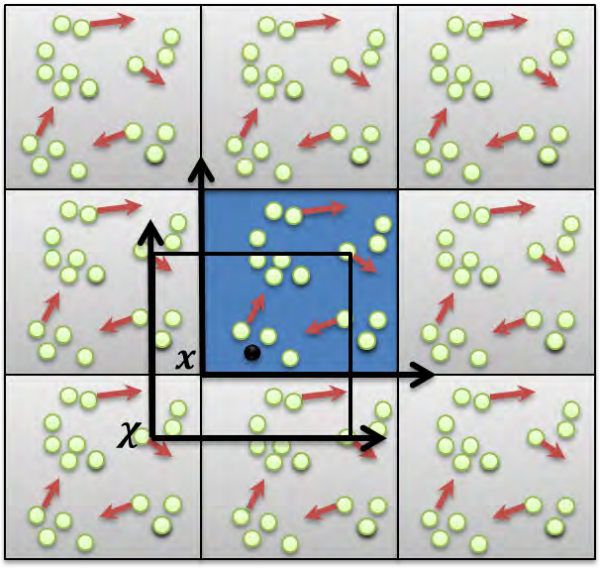


Figura 3.13 - Mudança do sistema de coordenadas

O cálculo deste potencial é feito na GPU pela função **bodyBodyInteration** (ver Apendice D.2.2.1).

Cálculo da aceleração

O passo de tempo deve ser escolhido para garantir a estabilidade e precisão do esquema numérico. A equação 2.23 descreve o critério usado para defini-lo a partir da norma da aceleração máxima e da velocidade máxima. A função **ComputeAcceleration** (do *device*, ver Apêndice D.2.2.2) calcula as acelerações, a norma da aceleração e a norma da velocidade de todas as partículas, e logo procura o valor máximo da norma da aceleração e da velocidade. Esta função invoca N *threads* da função **bodyBodyInteration**, descrita na seção anterior (ver Figura 3.14), para calcular o potencial gravitacional que age em cada partícula, e obter a aceleração utilizando a seguinte equação:

$$\mathbf{a_j} = -\mathbf{B(p)}\nabla\phi(\mathbf{x_j}) - \mathbf{2\,A(p)u} \tag{3.11}$$

Após todos os *threads* do *kernel* **ComputeAcceleration** terminarem as execuções, o *host* chama a função **Computetimestep**(ver Apêndice D.2.2.3), em que a partir dos valores máximos da norma da aceleração e da velocidade, calcula o passo de tempo e chama a função seguinte para integrar as equações do movimento.

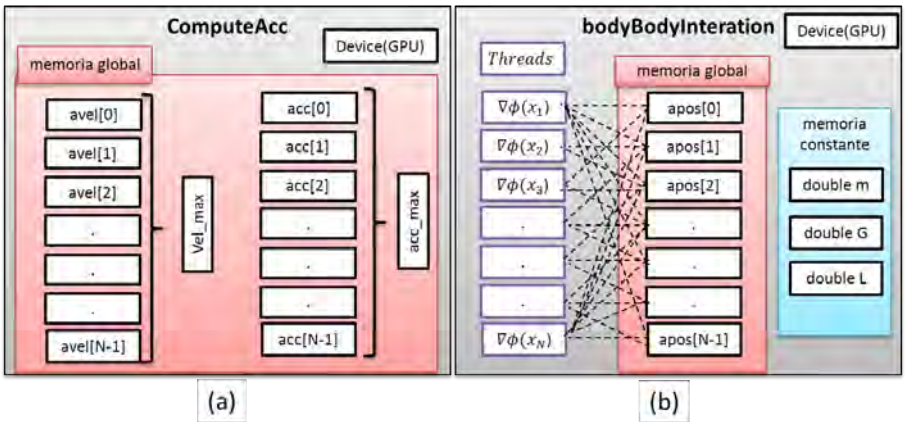


Figura 3.14 - (a) Variáveis utilizadas pela função **?**bfComputeAcceleration, (b)Variáveis utilizadas pela função **bodyBodyInteration**

3.2.3.4 Integração

A função **Integrate** é responsável por calcular as posições e velocidades quando é chamada. As *threads* não são executadas todas simultaneamente, porque o número de partículas é muito grande e os cálculos são executados em vários *warps*, e como cada *thread* precisa da informação do passo anterior para integrar as equações conforme o esquema explicado em 2.2.4, temos uma dependência de dados, que pode ser evitada utilizando dois pares de vetores: o primeiro para salvar posição e velocidades atuais e o segundo para salvar as novas posições e velocidades (ver Figura 3.15).

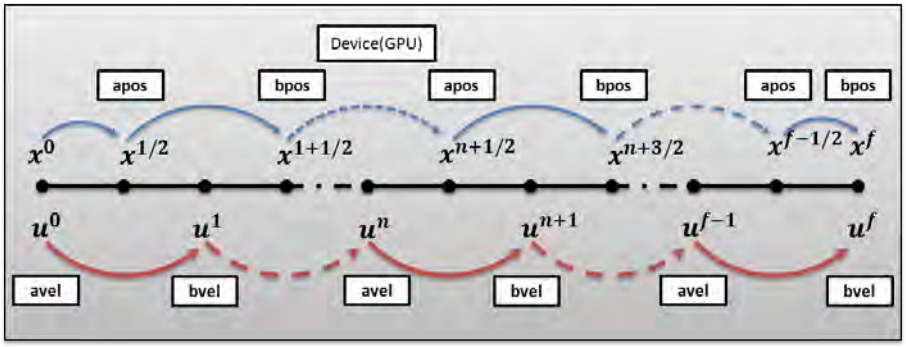


Figura 3.15 - Esquema Leap Frog na GPU

As condições de fronteira periódicas, conforme foi explicado na seção 2.2.2, impõem que as partículas fiquem dentro da caixa. A função *mod* que implementa a divisão

euclidiana (BOUTE, 1992) é utilizada com este objetivo, definida como segue:

$$q \in Z, \tag{3.12}$$

$$D = n\,q + r, \tag{3.13}$$

$$0 \leq r <\mid n \mid, \tag{3.14}$$

onde q é quociente inteiro de D e n. Se consideramos $n = L$ (tamanho da caixa) na equação 3.12, a nova posição da partícula seria r. Finalmente podemos reescrever as equações 3.2 como segue:

$$\text{For} \quad j = 1, \ldots, N, \tag{3.15}$$

$$\mathbf{u_j^{n+1}} = \mathbf{u_j^n} + \frac{\mathbf{\Delta p}}{\mathbf{1 + A(p_n)}}\mathbf{a_j^{n+1/2}} \tag{3.16}$$

$$\mathbf{x_j}^{\mathbf{n+3/2}} = \text{mod}\left(\mathbf{x_j^{n+1/2}} + \mathbf{\Delta p\, u_j^{n+1}}, \mathbf{L}\right), \tag{3.17}$$

onde $\mathbf{x_j}^{\mathbf{n+3/2}}$ sempre ficará dentro da caixa como esperado (para mais detalhes ver Apêndice D.2.2.4).

Integração das equações de Friedmann

A variação do fator de escala a em relação ao tempo é descrita pelas equações de Friedmann 2.8. Elas podem ser integradas pelas aproximações de Taylor conforme explicado na seção 2.2.4. No COLATUS, elas são integradas na CPU. A partir do valor inicial do *redshift*, calculamos o valor inicial do fator de escala como segue $a_{ini} = 1/(z_{ini} + 1)$. Se usamos o parâmetro $p = a^\alpha$ como variável temporal o seguinte valor de a^{n+1} é definido a partir do passo de integração $p^{n+1} = p^n + \Delta p$, do que segue $a^{n+1} = (p^{n+1})^{1/\alpha}$. Quando usamos a parametrização de Efstathiou G.; Davis (1985), não precisamos usar as aproximações de Taylor. Porém a precisão da integração vai depender do valor de α (Ver Apêndice D.2.2.5).

3.2.3.5 Cálculo da Energia

O cálculo e armazenamento da energia total da caixa durante a simulação permite avaliar a precisão da integração numérica pela conservação da energia. A conservação da energia num sistema comovel é verificada usando as equações de Layzer-Irvine

apresentadas na seção 2.2.5. Para facilitar a leitura, reescrevemos a Equação 2.30:

$$a_n\, T_n + U_n + \sum_{k=1}^{n} a_k^2 T_k da = C'_n, \tag{3.18}$$
$$T_n = \frac{1}{2}\sum_{i=1}^{N} m\,(\mathbf{u_i^n})^{\mathbf{2}}.\mathbf{U_n} = \frac{\mathbf{1}}{\mathbf{2}}\sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{\Phi}(\mathbf{x_i^n}),$$

onde o valor de C'_n deve ficar constante ou variar numa porcentagem muito baixa para garantir a precisão das simulações. O custo computacional de calcular a energia potencial U_n é $O(N^2)$, por isso foi desenvolvida uma função *kernel* **gEnergyT** para calculá-la (ver Apêndice D.2.4). Como os *snaspshots* são salvos em intervalos predefinidos separados por várias iterações, o terceiro termo da equação $\sum_{k=1}^{n} a_k^2 T_k da$ é calculado separadamente em cada passo de tempo usando a regra do trapézio (ver Apêndice D.2.4).

3.2.3.6 Descrição do Arquivo de Saída: out.xml

No arquivo de configuração do COLATUS são definidos os parâmetros da simulação, que são os seguintes:

- *NumBodies*: número de partículas da simulação.
- *Box*: Tamanho da caixa.
- *Outputbasefilename*: nome base dos arquivos de saída.
- *Snapshots*: o que determina quantos snapshots foram salvos.
- *Binaryoutput* = [Yes|Nao]: define se os *snapshots* foram salvos em arquivos binários ou ascii.
- *Snapxx*: são os *redshifts* de cada um dos *snapshots* salvos.
- *Outfileenergy*: o nome do arquivo que contém a evolução da energia durante a simulação.
- *Outfiletracking*: o nome do arquivo onde estão as informações da partícula selecionada para a análise.
- *Outfilescalerfactor*: o nome do arquivo onde foram guardados os valores do fator de escala a durante a simulação.

- *Simulationconfig*: o nome do arquivo que contém configurações da simulação.

Um exemplo do arquivo de saída se encontra no Apêndice D.2.5. Este arquivo é usado como entrada da aplicação de visualização que será descrita a seguir.

3.2.4 Visualização dos Resultados: NVISUALGPU

A aplicação de Visualização NVISUALGPU permite ler os arquivos de saída do COLATUS renderizá-las utilizando as GPUs (ver Figura 3.16). Ela é uma adaptação da primeira versão do COLATUS em CUDA desenvolvida na UFF. Essa versão CUDA permite realizar simulações N-Corpos em GPU e renderizar as posições das partículas em tempo real utilizando OpenGL. A visualização e simulação em tempo real requer mais recursos computacionais da GPU. Consequentemente a adaptação consistiu na criação de funções para ler os parâmetros de saída do COLATUS e arquivos de dados de maneira a tornar possível renderizar os resultados após realizada a o término da simulação (ver Figura 3.17).

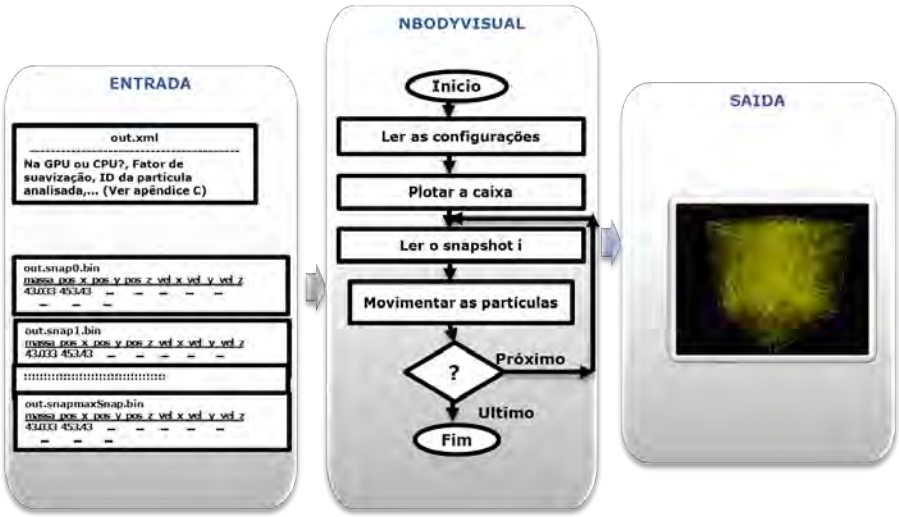


Figura 3.16 - Esquema do programa de visualização: NVISUALGPU

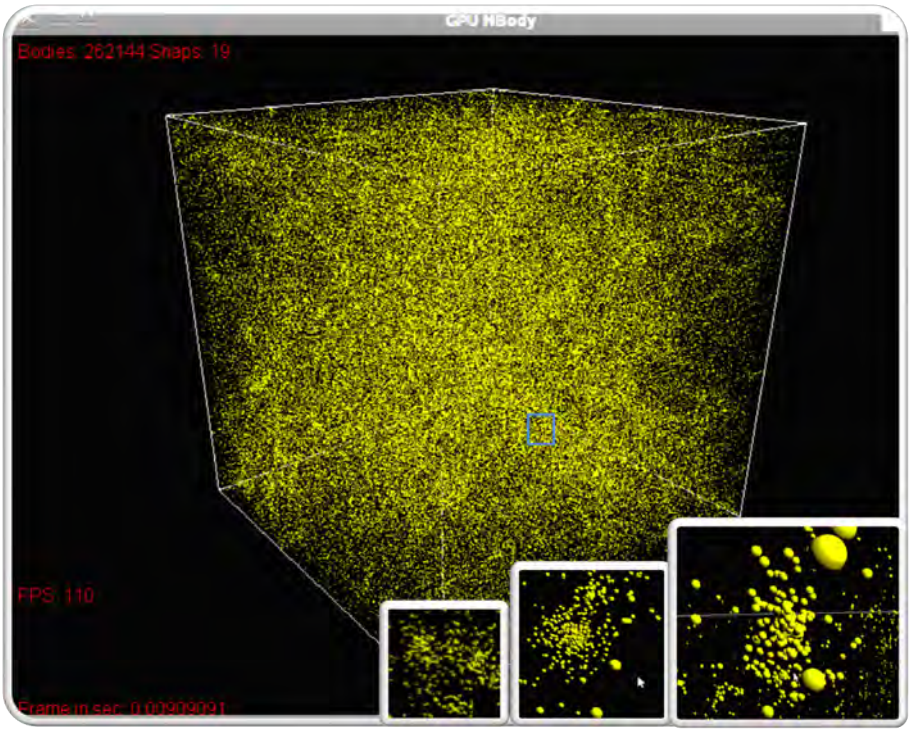


Figura 3.17 - Interface gráfica para visualização na GPU

4 PRIMEIRAS SIMULAÇÕES, RESULTADOS E VALIDAÇÕES

4.1 Simulações N-Corpos do Modelo ΛCDM

Os resultados apresentados nesta seção sáo provenientes de simulações realizadas considerando o modelo ΛCDM, ou seja, baseando-se nos mesmos parâmetros das simulações do Consorcio Virgo Kauffmann J.M. Colberg (1999), para posteriores comparações. Entretanto, outros parâmetros podem ser simulados alterando-se apenas o arquivo de configuração (simcfg.xml) e gerando as condições iniciais correspondentes. Os parâmetros considerados nas simulações realizadas são mostrados na tabela 4.1. Para todos os casos foi utilizado $\Omega_K = 0$, $\Omega_R = 8.010^{-5}$.

Tabela 4.1 - Parâmetros usados nas simulações realizadas:

	Ω_m	Ω_Λ	h	N Partículas	L Mpc/h	m $10^{10}M_\odot/h$	ϵ Kpc/h	σ_8	z_{init}
(a)	0.3	0.7	0.72	16^3	8.8	1.4	20	0.9	30
(b)	0.3	0.7	0.72	24^3	13.2	1.4	20	0.9	30
(c)	0.3	0.7	0.72	32^3	17.6	1.4	20	0.9	30
(d)	0.3	0.7	0.72	48^3	26.5	1.4	20	0.9	30
(e)	0.3	0.7	0.72	64^3	35.3	1.4	20	0.9	30
(f)	0.3	0.7	0.72	80^3	44.1	1.4	20	0.9	30
(d)	0.279	0.721	0.7	64^3	64	7.75	20	0.821	10
(e)	0.279	0.721	0.7	64^3	96	26.16	20	0.821	10
(f)	0.279	0.721	0.7	64^3	128	62.01	20	0.821	10
(g)	0.279	0.721	0.7	80^3	80	7.75	20	0.821	10

4.1.1 Unidades de Medidas Utilizadas

As unidades de medidas estão em função do parâmetro de Hubble h, a massa do sol $M_\odot$ e o *parsec* .

- Massa em $10^{10}M_\odot/h$.
- Distâncias em Mpc/h.
- Velocidades em Km/s.
- Tempo normalizado em relação a um tempo de Hubble $1/H_0$.

- A energia foi normalizada dividindo-a pelo número de partículas e o valor da massa de cada partícula.

4.2 Comparação dos *Snapshots* do COLATUS e do Consórcio Virgo

Conforme explicado anteriormente, utilizamos as simulações de alta resolução em caixas pequenas, disponíveis no site `http://www.mpagarching.mpg.de/Virgo/data_download.html`. Na Figura 4.1, é apresentado a distribuição de partículas de uma simulação realizada pelo COLATUS no *redshifts* $z = 0$, utilizando os parâmetros (f) da Tabela 4.1.

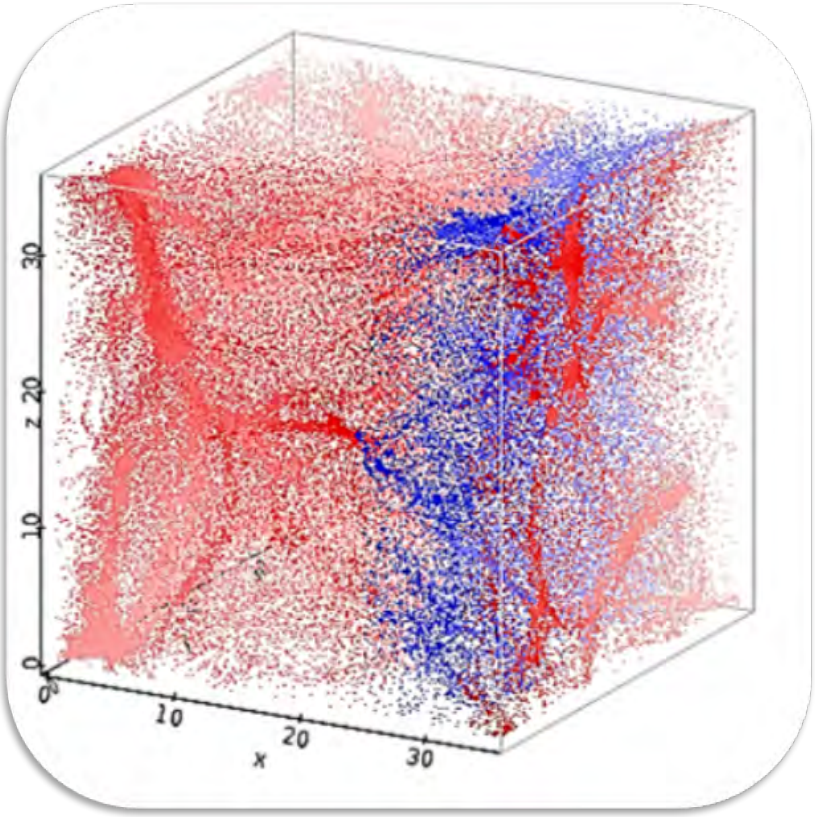


Figura 4.1 - Distribuição de partículas para $z = 0$. Em azul indicamos a seção de $8Mpc/h$ que foi utilizada para comparar visualmente com os resultados do VIRGO

O N-GENIC foi configurado para gerar as condições iniciais consistentes com o *redshift* $z = 30$, e o passo de tempo foi ajustado usando o valor de $\eta = 0,3$. Após 36 horas e 30 minutos e realizando 666 iterações obtivemos a sequência de *snapshots* mostrada na Figura 4.2.

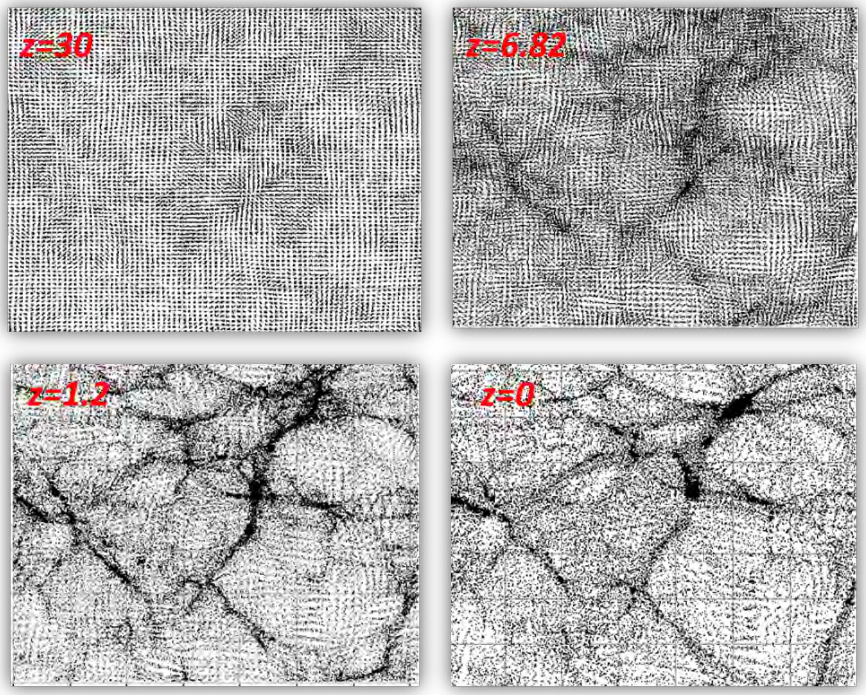


Figura 4.2 - *Snapshots* das simulações realizadas pelo COLATUS, $z = \{30.0, 6.82, 1.2, 0.0\}$ (amostra de 8x40x40 $(Mpc/h)^2$)

Finalmente, observamos que, visualmente a saída gerada pelo COLATUS (Figura 4.3, a) é cosmológicamente compativel com aquela gerada por simulações do Consórcio Virgo (Figura 4.3, b). A compatibilidade é verificada principalmente debido a formação de filamentos do tipo "grande parede".

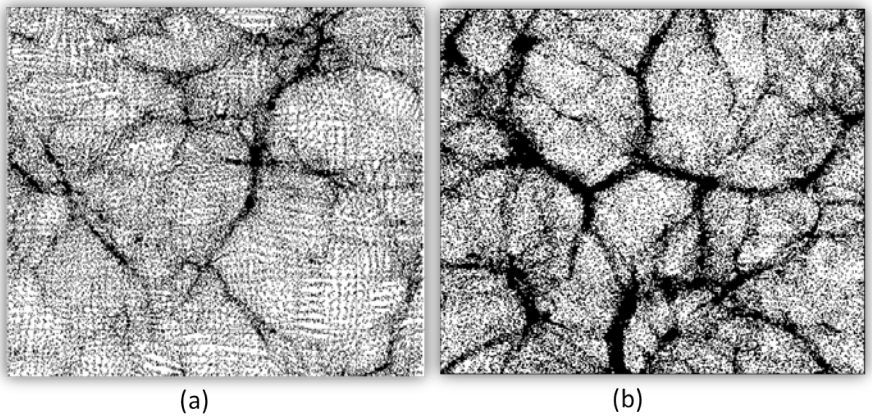


Figura 4.3 - *Snapshots* das simulações ΛCDM (a) Seção de $8Mpc/h$ de uma amostra de (40x40 $(Mpc/h)^2$), COLATUS, (b) Seção $8Mpc/h$ de uma amostra de (40x40) $(Mpc/h)^2$), VIRGO
Fonte: Kauffmann J.M. Colberg (1999)

4.3 Diferenças RMS entre as Posições e Velocidades das Implementações em GPU vs CPU

As simulações do COLATUS são realizadas utilizando variáveis de precisão dupla, consequentemente, aparecem erros de arredondamento e diferenças quando comparamos os resultados da CPU e da GPU. Conforme Delgado et al. (2011), Whitehead e FitFlorea (2011) isso ocorre por vários motivos: (i) as GPUs têm uma arquitetura diferente da CPUs, por exemplo, as GPUs têm unidades de multiplicação-adição e as CPUs comumente não; (ii) A paralelização do código pode mudar a ordem das operações; (iii) A implementação do standard IEEE 754 por parte das GPUs não garante que os resultados sejam idênticos, inclusive entre GPUs. As Figuras 4.4 e 4.5 mostram que as diferenças de resultado entre CPU e GPU não modifica significativamente a distribuição de matéria, nem as coordenadas de uma partícula ao longo das simulações.

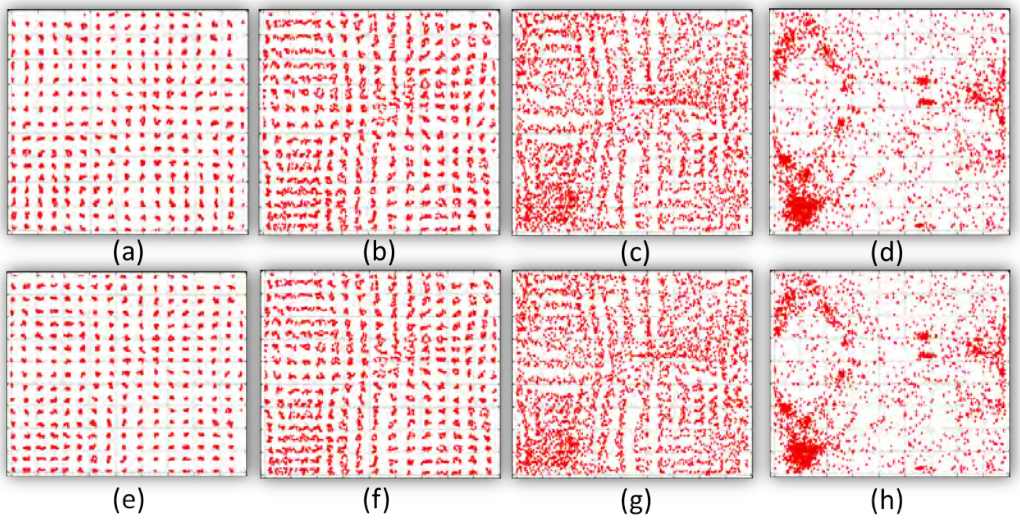


Figura 4.4 - $z = \{30.0, 10.0, 5.0, 0.0\}$, (a,b,c,d) em CPU e (e,f,g,h) em GPU($L = 8.8 Mpc/h$)

Essas diferenças de resultado entre a CPU e a GPU foram avaliadas quantitativamente, calculando as diferenças quadráticas médias entre as posições e velocidades

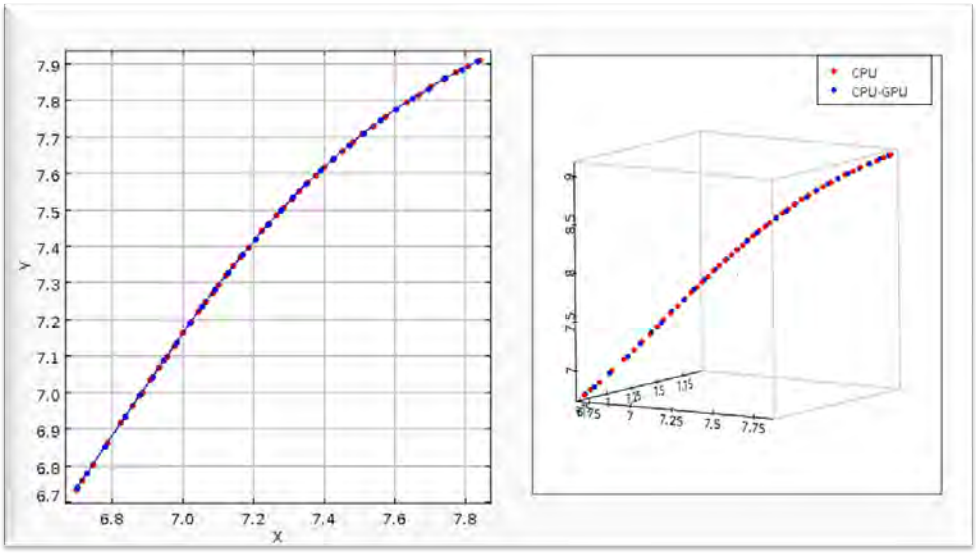


Figura 4.5 - Evolução da posição de uma partícula individual

das partículas correspondentes, da seguinte maneira:

$$\Delta x_{rms} = \left(\frac{\sum_{j=1}^{N} \left(\mathbf{x_j^{gpu}} - \mathbf{x_j^{cpu}} \right)^2}{N} \right)^{1/2}, \tag{4.1}$$

$$\Delta v_{rms} = \left(\frac{\sum_{j=1}^{N} \left(\mathbf{v_j^{gpu}} - \mathbf{v_j^{cpu}} \right)^2}{\sum_{j=1}^{N} \left(\mathbf{v_j^{cpu}} \right)^2} \right)^{1/2}. \tag{4.2}$$

A Tabela 4.2 apresenta os resultados das comparações considerando as mesmas condições iniciais e parâmetros das simulações.

Tabela 4.2 - Diferenças RMS entre as posições e velocidades correspondentes em função dos *redshifts*

Simulação	*Redshifs*	10.0	5.0	1.0	0.0
(a) $N = 16^3$,	Δx^{rms}(Mpc/h)	$5.590\ 10^{-7}$	$1.020\ 10^{-4}$	$7.873\ 10^{-3}$	$5.482\ 10^{-2}$
$L = 8.8$ (Mpc/h)	Δv_{rms}(km/s)	$9.140\ 10^{-9}$	$1.047\ 10^{-4}$	$2.381\ 10^{-3}$	5.38710^{-2}

4.4 Desempenho do *COLATUS_ENVIU*

Várias simulações foram realizadas com o objetivo de avaliar quantitativamente os ganhos obtidos com a utilização de computação híbrida com GPUs. Pela complexi-

dade do algoritmo utilizado, o tempo de computação aumenta de forma proporcional a N^2. Foram feitos experimentos para comparar o desempenho do algoritmo em GPUs, em relação ao desempenho em CPU.

As medidas mostradas na Tabela 4.3 incluem os tempos de transferência de dados da memória do *host* para a memória do *device* e vice-versa. Além de verificar o resultado dos cálculos executados, estes experimentos visaram estudar o comportamento dos tempos de processamento em relação a quantidade de partículas. Conforme as especificações da capacidade computacional (*compute capability*) das GPUs utilizadas GTX 580 e da GTX 680, adotou-se:

- número de *threads* por bloco $nthreadpblock = 512$(GTX 580), e 1024(GTX 680).
- número de blocos = $\lfloor N/nthreadpblock \rfloor$(número de partículas).

Para avaliar o desempenho, foram consideradas 100 iterações do algoritmo, o tempo foi medido a partir da vigésima iteração para considerar iterações onde o fluxo de dados já esta completamente estabelecido. A Tabela 4.3 apresenta os tempos obtidos. A partir dos resultados da Tabela 4.3, calculou-se o desempenho (*speed-up*) do

Tabela 4.3 - Tabela comparativa do desempenho

Partículas	Tempo da CPU	a Tempo da GPU	Tempo da GPU	Speed-up	Speed-up
	(seg)	(seg)	(seg)		
	Intel I7 950	GTX 580	GTX 680	GTX 580	GTX 680
	Quad-Core	(100passos)	(100passos)		
16^3	147.8	2.9	2.4	50.9	61.8
24^3	1614.9	18.5	26.5	87.3	60.9
32^3	9073.6	88.5	122.9	102.5	73.83
48^3	103354.0	917.9	1450.6	112.7	71.25
64^3	580710.5	4886.6	7655.7	118.7	75.85
80^3	2215234.8	19613.5		112.9	

uso da computação híbrida com GPU em relação a execução serial, o que verifica-se na Figura 4.6. O *speed-up* (S) é obtido pela razão entre o tempo gasto na execução serial e o tempo gasto na execução paralela, ou seja, $S = \frac{T_s}{T_p}$, onde T_s é o tempo sequencial e T_p é o tempo paralelo. Na Tabela 4.3 podemos observar um ganho de

desempenho de até 118.7 vezes para a GTX 580 em relação a implementação serial e de até 75.85 vezes para a GTX 680. Os experimentos na GTX 680 (Kepler) obtiveram desempenho menor que na GTX 580 (Fermi). Isso ocorreu porque todos os cálculos foram executados em precisão dupla e a arquitetura Kepler da placa GTX 680 não implementa essa funcionalidade. Nessa arquitetura, a precisão dupla é apenas emulada, o que diminui consideravelmente seu desempenho quando comparado ao desempenho da arquitetura Fermi.

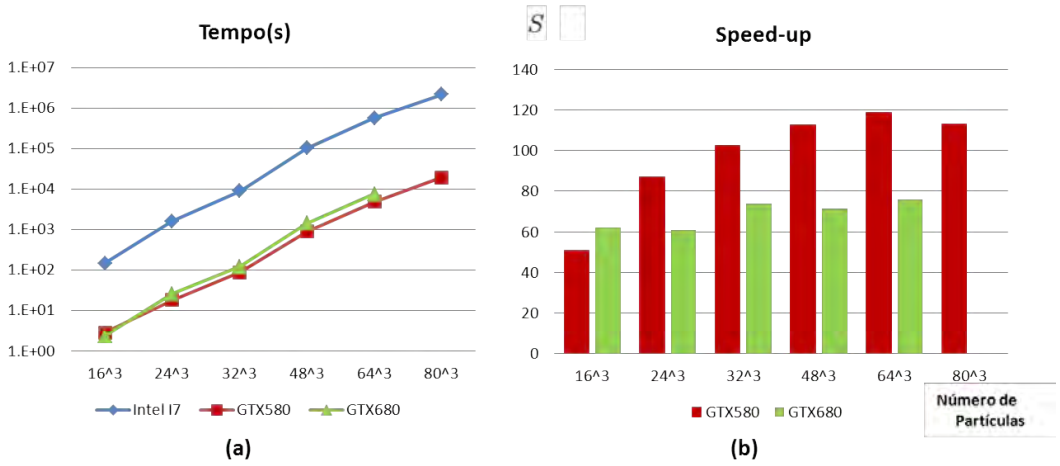


Figura 4.6 - Avaliação do desempenho: (a) Tempo de 100 iterações, (b) *Speedup*.

Finalmente, na Tabela 4.4 é mostrado o tempo total da execução, o tempo do cálculo da energia e o tempo gasto pela GPU para realizar as simulações.

Tabela 4.4 - Tempo de uma simulação completa

Partículas	Passos de tempo	Tempo do cálculo da energia (seg)	Tempo da GPU(seg)	Tempo Total (seg)
16^3	4331	0.77	118.45	119.22
24^3	1113	1.33	187.35	188.68
32^3	1372	2.47	1149.16	1151.63
48^3	572	274.09	5248.84	5522.93
64^3	595	492.55	29109.16	29601.71
80^3	664	1377.74	130233.57	131611,31

4.5 Testes de Validação

4.5.1 Conservação da Energia

O COLATUS oferece a possibilidade de utilizar três esquemas diferentes de cálculo do passo de tempo para integrar as equações do movimento:

- Passo de tempo fixo: Nesse caso, o passo de tempo é calculado a partir da idade do Universo. A idade do Universo depende do *redshifts* inicial (z_0) e final (z_f) e as equações de Friedmann, como segue:

$$T = \frac{1}{H_0}\int_{a_0}^{a_f} \frac{1}{\sqrt{\Omega_R\, a^{-4} + \Omega_M\, a^{-3} + \Omega_K\, a^{-2} + \Omega_\Lambda}}, \tag{4.3}$$

  em seguida a partir da quantidade máxima de passos de tempo, definida arbitrariamente, mas considerando que se for muito pequena a precisão será comprometida. Logo, o passo de tempo pode ser calculado como segue:

$$\Delta t = \frac{T}{H_0\, maxsteps}, \tag{4.4}$$

  lembrando que foi usado o tempo parametrizado pelo tempo de Hubble (conforme descrito na seção 2.2.4). O esquema de integração empregado é o seguinte:

$$\begin{aligned}
\mathbf{a_j^{n+1/2}} &= -B(t_n)\nabla_x\phi(\mathbf{x_j^{n+1/2}}) - \mathbf{2\,A(t_n)u^n} && (4.5)\\
\mathbf{u_j^{n+1}} &= \mathbf{u_j^n} + \frac{\mathbf{\Delta t}}{\mathbf{1 + A(t_n)}}\mathbf{a_j^{n+1/2}} && (4.6)\\
\mathbf{x_j}^{\mathbf{n+3/2}} &= \mathbf{x_j^{n+1/2}} + \mathbf{\Delta t\, u_j^{n+1}}. && (4.7)\\
t_{n+1} &= t_n + \Delta t, && (4.8)
\end{aligned}$$

  onde $A(t^n) = \frac{H(t_n)}{2}$ e $B(t^n) = \frac{1}{a^2}$. A simulação está concluida quando $t^{n+1} = T$.

- Passo de tempo variável: Nesse caso, o passo de tempo é calculado seguindo o critério apresentado na equação 2.23, que re-escrevemos para facilitar a leitura:

$$\Delta t = \sqrt{\frac{\eta}{a_{max}}} \quad \text{e} \quad \Delta t = \frac{\eta}{v_{max}} \tag{4.9}$$

- Passo de tempo variável usando o parâmetro $p = a^\alpha$: Nesse caso, o passo de

tempo é calculado seguindo o mesmo critério apresentado na equação 4.9 mas utilizando as acelerações e velocidades parametrizadas, isto é $a'_j = \frac{d\mathbf{v_j}}{dp}$ e $u'_j = \frac{d\mathbf{x_j}}{dp}$ respectivamente. Nota-se que a escolha do α afeta diretamente a precisão da integração.

Na Tabela 4.5, mostra-se a variação da energia em função da quantidade de iterações para cada esquema de integração mencionado. Observa-se na coluna 5 uma variação muito pequena da energia, ou seja, ela se conserva ao longo das iterações.

Tabela 4.5 - Conservação da energia para os diferentes passos de tempo.

Iteração	$(a\,T)$	(U)	$(\int a^2 T)$	(C')
Passo fixo				
0	0.01	-662.20	0.00	-662.05
230	21.204286	-679.222766	8.630142	-649.388337
520	89.473835	-761.764570	35.120438	-637.170297
730	134.341642	-827.805893	63.089865	-630.374386
1060	196.188700	-941.360140	113.546456	-631.624984
Passo variável				
0	0.01	-662.20	0.00	-662.05
220	12.585517	-669.817610	5.212858	-652.019236
550	91.750685	-764.136947	36.191764	-636.194498
740	120.095499	-802.122171	53.052509	-628.974164
1000	201.967282	-951.971773	116.614335	-633.390156
$\alpha = 0.4$				
0	0.01	-662.20	0.00	-662.05
250	0.290265	-661.831412	0.052694	-661.488453
560	1.394637	-661.273148	0.473690	-659.404822
750	4.080347	-661.884794	1.627884	-656.176563
1020	31.925900	-690.438600	12.605222	-645.907479
$\alpha = 0.66$				
0	0.01	-662.20	0.00	-662.05
280	0.462882	-661.669303	0.114679	-661.091742
550	2.052878	-661.186313	0.832615	-658.300820
730	10.207200	-668.471680	4.464287	-653.800193
1020	111.048703	-805.850049	49.485675	-645.315672
$\alpha = 0.8$				
0	0.01	-662.20	0.00	-662.05
250	0.685873	-661.472168	0.226033	-660.560262
540	5.367418	-664.049611	2.526163	-656.156029
750	40.420404	-716.685085	17.518776	-658.745905
892	150.264153	-932.781585	80.630346	-701.887086

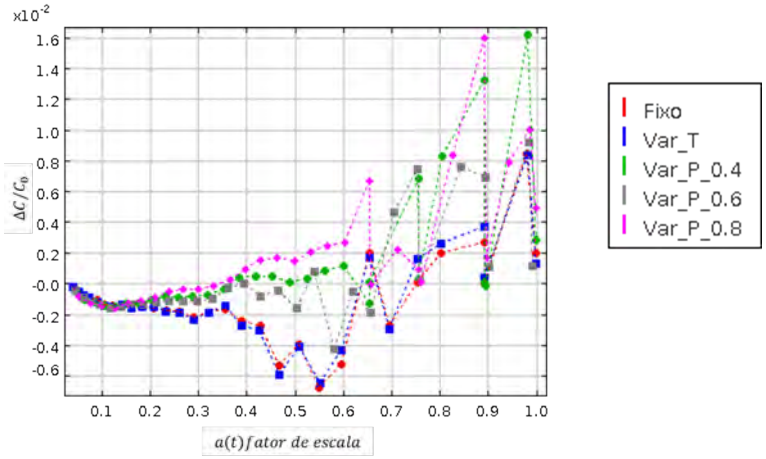


Figura 4.7 - Evolução de $\Delta C/C_0$ em função do fator de escala

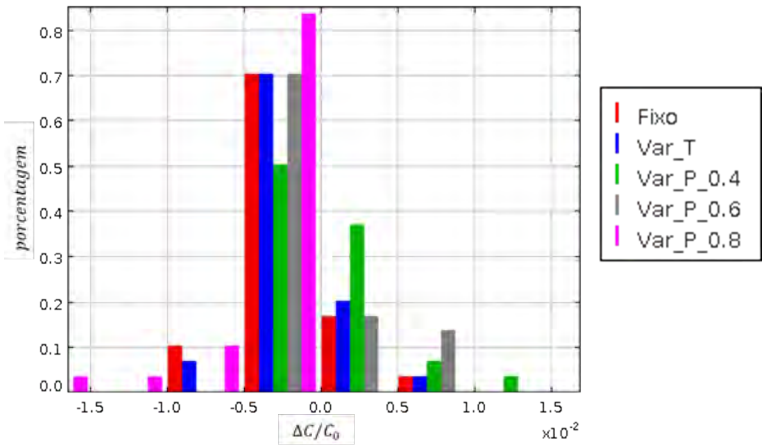


Figura 4.8 - Histograma do desvio $\Delta C/C_0$

As Figuras 4.7 e 4.8 mostram a variação percentual da energia total do sistema. O método de passo fixo conseguiu uma dispersão menor que 1%, mas é preciso considerar que ele demorou 2000 passos de tempos para chegar no *redshitfs* $z = 0$. Por essa razão ele não foi usado, já que os esquemas que usam passos de tempo variáveis demoram apenas cerca de 1000 passos de tempo para chegar até $z = 0$. Se comparamos os métodos de passo variável, o método que usa a variável temporal e o parâmetro $p = a^{\alpha}$, onde $\alpha = 0.66$, permitiram alcançar uma dispersão de energia menor que 1%.

4.5.2 Análise das Estruturas Geradas

Nessa seção, apresenta-se uma análise das estruturas geradas, considerando o *redshift* $z = 0$ e os parâmetros descritos nos itens c,d,e da Tabela 4.1.

Para identificar as estruturas que se formaram, utilizou-se o algoritmo FoF implementado por Huchra e Geller (1982) e Caretta et al. (2008). Essa aplicação também calcula as propriedades dos halos identificados, isto é, a massa total, posição, velocidade média e dispersão de velocidades dos halos.

A massa total M_j de cada halo é obtida somando-se a massa individual m_i de cada partícula contida no halo, isto é $M_j = \sum_{i=0}^{K_j} m_i$, onde K_j é o número de partículas contidas em cada halo. As posições ou centro de massa do halo são obtidas a partir da média das coordenadas das partículas do halo. Para obter a velocidade média $\overline{v}$, somam-se as velocidades de todas as partículas e divide-se pelo número total de partículas do halo, conforme $\overline{v}_j = \frac{1}{K_j} \sum_{i=0}^{K_j} v_i$. Finalmente, a dispersão de velocidades é obtida através do desvio padrão da velocidade das partículas de cada halo. O desvio padrão é dado por $S = \sqrt{\frac{1}{K_j - 1} \sum_{i=0}^{K_j} (v_i - \overline{v}_j)^2}$, onde $\overline{v_j}$ é a velocidade média.

Para identificar halos de aglomerados de galáxias, usualmente utiliza-se o valor de $b = 0.2$ (Mpc/h) e um número mínimo de partículas (LACEY; COLE, 1994). Para identificar os halos de galáxias, utilizou-se o algoritmo FoF com $b = 0.1$ juntamente com o critério de *boundedness* (CARETTA et al., 2008) para selecionar apenas os halos virializados, isto é, aqueles cuja velocidade de dispersão é proporcional a raiz quadrada da massa (número de partículas de cada halo) (ver Equação 2.31 e Figura 4.9).

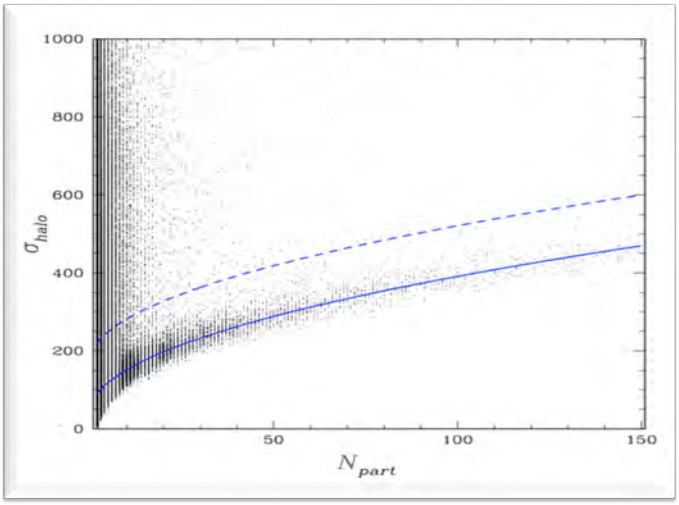


Figura 4.9 - Dispersão de velocidade dos halos de galáxias identificadas na simulação do Virgo.
Fonte: Caretta et al. (2008)

A Figura 4.10 mostra a dispersão de velocidade dos halos de galáxias em função do número de partículas dos halos identificados. Foram considerados como virializados os halos cujas velocidades de dispersão estão entre as curvas laranja e amarela. Essas curvas foram obtidas fazendo um ajuste de mínimos quadrados para uma função do tipo $\overline{v} = aK + b$. Com a resolução de massa da partícula ($1,4\ 10^{10} M\odot$) utilizada nas simulações, um halo de matéria escura de uma galáxia como a Via-Láctea ($M \approx 10^{12} h^{-1} M_{\odot}$), por exemplo, é obtida com aproximadamente 100 partículas.

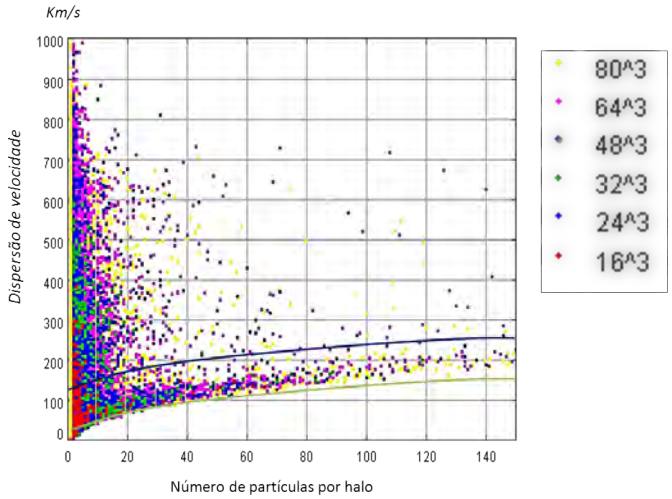


Figura 4.10 - Dispersão de velocidade dos halos de galáxias identificadas

Estes resultados são consistentes com os apresentados em Caretta et al. (2008), Ruiz (2011), consequentemente podemos dizer que COLATUS conseguiu simular o colapso gravitacional gerando estruturas virializadas de maneira similar as geradas pelas simulações do consórcio VIRGO que utilizaram o GADGET (KAUFFMANN J.M. COLBERG, 1999; SPRINGEL, 2005). Na Tabela 4.6, tem-se a quantidade total de halos de galáxias identificadas pelos algoritmo FoF, com número mínimo de partículas igual a dois e a quantidade desses halos com massas da ordem $10^{11} M_{\odot}$ e $10^{12} M_{\odot}$. Observa-se que aumentando o tamanho da caixa e o número de partículas, aumenta-se a quantidade de estruturas geradas.

Na Tabela 4.7, tem-se a quantidade total de halos de grupos/aglomerados identificadas pelo algoritmo FoF, com massa mínima $10^{13} M_{\odot}$ e máxima da ordem $10^{15} M_{\odot}$. Observa-se que foram encontrados poucos halos, provavelmente porque o volume e a quantidade de partículas utilizadas nas simulações realizadas são ainda pequenos.

Tabela 4.6 - Halos de galáxias identificadas

N	$L(Mpc/h)$	Halos de Galáxias	%Halos	%Halos
		0.1 Mpc/h	$M \simeq 10^{11} M_{\odot}$	$M \simeq 10^{12} M_{\odot}$
16^3	8.8	315	287	27
24^3	13.2	860	807	48
32^3	17.6	2321	2163	152
48^3	26.4	7518	7015	487
64^3	35.3	17582	16336	1215
80^3	44.1	30138	28924	1137

Tabela 4.7 - Halos de grupos/aglomerados identificadas

N	$L(Mpc/h)$	Halos de Aglomerados
		0.2 Mpc/h $M \approx 10^{15} M_{\odot}$
16^3	8.8	8
24^3	13.2	18
32^3	17.6	41
48^3	26.4	88
64^3	35.3	225
80^3	44.1	234

4.5.3 Função de Correlação de Dois Pontos

A função de correlação de dois pontos $\xi(r)$ para a distribuição de partículas foi calculada usando a aplicação desenvolvida por (ALONSO, 2012). Esta aplicação denomina*da Correlation Utilities and Two-point Estimation* (CUTE) esta disponível no site `http://members.ift.uam-csic.es/dmonge/CUTE.html`, e recebe como entrada um arquivo com as coordenadas de todas a partículas e calcula a função de correlação de dois pontos utilizando o estimador (LANDY; SZALAY, 1993).

A Figura 4.11 mostra como varia a função correlação de dois pontos, quando aumenta-se o número de partículas e o tamanho da caixa. Os resultados indicam que em escalas maiores a correlação aumenta, uma vez que, ao aumentar o tamanho da caixa e o número de partículas obtém-se estruturas maiores.

A Figura 4.12 mostra como varia a função correlação de dois pontos durante uma simulação (neste caso foi simulação (d) de 80^3 partículas).

Conforme se verifica na Figura 4.12, a função correlação aumenta a medida que o

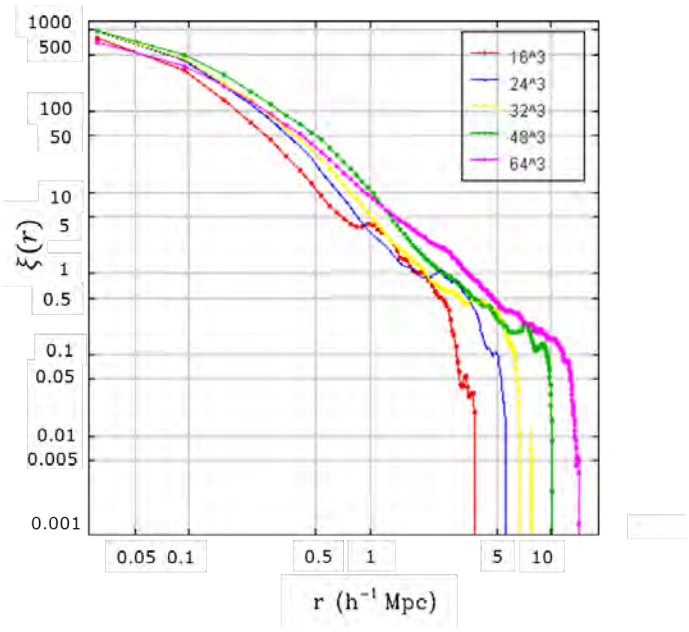


Figura 4.11 - Função de correlação em função da quantidade de partículas($z = 0$)

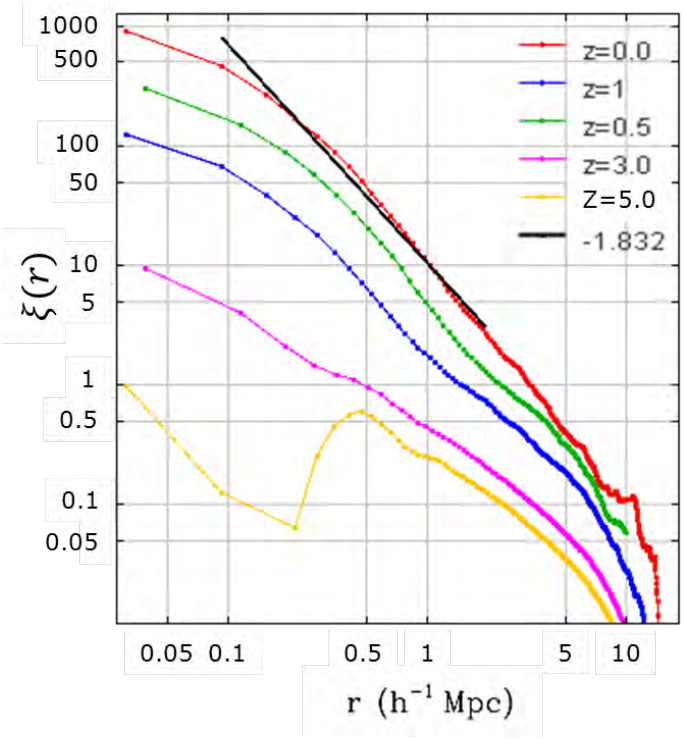


Figura 4.12 - Função de correlação em função do *redshift*

redshift tende a zero ($z = 0$), evidenciando assim a formação de estruturas. Esses resultados são similares aos obtidos por Caretta et al. (2008) (ver Figura 4.13).

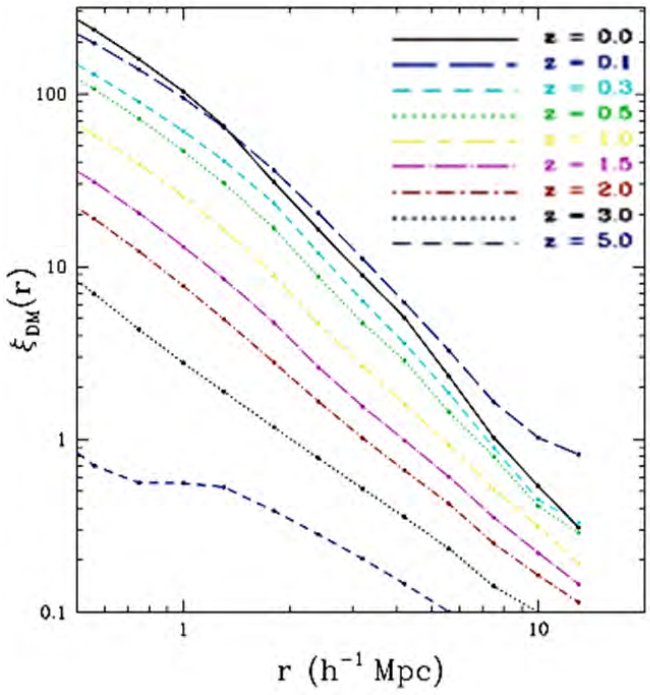


Figura 4.13 - Função de correlação em função do *redshift* da Simulação de Escala Intermediária do Virgo.
Fonte: Caretta et al. (2008)

4.6 Informações Extraídas da Dinâmica de Uma(s) Partícula(s)

O simulador COLATUS permite extrair informação (posição e velocidade) de um grupo de partículas em cada passo de tempo da simulação como é mostrado na Figura 4.14. Essa abordagem têm por objetivo de identificar padrões e processos análogos à turbulência a partir de uma análise estatística da dinâmica das partículas.

Taylor (1921), desenvolveu uma teoria estatística da turbulência. Essa teoria permite fazer uma caracterização da turbulência em fluidos. A idéia é marcar pares de partículas e observar como elas se afastam uma das outras. A medida que o processo dinâmico evolui, a distância entre elas inicialmente vai crescer linearmente com o tempo e, depois de um tempo suficientemente grande, essa distância é proporcional a raiz quadrada do tempo (Ver Apêndice B para mais detalhes).

A Figura 4.15 mostra a evolução das distâncias em função do tempo parametrizado $\tau = t/H_0$. Em seguida, essas distâncias foram normalizadas da seguinte maneira $d_n^i = \frac{d^i - d^i_{min}}{d^i_{max} - d^i_{min}}$, onde $i = \{1, \cdots, k\}$, k é ID da partícula considerada. Após isso, calculou-se a media $\overline{d}_n = \frac{1}{nk} \sum_{i=1}^{k} d_n^i$, onde nk é o número de partículas consideradas.

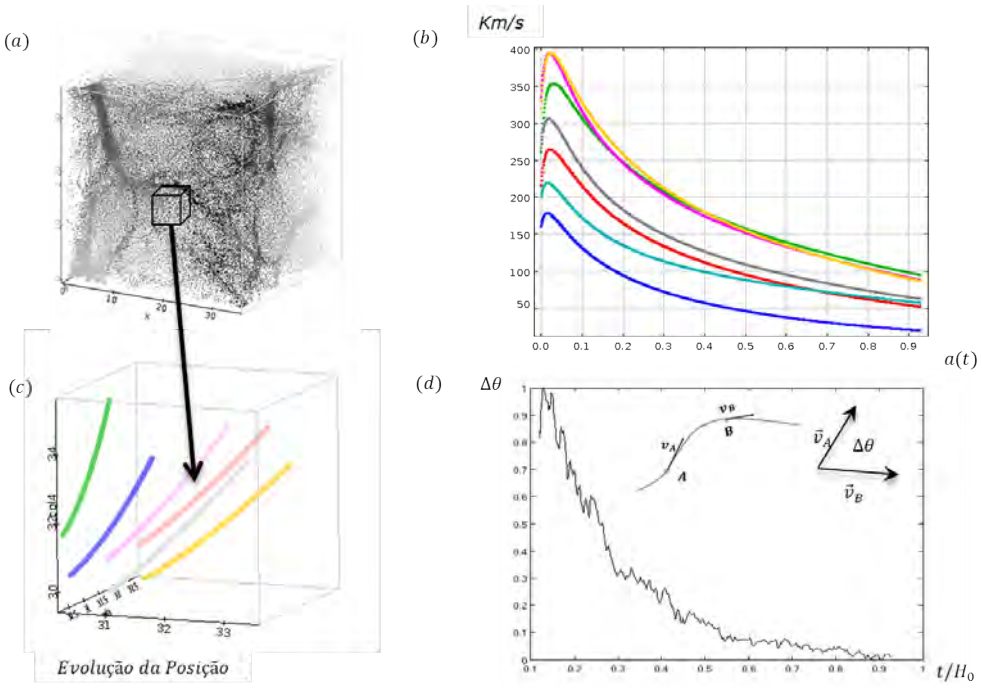


Figura 4.14 - (a) Seleção das partículas analisadas, (b) Velocidade das partículas analisadas em relação ao fator de escala, (c) Evolução da posição das partículas analisadas, (d) Variação angular das direções de uma partícula em relação ao tempo.

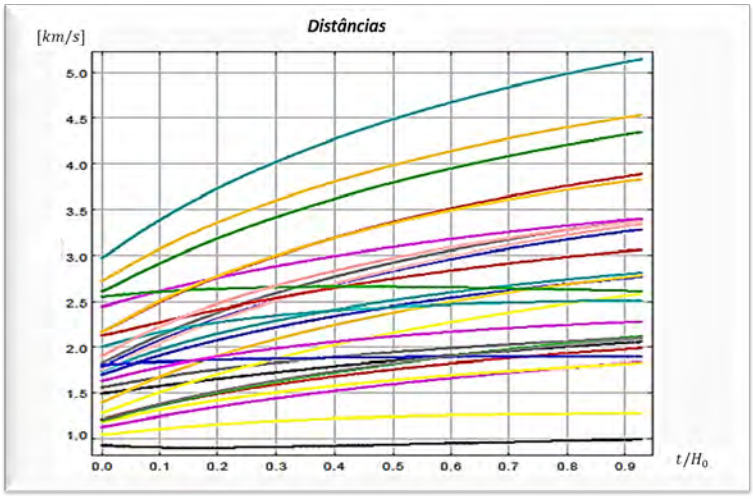


Figura 4.15 - Evolução das distâncias entre todas partículas

Finalmente ajustou-se pelos mínimos quadrados uma curva do tipo $\overline{d}_n = a\sqrt{\tau} + c$, resultando $a = 1.195\ (1.189, 1.202)$ e $c = -0.1453\ (-0.1494, -0.1413)$. Pode-se observar que os resultados exibidos nas Figuras 4.15 e 4.16 são consistentes com o comportamento previsto pela teoria de Taylor (Ver Apêndice B), sugerindo que a dinâmica das partículas durante o colapso gravitacional comporta-se de maneira análoga ao deslocamento instável imposto por um fluido turbulento (CARETTA et

al., 2008). Para uma caracterização fina deste fenômeno, o simulador permite extrair séries temporais da variação da direção angular de uma partícula durante o colapso.

O padrão de variabilidade observado no exemplo da Figura 4.17, evidencia a importância de incluir o traçador Lagrangeano para o estudo fino das instabilidades presentes na dinâmica de cada partícula ao longo da formação dos filamentos. Em especial, esse resultado abre uma nova perspectiva para o estudo de processos do tipo Advecção Caótica (ou Turbulência Lagrangeana) previstos a partir da análise de dados gerados com o GADGET2 (ROSA et al., 2009). Essa nova proposta analítica poderá proporcionar um novo método para validação de modelos em cosmologia. Principalmente para análise comparada entre modelos e dados observacionais de altíssima resolução previstos para os próximos anos.

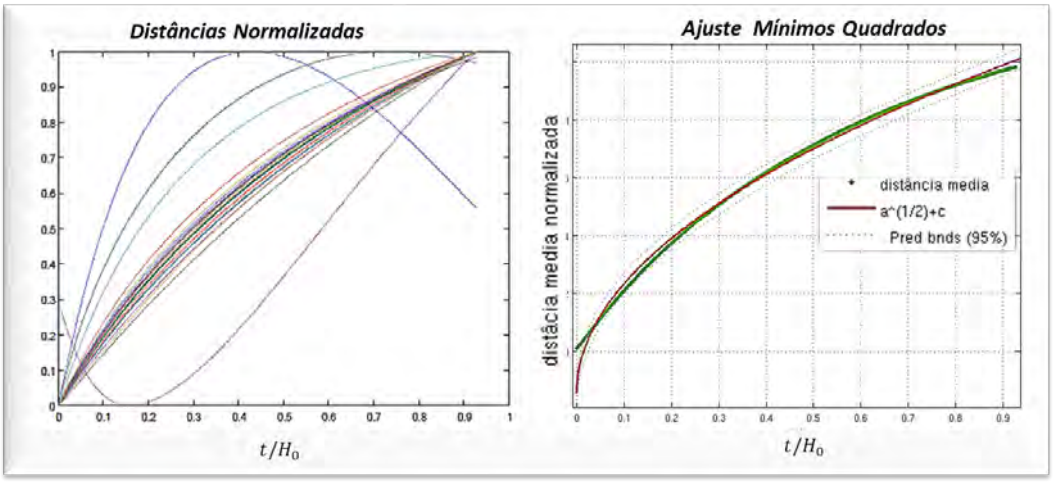


Figura 4.16 - (a)Evolução das distâncias entre todas partículas normalizadas, (b) Ajuste de curva.

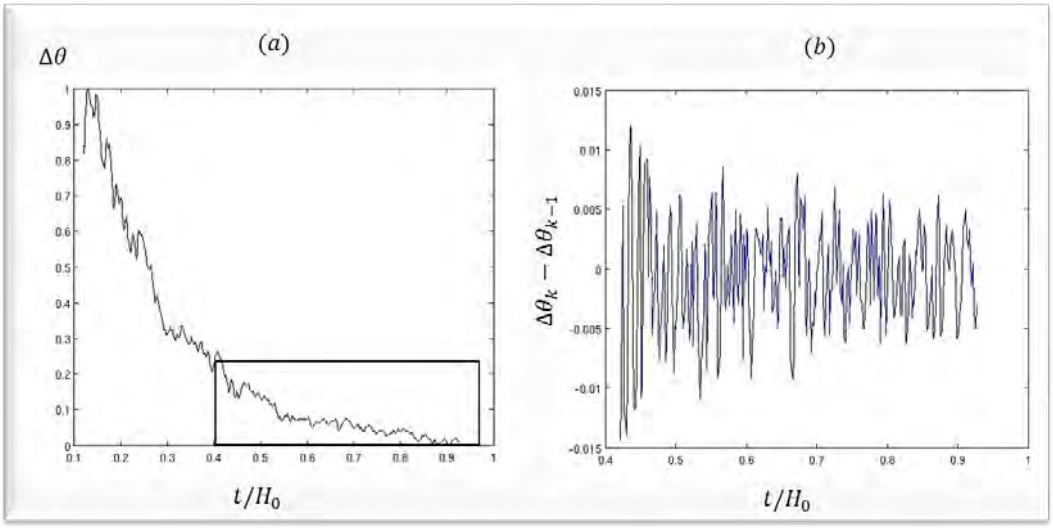


Figura 4.17 - (a)Variação angular das direções na forma normalizada, (b) Serie temporal das diferenças da variação angular normalizadas.

5 CONCLUSÃO

Neste trabalho, foi desenvolvido um novo simulador N-corpos (COLATUS ENVIU) utilizando uma abordagem em computação híbrida baseada em GPUs. A implementação do código foi realizada usando a linguagem de programação C/C++ e a plataforma CUDA. Com o objetivo de analisar o desempenho da versão CUDA, foi implementado também uma versão serial do algoritmo. A comparação de desempenho de ambas versões (serial e paralela) mostrou um ganho considerável para a versão em CUDA (118 vezes mais rápida com 64^3 partículas). No entanto, pode se observar uma leve diminuição no *speed-up* a medida que o número de partículas aumenta (112 para 80^3 contra 118 para 64^3). Isso ocorre porque a GPU possui uma capacidade limitada para processar vários *threads* de forma concorrente e adotou-se um esquema de paralelização onde a cada *thread* é atribuído uma partícula. Com o aumento de partículas na simulação, o conjunto total é dividido em vários subconjuntos de *threads* que serão executados sequencialmente. Uma alternativa para melhorar o desempenho seria atribuir várias partículas a um mesmo *thread*.

As diferenças entre a arquitetura da CPU e da GPU podem gerar pequenas variações nas posições e velocidades obtidas pela GPU. Dessa forma, essas variações foram avaliadas e encontrou-se que não alteram significativamente a distribuição e a dinâmica das partículas.

Na integração das equações do movimento, foram implementados três esquemas de cálculo de passo de tempo: passo fixo, passo variável em t e passo variável em p. Todos esses esquemas permitiram avaliar a condição de Layzer-Irvine de forma satisfatória, porém com diferentes números de iterações para se chegar ao *redshift* $z = 0$. O esquema de passo variável em p com $\alpha = 0.66$ foi o que utilizou um menor número de iterações.

Em uma primeira análise das estruturas geradas, podemos observar que o simulador foi capaz de reproduzir padrões da teia cósmica observada no Universo. Utilizando a técnica de agrupamento *Friends of Friends* foi possível identificar estruturas como halos de galáxias e de aglomerados de galáxias. Por meio da análise do Virial, identificou-se estruturas ligadas gravitacionalmente e em equilíbrio dinâmico.

Para a amostra de partículas simuladas, realizou-se também uma análise estatística por meio da função de correlação de dois pontos. Os resultados obtidos foram similares aos obtidos por Caretta et al. (2008) para a Simulação em Escala Intermediária do consorcio Virgo.

Com a inclusão do traçador Lagrangeano (Figura 4.17), concluímos que todos os objetivos previstos no Capitulo 1 desta dissertação foram satisfatoriamente alcançados. Dessa forma, o simulador *COLATUS_ENVIU* está disponível para as primeiras pesquisas independentes de cosmologia computacional para pesquisadores do INPE e de outros grupos nacionais e internacionais.

5.1 Trabalhos Futuros

Como trabalhos futuros propomos:

- Realizar simulações em um volume maior, considerando uma melhor resolução de massa e avaliar as estruturas formadas também por meio da Função de Massa e do Espectro de Potência.
- Incorporar um gerador de condições iniciais baseadas na teoria de perturbações de segunda ordem (SCOCCIMARRO et al., 2012).
- Implementar métodos mais eficientes para calcular a força e considerar também a hidrodinâmica para incluir o gás.
- Adaptar o código para possibilitar a execução com várias GPUs.
- Desenvolver uma nova versão adequada para a arquitetura Kepler, visando aproveitar seu paralelismo dinâmico.
- Otimizar o código utilizando a memória compartilhada, para diminuir o acesso a memória global.
- Considerar um conjunto maior de partículas para fazer a análise do Teorema de Taylor.
- Gerar simulações, introduzindo potenciais de deformação, para testar as hipóteses alternativas do Projeto Cusp Cosmology (ver Apêndice B).
- Incorporar métodos para analisar o movimento das galáxias considerando medidas de velocidades radiais observadas em aglomerados (CAPELATO et al., 1991).

REFERÊNCIAS BIBLIOGRÁFICAS

AARSETH, S. J. Direct methods for n-body simulations. In: BRACKBILL, J. U.; COHEN, B. I. (Ed.). **Multiple time scales, p. 377 - 418**. [S.l.: s.n.], 1985. p. 377–418. 2

_____. **Gravitation N-Body Simulation**: Tools and algorithms. New York, USA: Cambridge Unversity Press, 2003. 431 p. 9, 12, 15

ALONSO, D. **CUTE solutions for two-point correlation functions from large datasets**. 2012. 8 p. Disponível em: <astro-ph/1210.1833>. Acesso em: 21-01-2013. 55

APPEL, A. Merger rates in hierarchical models of galaxy formation - part two - comparison with n-body simulations. **SIAM J. Sci. Stat. Comput.**, v. 6, p. 85, 1985. 2

BAGLA, J. S. Cosmological n-body simulation: Techniques, scope and status. **Current Science**, v. 88, p. 1088, 2008. 13, 28

BARNES, J.; HUT, P. A hierarchical o(n log n) force-calculation algorithm. **Nature**, v. 324, p. 446–449, 1986. 2, 13

BERTSCHINGER, E. Simulations of structure formation in the universe. **Annual Review Astronomy Astrophysics**, v. 36, p. 599–654, 1998. 3, 9, 12, 13, 17, 28

BINNEY J.; TREMAINE, S. **Galactic Dynamics**: Princeton series in astrophysics. Princeton, USA: Princeton University Press, 2003. 885 p. 5

BODNHEIMER, P.; LAUGHLIN, G. P.; YORKE, H. W. **Numerical Methods in Astrophysics**: An introduction. New York, USA: Taylor and Francis, 2007. 329 p. 12, 15

BOUTE, R. T. The euclidean definition of the functions div and mod. **ACM Trans. Program. Lang. Syst.**, New York, USA, v. 14, n. 2, p. 127–144, 1992. 39

BRODTKORB, A. R.; DYKEN, C.; HAGEN, T. R.; HJELMERVIK1, J. M.; STORAASLI, O. O. State-of-the-art in heterogeneous computing. **Scientific Programming**, v. 18, p. 1–33, 2010. 84

CAPELATO, H. V.; MAZURE, A.; PROUST, D.; VANDERRIEST, C.; LEMONNIER, J. P.; SODRE JR., L. New measurements of radial velocities in

clusters of galaxies. iii. **Astronomy and Astrophysics Supplement Series**, v. 90, p. 355–364, oct 1991. 62

CARETTA, C. A.; ROSA, R. R.; VELHO, H. F. C.; RAMOS, F. M.; MAKLER., M. Evidence of turbulence-like universality in the formation of galaxy-sized dark matter haloes. **Astron. Astrophys**, v. 48, 2008. 17, 53, 54, 56, 57, 59, 61, 77

CHABRIER, G. **Structure Formation in Astrophysics**. New York, USA: Cambridge University Press, 2009. 456 p. 7

DAVIS, M.; PEEBLES, P. A survey of galaxy redshifts. v - the two-point position and velocity correlations. **Astrophysical Journal**, v. 267, p. 465–482, 1983. 19

DEKEL A.AND OSTRIKER, J. **Formation of Structure in the Universe**. Cambridge, UK: Cambridge University Press, 1999. 470 p. 1

DELGADO, J.; GAZOLLA, J.; CLUA, E.; SADJADI, S. M. A case study on porting scientific applications to gpu/cuda. v. 2, p. 3–11, 2011. 46

D'INVERNO, R. **Introducing Einstein's Relativity**. Oxford, UK: Clarendon Press, 1992. 383 p. 82

DOROUD, N.; SMOLIN, L. An action for higher spin gauge theory in four dimensions. **Europhysics Letters (Print)**, p. 28, 2011. Disponível em: <arXiv:1102.3297v1>. 82

EFSTATHIOU G.; DAVIS, M. W. S. D. M. F. C. S. Numerical techniques for large cosmological n-body simulations. **Astrophysical Journal Supplement Series**, v. 57, p. 241–260, 1985. 14, 15, 28, 39

EINSTEIN, A. Kosmologische betrachtungen zur allgemeinen relativit tstheorie. **Sitzungsberichte der Preuischen Akademie der Wissenschaften**, p. 142, 1917. 71

EISENSTEIN, D. J.; HU, W. Baryonic features in the matter transfer function. **The Astrophysical Journal**, v. 496, n. 2, p. 605, 1998. Disponível em: <http://stacks.iop.org/0004-637X/496/i=2/a=605>. 28

ELSEN, E.; HOUSTON, M.; VISHAL, V.; DARVE, E.; HANRAHAN, P.; PANDE, V. N-Body Simulation on GPUs. In: **Poster: SC06 International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2006. 2, 23

FLYNN, M. Some computer organizations and their effectiveness. **IEEE Trans. Comput.**, C-21, p. 948, 1972. 83

FRIEDMAN, A. Uber die krummung des raumes. **Zeitschrift fur Physik**, v. 10, p. 1:377–386, 1922. 71

GABUROV, E.; HARFST, S.; ZWART, S. P. Sapporo: A way to turn your graphics cards into a grape-6. **Instrumentation and Methods for Astrophysics**, 2009. Disponível em: <http://arxiv.org/abs/0902.4463v2>. 24

GINGOLD, R. A.; MONAGHAN, J. J. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. **Monthly Notices of the Royal Astronomical Society**, v. 181, p. 375–389, nov. 1977. 3

GREEN, A. D. **Cosmological Application of Multigrid Methods**. 155 p. p. Tese (Phd Thesis) — University of Oxford, St. Peterś College, 2000. 8, 15

GUTH, A. H. A possible solution to the horizon and flatness problems. **Physical Review D**, v. 23, p. 347–356, 1981. 73

HAMADA, T.; NARUMI, T.; YOKOTA, R.; YASUOKA, K.; NITADORI, K.; TAIJI, M. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In: **Proceedings of the Conference on High Performance Computing Networking,Storage and Analysis**. New York, NY, USA: ACM, 2009. (SC 09), p. 62:1–12. 23, 24

HAMILTON, A. J. S. Toward better ways to measure the galaxy correlation function. **Astrophysical Journal**, v. 417, p. 19, 1993. 19

HINSHAW, G.; LARSON, D.; KOMATSU, E.; SPERGEL, D. N.; BENNETT, J. D. C. L.; NOLTA, M. R.; HALPERN, M.; HILL, R. S.; ODEGARD, N.; PAGE K. M. SMITHAND J. L. WEILANDAND B. GOLD, N. J. L.; KOGUT, A.; LIMON, M.; MEYER, S. S.; TUCKER, G. S.; WOLLACK, E.; WRIGHT, E. L. Nine-year wilkinson microwave anisotropy probe (wmap) observations: Cosmological parameter results. **The Astrophysical Journal Supplement Series, Submited**, p. 31, 2012. Disponível em: <arXiv:1212.5226v1>. 11, 28, 71, 72, 73, 80, 97

HOCKNEY, R. W.; EASTWOOD, J. W. **Computer Simulation Using Particules**. New York, USA: Adam Hilger, Bistol and New York, 1988. 431 p. 12, 14, 16

HOLMBERG, E. On the clustering tendencies among the nebulae. ii. a study of encounters between laboratory models of stellar systems by a new integration procedure. **Astrophysical Journal**, v. 94, p. 385, 1941. 2

HUBBLE, E. A relation between distance and radial velocity among extra-galactic nebulae. **Proceedings of the National Academy of Sciences**, v. 15, p. 3:167, 1929. 71, 72, 73

HUCHRA, J. P.; GELLER, M. J. Groups of galaxies. i - nearby groups. **Astrophysical Journal**, v. 257, p. 423–437, 1982. 17, 53

J.S.BAGLA; RAY, S. Comments on the size of the simulation box in cosmological n-body simulations. **Mon.Not.Roy.Astron.Soc. 358**, v. 358, p. 1076–1082, 2005. 28

JUNIOR, J. Ricardo da S.; CLUA, E. W. G.; MONTENEGRO, A.; LAGE, M.; DREUX, M. d. A.; JOSELLI, M.; PAGLIOSA, P. A.; KURYLA, C. L. A heterogeneous system based on gpu and multi-core cpu for real-time fluid and rigid body simulation. **International Journal of Computational Fluid Dynamics**, v. 26, n. 3, p. 193–204, 2012. 21

KAUFFMANN J.M. COLBERG, A. D. S. W. G. Clustering of galaxies in a hierarchical universe: I. methods and results at z=0. **Mon. Not. R. Astron. Soc.**, v. 303, p. 188–206, 1999. 28, 43, 45, 54

KERSCHER, M.; SZAPUDI, I.; SZALAY, A. S. A comparison of estimators for the two-point correlation function. **Astrophysical Journal**, v. 535, p. L13–L16, 2000. 20

KIRK, D.; HWU, W.-M. **Programando para Processadores Paralelos**: Uma abordagem prática da programação de gpu. Rio de Janeiro, BR: ElSevier, 2011. 204 p. p. 22, 23, 84, 85, 87

KROUPA, P.; FAMAEY, B.; BOER, K. S. de; DABRINGHAUSEN, J.; PAWLOWSKI, M. S.; BOILY, C. M.; JERJEN, H.; FORBES, D.; HENSLER, G.; METZ, M. Local-group tests of dark-matter concordance cosmology . towards a new paradigm for structure formation. **Astrophysical Journal**, v. 523, p. 22, 2010. 79

LACEY, C.; COLE, S. Merger rates in hierarchical models of galaxy formation - part two - comparison with n-body simulations. **Monthly Notices of the Royal Astronomical Society,**, v. 271, p. 676, 1994. 18, 53

LANDY, S.; SZALAY, A. Bias and variance of angular correlation functions. **Astrophysical Journal**, v. 412, p. 64–71, 1993. 19, 20, 55

LEMAITRIE, G. Un univers homogene de masse constante et de rayon croissant rendant compte de la vitesse radiale des nubuleuses extra-galactique. **Annales de la Societe Scientifique de Bruxelles**, A47, p. 49–59, 1927. 71

LIDDLE, A. **An Introduction to Modern Cosmology**. West Sussex, UK: John Wiley and Songs Ltd., 2003. 189 p. 7

LUCY, L. B. A numerical approach to the testing of the fission hypothesis. **Astronomical Journal**, v. 82, p. 1013–1024, dez. 1977. 3

MCCREA, W. H.; MILNE, E. A. Newtonian universes and the curvature of space. **The quarterly journal of mathematics**, 1934. 8

MO, H.; VANDENBOSCH, F.; WHITE, S. **Galaxy Formation and Struture**. New York, USA: Cambridge University Press, 2010. 842 p. 3, 7, 8, 10, 12, 72, 73, 74

NITADORI, K.; AARSETH, S. J. Accelerating nbody6 with graphics processing units. **Monthly Notices of the Royal Astronomical Society**, v. 424, n. 1, p. 545–552, 2012. 2

NVIDIA. Whitepaper, **White Paper NVIDIAs Next Generation CUDA Compute Architecture: Kepler GK110.** 2012. 24 p. p. Disponível em: <http://www.nvidia.com/object/cuda_home_new.html>. Acesso em: 14-jan-2013. 88, 89, 90, 91, 92

NYLAND, L.; HARRIS, M.; PRINS, J. Fast n-body simulation with cuda. In: NGUYEN, H. (Ed.). **GPU Gems 3**. [S.l.]: Addison Wesley Professional, 2007. cap. 31. 2, 23

OWENS, J. D.; LUEBKE, D.; GOVINDARAJU, N.; HARRIS, M.; KRüGER, J.; LEFOHN, A. E.; ; PURCELL, T. J. A survey of general-purpose computation on graphics hardware. **The European Association for Computer Graphics) and Blackwell Publishing.**, v. 27, p. 80–113, 2007. 85, 88

PEEBLES, P. J. E. **The Large-Scale Structure of the Universe**. New Jersey, USA: Princeton University Press, 1980. 8, 19

PEEBLES, P. J. E.; HAUSER, M. G. Statistical analysis of catalogs of extragalactic objects. iii. the shane-wirtanen and zwicky catalogs. **Astrophysical Journal Supplement**, v. 28, p. 19, nov 1974. 19

PERLMUTTER, S.; ALDERING, G.; VALLE, M. della; DEUSTUA, S.; ELLIS, R. S.; FABBRO, S.; FRUCHTER, A.; GOLDHABER, G.; GROOM, D. E.; HOOK, I. M.; KIM, A. G.; KIM, M. Y.; KNOP, R. A.; LIDMAN, C.; MCMAHON, R. G.; NUGENT, P.; PAIN, R.; PANAGIA, N.; PENNYPACKER, C. R.; RUIZ-LAPUENTE, P.; SCHAEFER, B.; WALTON, N. Discovery of a supernova explosion at half the age of the universe. **Nature**, v. 391, p. 51–54, 1998. 71, 73

PRESS, W. H.; SCHECHTER, P. Formation of galaxies and clusters of galaxies by self-similar gravitational condensation. **Astrophysical Journal**, v. 187, p. 425–438, 1974. 2

RIESS, A. G.; FILIPPENKO, A. V.; CHALLIS, P.; CLOCCHIATTI, A.; DIERCKS, A.; GARNAVICH, P. M.; GILLILAND, R. L.; HOGAN, C. J.; JHA, S.; KIRSHNER, R. P.; LEIBUNDGUT, B.; PHILLIPS, M. M.; REISS, D.; SCHMIDT, B. P.; SCHOMMER, R. A.; SMITH, R. C.; SPYROMILIO, J.; STUBBS, C.; SUNTZEFF, N. B.; ; TONRY, J. Observational evidence from supernovae for an accelerating universe and cosmological constant. **The astronomical Journal**, v. 116, p. 1009–1038, 1998. 71

ROSA, R. An alternative coupled expanding universe without cosmological singularity. In: **38th COSPAR Scientific Assembly**. [S.l.: s.n.], 2010. v. 38, p. 3663. 81

ROSA, R. R. Apostila, **Tópicos de Cosmologia Computacional**. São Jose dos Campos, Brasil: [s.n.], 2012. 18 p. 7

ROSA, R. R.; RAMOS, F. M.; CARETTA, C. A.; VELHO, H. F. C. Extreme event dynamics in the formation of galaxy-sized dark matter structures. **Computer Physics Communications**, v. 4, p. 180:621–624, 2009. 59, 78, 79

RUIZ, R. S. d. R. **Turbulência em cosmologia: análise de dados simulados e observacionais usando computação de alto desempenho**. 145 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011-05-26 2011. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m19/2011/04.27.19.23>. Acesso em: 19 abr. 2013. 17, 54

SCOCCIMARRO, R.; HUI, L.; MANERA, M.; CHAN, K. C. Large-scale bias and efficient generation of initial conditions for nonlocal primordial non-gaussianity. **Phys. Rev. D**, American Physical Society, v. 85, p. 083002, Apr 2012. Disponível em: <http://link.aps.org/doi/10.1103/PhysRevD.85.083002>. 62

SPRINGEL, V. The cosmological simulation code gadget-2. **Mon. Not. R. Astron. Soc.**, v. 3, p. 31, 2005. 5, 54

SPRINGEL, V.; WHIT, S.; JENKING, A.; FRENK, C.; YHOSIDA, N.; GAO, L.; NAVARRO, J.; THACKER, R.; CROTON, D.; HELLY, J.; PEACOCK, J. A.; COLE, S.; THOMAS, P.; COUCHMAN, H.; EVARARD, A.; COLBERG, J.; PEARCE, F. Simulating the joint evolution of quasars, galaxies and their large-scale distribution. **Nature**, v. 435, p. 629, 2005. 17

SPRINGEL, V.; YOSHIDA, N.; WHITE, S. D. M. Gadget: a code for collisionless and gasdynamical cosmological simulations. **New Astronomy**, v. 6, p. 79–117, apr 2001. 3, 5

STALDER, D. H.; ROSA, R. R.; JUNIOR, J. R. da S.; CLUA, E.; RUIZ, R. S. R.; VELHO, H. F. C.; RAMOS, F. M.; ARAUJO, A. S. d. S.; CONRADO, V. G. A new gravitational n-body simulation algorithm for investigation of cosmological chaotic advection. In: INTERNATIONAL SCHOOL ON FIELD THEORY AND GRAVITATION, 6., 23-27 apr. 2012, Petropolis. **Proceedings...** [S.l.], 2012. AIP CONF PROC, 1483, p. 447 452. ISBN 0094-243X. ISSN 978-0-7354-1095-4. and 10.1063/1.4756992. Acesso em: 19 abr. 2013. 2

STEPHANI, H.; KRAMER, D.; MACCALLUM, M.; HOENSELAERS, C.; HERLT, E. **Exact Solution to Einstein's Field Equations**. Cambridge, UK: Cambridge University Press, 2003. 80

TAYLOR, G. I. Diffusion by continuos movements. **Proceedings of London Mahtematical Society**, v. 20, p. 193–211, 1921. 57, 77, 78

VELHO, H. F. de C. **Modelagem Matemática em Tubulência Atmosférica**. São Carlos-SP, Brasil: Sociedade Brasileira de Matemática Aplicada e Computacional, 2010. 77

WEINBERG, S. **Cosmology**. Oxford, UK: Oxford Unversity Press, 2008. 612 p. 74

WHITE, S. D. M. The dynamics of rich clusters of galaxies. **Monthly Notices of the Royal Astronomical Society**, v. 177, p. 717–733, 1976. 2

WHITEHEAD, N.; FITFLOREA, A. Percision and performance floating point and ieee 754 compliance for nvidia gpus. p. 7, 2011. Disponível em: <http://developer.nvidia.com/sites/default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf>. 46

ZEL'DOVICH, Y. B. Gravitational instability: An approximate theory for large density perturbations. **Astronomy and Astrophysics**, v. 5, p. 84–89, mar. 1970. 11

ZWICKY, F. On the masses of nebulae and of clusters of nebulae. **Astrophysical Journal**, v. 86, p. 217, 1937. 71

APÊNDICE A - FUNDAMENTOS DE COSMOLOGIA

A.1 Breve Histórico da Cosmologia

A teoria da gravitação universal (Newton, 1687) e a Teoria da Relatividade Geral (TRG) (EINSTEIN, 1917) forneceram as ferramentas que consolidaram a cosmologia moderna no início do século XX, impulsionando grandes projetos de observações astronômicas, em escalas cosmológicas no século XXI. Na TRG, Einstein introduziu a constante cosmológica Λ para compensar os efeitos da gravitação. Sem este ingrediente o universo colapsaria sobre si mesmo.

Em 1922, Friedmann construiu um modelo do universo em expansão sem a constante cosmológica (FRIEDMAN, 1922). De forma independente, George Lemaître em 1927 também mostrou que o balanço entre a força de atração gravitacional e uma força repulsiva (constante cosmológica) poderia dar origem a um Universo em contração ou em expansão (LEMAITRIE, 1927). Em 1929 as observações realizadas por Edwin Hubble e colaboradores consolidaram o modelo expansionista mediante a observação da expansão linear de 24 galáxias relacionando o desvio para o vermelho (*redshifts*) com a distância. Em um Universo descrito pelo espaço-tempo em expansão, como mostrado na Figura A.1 quanto maior é a distância entre duas ou mais galáxias, maior o desvio para o vermelho na relação mútua entre cada par. Consequentemente, como mostrado na Figura A.1, a velocidade de afastamento cresce com a distância (HUBBLE, 1929). O afastamento das galáxias observado por Vesto Slipher, Edwin Hubble e Milton Humason, sugeriu para a comunidade científica que no passado elas deveriam estar mais próximas. Estimações recentes indicam que aproximadamente 13,7 bilhões de anos atrás (HINSHAW et al., 2012) todas as galáxias deveriam estar num mesmo ponto, uma singularidade espaço-tempo que se expandiu (esse processo foi denominado *Big Bang* por Fred Hoyle).

Em 1998, o resultado da análise minuciosa das observações sistemáticas de supernovas tipo Ia indicaram que além do Universo se encontrar em expansão, essa se dá de forma acelerada. Os cosmólogos explicam a expansão acelerada como resultado do aumento de um tipo de energia extra anti-gravitacional no Universo a *energia escura*. Essa energia, que compõe cerca de 73% da densidade de energia do Universo, é necessária para explicar a taxa não-linear de expansão, interpretada no modelo como sendo a constante cosmológica Λ (PERLMUTTER et al., 1998; RIESS et al., 1998).

Outro ingrediente muito discutido e estudado é a *matéria escura*, ela surgiu a partir da observação das curvas de rotação das galáxias (ZWICKY, 1937). Fritz Zwicky

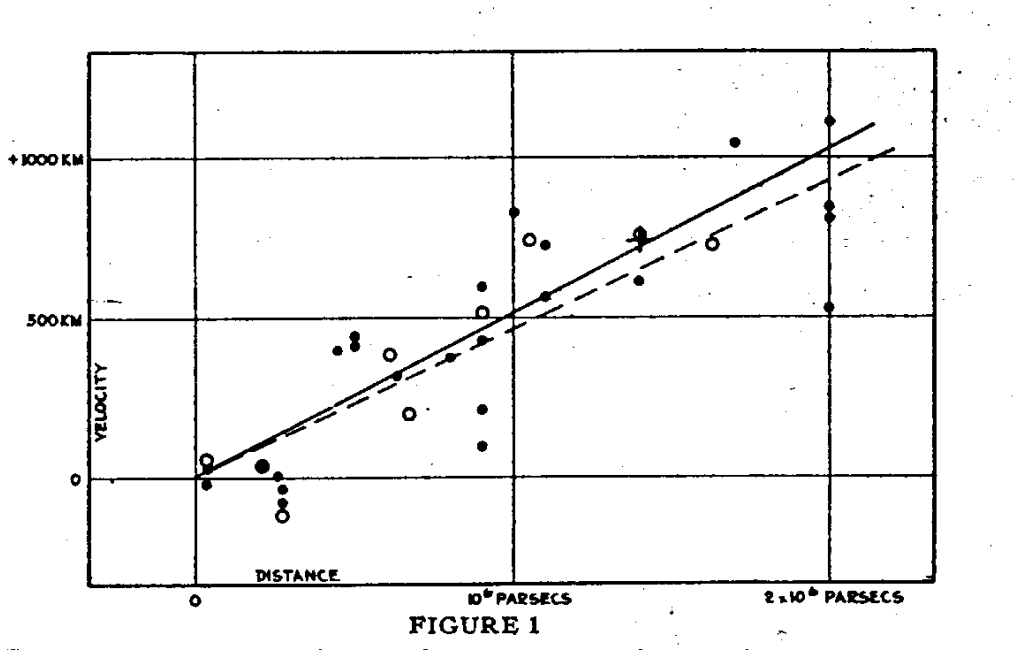


Figura A.1 - A velocidade de afastamento cresce com a distância
Fonte: Hubble (1929)

notou que as velocidades das galáxias em aglomerados eram muito maiores do que deveriam ser, e sugeriu a existência de um tipo de matéria que não emite radiação eletromagnética, mas adiante denominada *matéria escura*. Essa matéria extra não bariônica, compõe cerca de 23% da densidade de energia do Universo e é considerada fria (não-relativística), acolisional e que interage apenas gravitacionalmente (HINSHAW et al., 2012).

A adição de matéria escura nos modelos cosmológicos permite reproduzir estruturas similares às observadas. Atualmente, o modelo cosmológico padrão é denominado ΛCDM (Λ: da constante cosmológica; CDM: *Cold Dark Matter* (MO et al., 2010). Na próxima seção, será descrito as principais características desse modelo.

Se consideramos l_1 a distância de separação entre um par de galáxias num instante arbitrário t_1, podemos definir o *fator de escala* $a(t)$ para expressar a evolução da distância $l(t)$ em relação ao tempo como segue:

$$l(t) = a(t)l_1, \tag{A.1}$$

em seguida, a partir da Equação A.1 temos a lei de Hubble que descreve a relação da velocidade de separação ou aproximação entre um par de galáxias $v = \dot{l}$ em função

a distância de separação $l(t)$ entre elas como segue:

$$v = \frac{\dot{a}(t)}{a(t)} l(t) = H(t) l(t), \tag{A.2}$$

onde $H(t) = \dfrac{\dot{a}}{a}$ é o parâmetro de Hubble, notemos que se $\dot{a} = 0$, $\dot{a} > 0$, ou $\dot{a} < 0$ o teremos um Universo estático, em expansão, em contração respetivamente, mas a partir da descoberta de Hubble (1929) sabemos que $\dot{a} > 0$, atualmente se estima o valor do parâmetro de Hubble é $H(t_0) = H_0 = 70 \pm 0.05 kms^{-1}Mpc^{-1}$ (HINSHAW et al., 2012).

Mas conforme as evidências apresentadas por Perlmutter et al. (1998) sabemos a expansão do Universo se dá forma acelerada, consequentemente conforme Mo et al. (2010) é conveniente definir o parâmetro de desaceleração, como segue:

$$q = -\frac{\ddot{a}a}{\dot{a}^2}, \tag{A.3}$$

onde $q < 0$, isto é temos uma expansão acelerada.

A expansão modifica o comprimento de onda λ da radiação eletromagnética emitida pelos objetos extragaláticos observados pelos astrônomos. Isto se traduz num desvio para o vermelho, ou *redshift*(z), do espectro intrínseco do objeto que é definido como segue:

$$z = \frac{\lambda_o}{\lambda} - 1, \tag{A.4}$$

onde λ_o é o comprimento de onda observado e se assumirmos que a radiação foi emitida num tempo arbitrário t, o fator de escala também permite relacionar essas grandezas como segue

$$\frac{a_0}{a(t)} = \frac{\lambda_o}{\lambda}, \tag{A.5}$$

e em seguida temos que:

$$z = \frac{a_0}{a(t)} - 1. \tag{A.6}$$

A.1.1 Modelo ΛCDM

O modelo ΛCDM considera um evento primordial denominado *Big Bang*, a partir de uma singularidade inicial com densidade infinita que se expandiu exponencialmente (fator de expansão da ordem de 10^{27}) devido a um mecanismo conhecido como *inflação cósmica* (GUTH, 1981). Devido à expansão ocorreu o desacoplamento entre a matéria e a radiação a aproximadamente 380.000 anos após o *Big Bang*, resultando

na primeira superfície de espalhamento conhecida como *Radiação Cósmica de Fundo* (RCF).

Nessa abordagem, a singularidade primordial pontual (*point-like singularity*) é denominada *Singularidade de Friedmann*. Ela utiliza a métrica *Friedmann-Lemaître-Robertson-Walker* (FLRW), dada na Equação A.7 que é solução das equações de campos da relatividade geral e permite descrever a expansão do Universo após a inflação:

$$ds^2 = c^2 dt^2 - a^2(t)\left(\frac{dt}{1-\kappa\, r^2} + r^2\left(d\theta^2 + \sin^2\theta d\phi^2\right)\right), \tag{A.7}$$

onde r, θ, ϕ são as coordenadas espacias comóveis, t é o tempo, c a velocidade da luz, $a(t)$ é o fator de escala e κ é a curvatura do espaço-tempo. Essa métrica é compatível com o *Principio Cosmológico*, homogeneidade e isotropia observadas em grande escala após a inflação (WEINBERG, 2008).

Utilizando a métrica FLRW e a TGR obtém-se as equações de Friedmann que determinam a variação do fator de escala, que são dadas por:

$$\left(\frac{\dot{a}}{a}\right) = \frac{8\,\pi\, G}{3}\rho_m - \frac{\kappa\, c^2}{a^2} + \frac{\Lambda\, c^2}{3} \tag{A.8}$$

$$\frac{\ddot{a}}{a} = -\frac{4\,\pi\, G}{3}\left(\rho_m + \frac{3\,p}{c^2}\right) + \frac{\Lambda c^2}{3} \tag{A.9}$$

onde ρ é densidade media de matéria, p é a pressão, Λ é a constante cosmológica, G é a constante gravitacional universal (MO et al., 2010). A partir da Equação A.8 podemos definir a densidade crítica $\rho_{c,0}$ no tempo atual, correspondente a um Universo plano ($\kappa = 0$) e sem constante cosmológica ($\Lambda = 0$):

$$\rho_{c,0} = \frac{3\, H_0^2}{8\,\pi\, G} \approx 11.26 \quad h^2 \text{ protons}/m^3, \tag{A.10}$$

ela é utilizada para definir o parâmetro de densidade (Ω_0) no tempo atual em função da densidade média ($\overline{\rho}$) como segue:

$$\Omega_0 = \frac{\overline{\rho}_0}{\rho_{c,0}}, \tag{A.11}$$

onde densidade média, conforme Mo et al. (2010), está composta pela soma de todos os componentes que são citados a seguir:

(i) densidade de energia da matéria (bariônica mas escura), $\rho_m \propto a^3$

(ii) densidade de energia da radiação (fótons), $\rho_r \propto a^{-4}$

(iii) densidade de energia do vácuo, constante cosmológica ou energia escura, $\rho_\Lambda = \frac{c^2\Lambda}{8\pi G}$.

Portanto

$$\Omega_0 = \Omega_M + \Omega_R + \Omega_\Lambda \tag{A.12}$$

além disso, se definimos o parâmetro $\Omega_K = \frac{\kappa}{a^2 H_0^2}$ e utilizamos o parâmetro de desaceleração, as equações de Friedmann podem ser re-escritas como:

$$\frac{\dot{a}}{a} = H(t) = H_0\sqrt{\Omega_R\, a^{-4} + \Omega_M\, a^{-3} + \Omega_K\, a^{-2} + \Omega_\Lambda} \tag{A.13}$$

$$\frac{\ddot{a}\, a}{\dot{a}} = \Omega_M/2 - \Omega_\Lambda - \Omega_R \tag{A.14}$$

APÊNDICE B - ABORDAGENS ALTERNATIVAS DO GRUPO DE COSMOLOGIA COMPUTACIONAL DO LAC

B.1 Padrões e Processos Análogos à Turbulência

B.1.1 Teorema de Taylor

O comportamento turbulento é uma importante característica observada nos processos de formação das grandes estruturas do Universo (galáxias, aglomerados, etc)(CARETTA et al., 2008). Conforme Velho (2010) a teoria apresentada por Taylor (1921) poderia ser útil para caraterizar a evolução cósmica. O Teorema de G.I.Taylor estabelece que se a velocidade de um escoamento turbulento $v(t)$ é considerada um processo homogêneo estocástico, com função de autocorrelação $\rho(t)$, a distância media entre dois elementos de fluido num escoamento turbulento é dada por:

$$\langle x^2(t)\rangle = 2\langle v^2(t)\rangle \int_0^t (t-\tau)\tau d\tau. \tag{B.1}$$

A teoria cinética dos gases relaciona a difusividade K com produto da velocidade caraterística por um comprimento caraterística. Assim, difusividade é dimensionalmente o produto da velocidade pela distancia:

$$K \sim \langle v(t)\rangle\langle x(t)\rangle = \langle x(t)\frac{dx(t)}{dt}\rangle = \langle \frac{d}{dt}\left[\frac{1}{2}x^2(t)\right]\rangle = \langle \int_0^t v(t)v(t')dt'\rangle. \tag{B.2}$$

logo em seguida toma-se a média, temos que:

$$\frac{d}{dt}\left[\frac{1}{2}\langle x^2(t)\rangle\right] = \int_0^t \langle v(t)v(t')\rangle dt'. \tag{B.3}$$

Se assumimos que a velocidade é um processo estocástico homogêneo a função correlação é dada por $R(t) = \langle v(t)v(t')\rangle = \langle v^2\rangle\rho(\tau)$. Integrando a equação da difusividade e obtém-se:

$$\langle x^2(t)\rangle = 2\langle v^2\rangle \int_0^t \int_0^\tau \rho(t')dt'd\tau. \tag{B.4}$$

Uma análise assintótica da difusividade, para tempos longos ($t \longrightarrow \infty$) assumindo que $T_L \equiv \int_0^\infty (t-\tau)\tau d\tau$ é a escala de correlação lagrangiana, segue:

$$\langle x^2(t)\rangle = 2\langle v^2\rangle T_L t, \tag{B.5}$$

esta última equação mostra que a distância média entre parcelas de fluido num escoamento turbulento é proporcional a raiz quadrada do tempo transcorrido, isto é:

$$\langle x \rangle \propto \sqrt{t}. \tag{B.6}$$

O COLATUS pode ser usado para verificar se este comportamento assintótico se dá nas simulações cosmológicas.

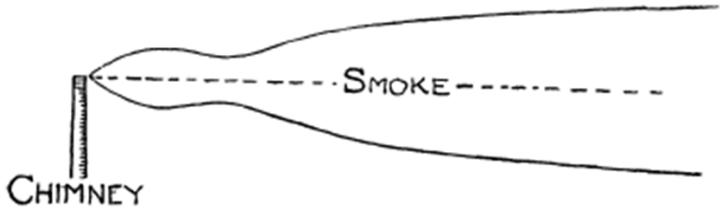


Figura B.1 - A fumaça saindo de uma chaminé mostra um exemplo as predições
Fonte: Taylor (1921)

B.1.2 Teoria do Valor Extremo Generalizado

Recentemente a partir das simulações ΛCDM feitas pelo Consorcio Virgo (VC) foi utilizado a teoria do valor extremo generalizado para mostrar à importância da advecção caótica(ou Turbulência Lagrangiana) em combinação com instabilidades gravitacionais (ROSA et al., 2009). A distribuição de probabilidade do valor extremo generalizado (*Generalized Extreme Value* - GEV, ver Equação B.7) combina as famílias de distribuições Gumbel, Fréchet e Weibull. Pelo teorema do valor extremo da distribuição, (GEV) é a distribuição limite dos máximos normalizados, de uma sequência de variáveis aleatórias independentes e com a mesma distribuição.

$$f(x,\mu,\sigma,\xi) = \frac{1}{\sigma}\left[1-\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\xi-1} e^{-\left[1-\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{1/\xi}} \tag{B.7}$$

onde $\left[1-\xi\left(\frac{x-\mu}{\sigma}\right)\right] > 0, \sigma > 0$ é o parâmetro de escala, μ é o parâmetro de localização e ξ é o parâmetro de forma.

A Figura B.2 mostra que a distribuição de energia dos halos de galáxias identificada

nas simulações do consórcio VIRGO pode ser representada por uma distribuição extrema generalizada. Esta é mais uma ferramenta estatística que pode ser usada para comparar os resultados das simulações do COLATUS com os resultados do VIRGO, GADGET e outros.

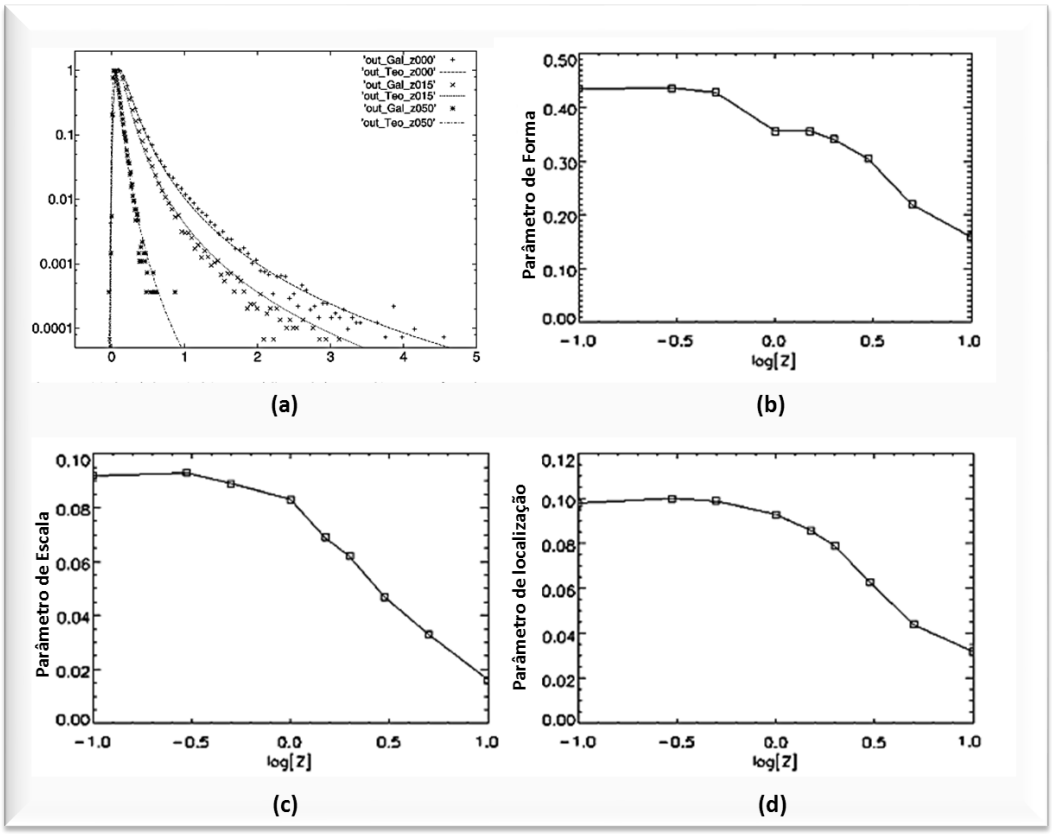


Figura B.2 - Analise do Valor Extremo Generalizado das Energia dos Halos: (a) Histograma da energia normalizada dos halos de galáxias, (b) Evolução do parâmetro de forma em função ao *redshifts*, (c) Evolução do parâmetro de escala em função ao *redshifts*, (d) Evolução do parâmetro de localização em função ao *redshifts*.
Fonte: Rosa et al. (2009)

B.2 Modelo Alternativo com Potencial de Deformação (*Cusp Cosmology*)

O modelo ΛCDM apesar de ser admitido como o modelo padrão da cosmologia contemporânea, apresenta limitações teóricas de grande repercussão, responsáveis por estimular o estudo de modelos alternativos. Entre as principais limitações, destacam-se os seguintes problemas (KROUPA et al., 2010):

- Inconsistência da Singularidade de Friedmann (SF) a partir do estudo do fator de congruência derivado das equações que descrevem o modelo. Este tratamento foi desenvolvido por Amal Kumar Raychaudhuri e Lev Landau. A existência de um ponto sem dimensão com densidade infinita é inconsistente com o elemento de linha resultante da solução da equação de Landau-Raychaudhuri (STEPHANI et al., 2003).

- A matéria escura é introduzida no modelo como **quantidade** extra de matéria gravitacional não-bariônica. Não há uma descrição física **qualitativa** para esta componente.

- A energia escura é introduzida no modelo como sendo a Constante Cosmológica. Porém, ainda não há uma interpretação física e qualitativa para a natureza e origem do parâmetro Λ.

Portanto, a questão mais geral que se apresenta diante do modelo padrão é explicar a natureza de três ingredientes cosmológicos fundamentais: singularidade primordial, matéria e energia escuras. E é notável destacar nesta questão que a explicação qualitativa para o ingrediente físico do modelo está restrita a apenas 4% do conteúdo bariônico que constitui o Universo (ver Figura B.3).

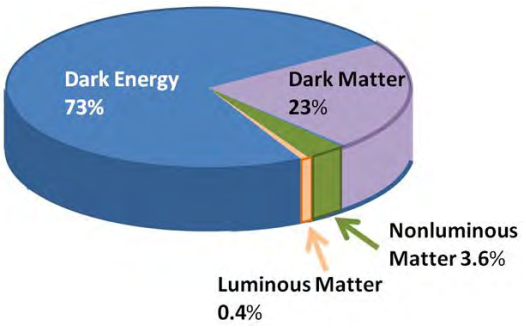


Figura B.3 - Composição do universo de acordo com o modelo Λ CDM e suas evidências observacionais
Fonte: Hinshaw et al. (2012)

Portanto, a questão mais geral que se apresenta diante do modelo padrão é explicar a natureza de três ingredientes cosmológicos fundamentais: singularidade primordial, matéria e energia escuras. E é notável destacar nesta questão que a explicação qualitativa para o ingrediente físico do modelo está restrita a apenas 4.6% do conteúdo bariônico que constitui o Universo (ver Figura B.3).

Abordagens alternativas em cosmologia devem procurar interpretar de forma explícita, de preferência no mesmo modelo, os três problemas cosmológicos fundamentais descritos anteriormente.

Neste contexto, um cenário alternativo, denominado *Cusp Cosmology*, baseado em princípios puramente geométricos foi proposto em 2010 (ROSA, 2010; **??**). O cenário inclui a hipótese de um universo primordial com uma distribuição de energia inicial da ordem de $10^{96} g/cm^3$ de forma homogênea em uma métrica relativística de curvatura nula, sem matéria bariônica e sem entropia (ver Figura B.4(a)). Neste cenário, demonstra-se que na ocorrência de uma instabilidade pontual primordial (condensação taquiônica), um fluxo de energia Λ é gerado sobre as infinitas geodésicas que terminam sobre o ponto de instabilidade resultando um acumulo contínuo de energia que formará um *cusp*. Considerando a geometria projetada em hipersuperfícies (x, ct) e (x, y, ct), o processo de formação do *cusp*é geometricamente compatível com a variação da curvatura gaussiana de nula para negativa, a qual pode ser descrita pela ação variacional sobre uma superfície tautócrona (ver Figura B.4(b))(ROSA, 2010). Posteriormente, a presença de um sistema geodésico tautócrono permite conjecturar que o acúmulo energético contínuo sobre um universo representado por essa geometria resultará na emergência de uma estrutura secundária de curvatura positiva (ver Figura B.4(c,d)) com densidade de energia finita cuja expansão resulta diretamente do fluxo de energia primordial infinita injetada de forma contínua através do *cusp*.

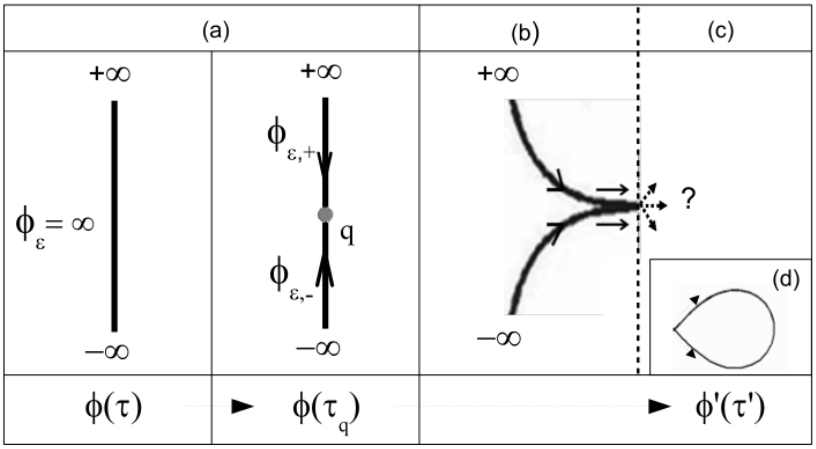


Figura B.4 - Esquema alternativo de um universo primordial baseado em primeiros princípios geométricos considerando a formação de um *cusp* com fluxo contínuo de energia.
Fonte: Rosa (2010)

De acordo com este cenário, a variedade global deve ser composta por três Lagrangianas uma vez que entre a estrutura primordial e a estrutura secundária, há a estrutura do *cusp*.

A primeira propriedade importante deste novo cenário é a ausência de uma singularidade de Friedmann (SF) presente nos modelos convencionais do Big Bang. Dessa forma, este cenário resolve o problema da inconsistência do fator de congruência não nulo que resulta da solução da equação de Landau-RayChaudhuri (este tratamento, em geral negligenciado nos modelos convencionais de Big Bang, compromete a presença da SF em modelos cosmológicos completos) (D'INVERNO, 1992).

A segunda propriedade importante neste cenário é a compatibilidade da estrutura secundária com os modelos padrão do tipo Big Bang (ex. ΛCDM). Neste cenário, a estrutura secundária pode ser interpretada como o Universo bariônico em expansão vinculado à estrutura primordial através do *cusp*. Dessa forma, o cenário também propõe uma nova interpretação para a energia escura como sendo aquela que flui através do *cusp*, podendo estar sujeita à diferentes tipos de modulações.

A terceira propriedade importante do novo cenário, é considerar a possibilidade de a matéria escura ser interpretada como um efeito de equivalência devido à deformação do espaço-tempo imposta pelo vínculo (*cusp*).

É importante notar que dentro deste novo cenário, cabem novas abordagens físicas, que devem ser investigadas e elaboradas com o objetivo de descrever a estrutura primordial em termos de partículas e campos. Várias abordagens alternativas, já em andamento, podem ser aplicadas à estrutura do *cusp*, com destaque para a gravidade quântica (DOROUD; SMOLIN, 2011). A partir da descrição física do *cusp* novas interpretações sobre a origem dos bárions poderão ser elaboradas.

APÊNDICE C - COMPUTAÇÃO HÍBRIDA COM TECNOLOGIA GPU/CUDA

C.1 Aspectos Gerais

A classificação de computadores paralelos (FLYNN, 1972) é a mais conhecida, ela se refere à maneira como um conjunto de dados(fluxo de dados) e conjunto de instruções(fluxo de instruções) são gerenciados em um determinado tipo de processamento.

- SISD (*Single Instruction Single Data*): Fluxo único de instruções sobre um único conjunto de dados. Enquadram-se nesta classificação as máquinas cuja arquitetura segue o padrão Von Newmann, isto é, processadores que executam uma instrução completa de cada vez, sequencialmente, cada uma delas manipulando um dado específico ou os dados daquela operação.

- SIMD (*Single Instruction Multiple Data*): Fluxo único de instruções em múltiplos conjuntos de dados. Neste caso o processador opera de modo que uma única instrução acessa e manipula um conjunto de dados, simultaneamente. Neste caso, a unidade de controle do processador aciona diversas unidades de processamento. Um exemplo desta categoria de arquitetura é o processador vetorial.

- MISD (*Multiple Instruction Single Data*): Fluxo múltiplo de instruções em um único conjunto de dados. Neste caso, o processador opera de modo que uma única instrução acessa e manipula um conjunto de dados, simultaneamente, isto não é muito comum mas é útil para tolerância a falhas.

- MIMD (*Multiple Instruction Multiple Data*): A categoria MIMD refere-se a computadores com múltiplos fluxos de instrução e múltiplos fluxos de dados. Os multiprocessadores encaixam-se nesta categoria, sendo que ambas as arquiteturas baseiam-se em processadores múltiplos.

Em uma configuração híbrida, as máquinas escalares têm sua capacidade de processamento aumentada por meio de dispositivos como por exemplo: GPP - *General Purpose Processor*, DSP - *Digital Signal Processor*, GPU - *Graphic Processor Unit* ou FPGA - *Field Programamble Gate Array* que atuam como co-processadores da *Central Processing Unit*(CPU). Dentro deste contexto, uma tendência mais recente e promissora é a utilização de Unidades de Processamento Gráfico (GPU), que são unidades de processamento que atuam em conjunto com a CPU. Essa abordagem tem

se popularizado devido ao seu enorme poder computacional e baixo custo. Na lista top 500 recentemente divulgada pelo site http://www.top500.org, o computador que ocupa a posição nº 1, ou seja, o computador mais veloz do mundo (supercomputador Titan) contém 18688 nós, sendo que cada nó contém uma CPU AMD opteron 6274 de 16 núcleos e uma GPU NVIDIA Tesla K20.

As GPUs sugiram na década de 1990 inicialmente como placas gráficas projetadas apenas para renderização de imagens. Na época, o principais fabricantes de placas gráficas eram: VIRGE, 3DFX, NVIDIA e ATI. A partir do ano 2000, a NVIDIA e a ATI iniciaram projetos arrojados para dominar o mercado das GPUs. A evolução das GPUs foi motivada pelo alto custo computacional exigido por aplicações gráficas (principalmente jogos 3D), onde os engenheiros buscam cada vez mais eficiência, desempenho e qualidade gráfica na exibição de animações visuais. Recentemente a capacidade de processamento das GPUs tornou-se tão expressiva que programadores de outras áreas e pesquisadores passaram a aprimorar formas de utilizá-las para outros propósitos, como por exemplo a computação científica. A partir dessas iniciativas surgiu a computação de propósito geral - GP-GPU (General-Purpose computation on Graphics Processing Units) (BRODTKORB et al., 2010).

Inicialmente, a programação em GPU era feita mediante instruções do OpenGL ou Direct3D, consequentemente isso não era tarefa fácil. Mas em 2007, a NVIDIA introduziu o modelo de programação CUDA - Compute Unified Device Architecture, que admite a execução de uma aplicação na CPU e na GPU, usando as ferramentas de programação C/C++ e o conjunto de bibliotecas e ferramentas disponibilizadas pela NVIDIA (KIRK; HWU, 2011). Além de CUDA, em 2008 a Kronos Group apresentou uma linguagem aberta para programar as GPUs, conhecida como OpenCL.

OpenCL é uma arquitetura para programar sistemas de computação híbridos, possibilitando o uso conjunto de CPUs, GPUs e outros dispositivos. OpenCL inclui uma linguagem para escrever *kernels* (funções executadas em dispositivos OpenCL), além das interfaces de aplicação que são usadas para definir e depois controlar as plataformas híbridas. Ela foi adotada para controladores de placas gráficas pela AMD/ATI, e pela NVIDIA, que oferece também suporte ao OpenCL como alternativa a CUDA. Também encontra-se em desenvolvimento um novo padrão de programação paralela, denominado OpenACC. Esse padrão desenvolvido pela NVIDIA, Cray Inc. e CASP pretende facilitar a programação das GPUs oferecendo um conjunto de diretivas para especificar quais regiões do código C, C++ e Fortran devem ser executadas na GPU.

Atualmente CUDA, mesmo sendo uma linguagem proprietária, é a ferramenta mais difundida e fácil de utilizar, e em algumas aplicações tem conseguido melhor desempenho que OpenCL (OWENS et al., 2007; KIRK; HWU, 2011).

No presente trabalho, são desenvolvidos programas utilizando-se linguagem C, C++ e CUDA para simular o processo de formação de estruturas em escalas cosmológicas utilizando GPUs.

C.2 GPU-Graphics Processing Unit

Em 2003, as limitações físicas e práticas para o aumento da frequência do clock dos processadores fizeram com que duas novas estratégias de aumento de desempenho ganhassem força dentro das fabricas de processadores (KIRK; HWU, 2011). A primeira estratégia começou colocando múltiplos núcleos (*multicore*) dentro de um mesmo chip, um exemplo atual são os microprocessadores Intel Core i7 que contem até seis núcleos (ver figura C.1, (a)), cada um sendo um processador independente. Esses processadores são projetados para maximizar a velocidade de execução dos programas sequenciais. A segunda estratégia foca-se na execução de várias aplicações em paralelo, colocando muitos núcleos (*many-core*). As GPUs são um exemplo do uso desta estratégia, atualmente a GTX 690, lançada em abril de 2012, com 3072 núcleos e tecnologia Kepler (ver figura C.1, (b)).

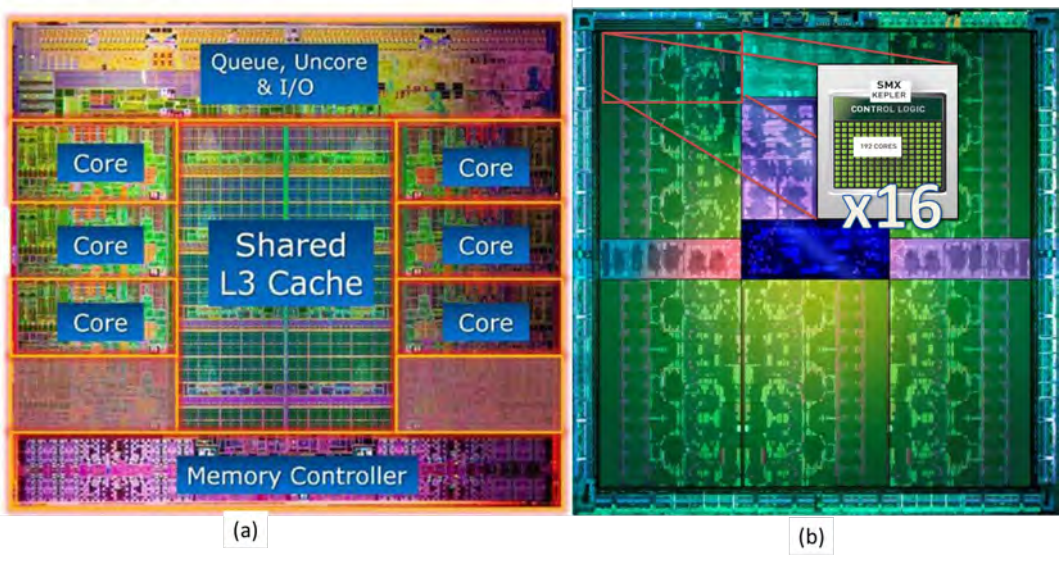


Figura C.1 - Em (a) Processador Intel Core i7 3960X com seis núcleos de CPU, em (b) GTX690 com 16 Multiprocessadores de 192 núcleos em cada um.

As GPUs estabeleceram uma ampla vantagem no poder computacional em relação às CPUs da Intel. Um processador Intel Core i7 3960X atinge apenas 119.45 GFlops/s (largura de banda da memoria 51.2 GB/s)[1], enquanto uma GPU GTX 690 da NVIDIA, pode atingir até 5,62 TeraFlop/s (largura de banda da memoria 384 GB/s), AMD também não ficou atrás, a Radeon HD 7990 pode atingir até 6,96 TeraFlop/s (largura de banda da memoria 2x264 GB/s)[2]. Na Figura C.2, observa-se a que, a partir de novembro de 2006, as GPUs da NVIDIA estabeleceram uma larga vantagem no poder computacional em relação às CPUs, e na figura C.4 pode se observar outra ampla vantagem das GPUs em relação as CPUs, a largura de banda da memória.

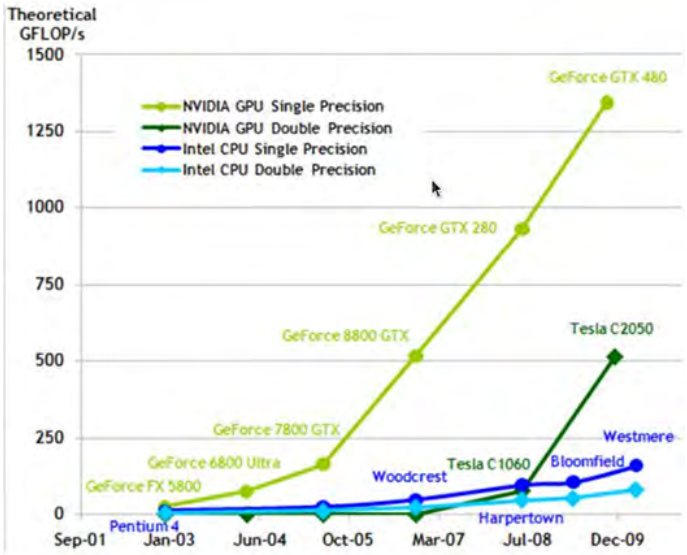


Figura C.2 - A evolução do desempenho da CPU vs GPU
Fonte: NVIDIA CUDA C Programming Guide

Architecture	Competes with NVIDIA Architecture	GPU Chip (Card Name)	SIMD Engines (Thread Processors)	Compute Cores SP / DP / SFU (5/1/1 per TP)	Max TFLOPs (SP / DP)
R600 Family	G80	RV670 (HD3870)	4 (64)	320 / 64 / 64	.496 / .099
R700 Family	GT200	RV770 (HD4870)	10 (160)	800 / 160 / 160	1.20 / .24
Evergreen Family		Juniper (HD5770)	10 (160)	800 / 0 / 160	1.36 / 0
	GF100	Cypress (HD5870)	20 (320)	1600 / 320 / 320	2.72 / .544
		Hemlock (HD5970)	20 x2 (640)	3200 / 640 / 640	4.64 / .928

Figura C.3 - A evolução do desempenho da ATI.
Fonte: http://www.microway.com/pdfs/GPGPU_Architecture_and_Performance_Comparison.pdf

[1]Disponível em: http://www.brightsideofnews.com/news/2011/11/14/review-intel-core-i7-3960x-and-intel-x79-dx79si.aspx?pageid=2 Acessado em: 13 jan. 2013.
[2]Disponível em: http://www.mersenne.ca/mfaktc.php?sort=gflops Acessado em: 13 jan. 2013.

A razão de tão grande lacuna de desempenho entre GPUs e CPUs está na filosofia de projetos fundamentais (ver Figura C.5) (KIRK; HWU, 2011). A arquitetura de uma CPU é otimizada para o desempenho de código sequencial, consequentemente para que uma *thread* seja executada pelos vários núcleos os processadores precisam muita memória cache e uma lógica de controle sofisticada, de tal forma a diminuir a latência e manter uma aparência sequencial. No caso das GPUs, elas são projetadas para executar um número massivo de *threads*, onde mais transistores são dedicados ao processamento de dados, em vez de cache de dados e controle de fluxo.

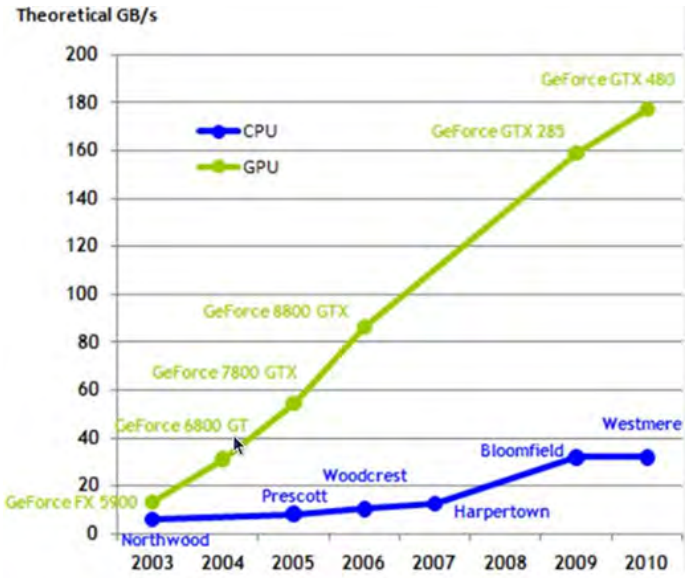


Figura C.4 - A evolução da largura de banda da GPU vs GPU
Fonte: NVIDIA CUDA C Programming Guide

Outra questão relevante é a largura de banda da memória, as GPUs tem aproximadamente 4 vezes a largura de banda dos chips de CPU disponíveis atualmente. Isso ocorre porque na CPU os requisitos dos sistemas operacionais, aplicações e dispositivos de E/S limitam o aumento da largura de banda na CPU. Mas como as GPUs inicialmente foram desenhadas para processar gráficos, a precisão e exatidão numérica das operações de ponto flutuante foram deixadas de lado em benefício da velocidade, porém nos últimos anos conforme outros tipos de aplicação foram utilizando as GPUs, os fabricantes alcançaram precisão e exatidão numérica equivalentes as CPUs nas novas GPUs.

Atualmente as GPUs estão equipadas as muitas unidades de processamento ou núcleos (*Streaming Processor*-SP) que estão agrupadas em multiprocessadores (*Streaming MultiProcessor*-SM) que compartilham a lógica de controle e a cache de

Figura C.5 - Diferenças na arquitetura da GPU e GPU

instruções. Cada GPU atualmente vem com até 6 gigabytes de memória própria (DRAM) para gerenciar os dados processados. Hoje as GPUs são utilizadas como co-processadores genéricos, ou seja, para realizar instruções computacionais de forma altamente paralela. Este novo uso das GPUs, foca-se em explorar suas vantagens em aplicações de propósito geral altamente paralelizáveis que exigem fluxo intenso de cálculos.

C.3 Arquitetura CUDA

Em novembro de 2007, a NVIDIA lança a CUDA (*Compute Unified Device Architecture*), é uma arquitetura de *hardware* e *software* que permite as GPUs da NVIDIA executar em paralelo programas escritos em C, C++, Fortran e outras linguagens (NVIDIA, 2012). A NVIDIA modificou a arquitetura do *hardware* da GPU e também desenvolveu o compilador CUDA C/C++, bibliotecas e software *runtime* para facilitar a programação da GPUs.

A arquitetura G80 foi a primeira geração de GPU NVIDIA habilitada para usar CUDA. Ela possuía 128 *shaders* programáveis, agora conhecidos como núcleos ou SP. Cada conjunto de 8 SP, são agrupados em um multiprocessador MP, cujos núcleos compartilham recursos, como memória local, registros de arquivos, unidades de load/store, escalonadores de *threads*. A G80 tinha 16 multiprocessadores, totalizando 128 núcleos por chip.

Em 2008, a arquitetura GT200 originou a unificação do processamento gráfico com a computação paralela. Entretanto, a terceira geração, Fermi, empregou uma abordagem completamente nova para projetar e criar-lo (OWENS et al., 2007). A Figura C.6 mostra o diagrama de blocos de alto nível da arquitetura Fermi. Cada multiprocessador tem 32 núcleos. As GPUs baseadas na arquitetura Fermi possuem 16

multiprocessadores, gerando um total até 512 núcleos CUDA por chip.

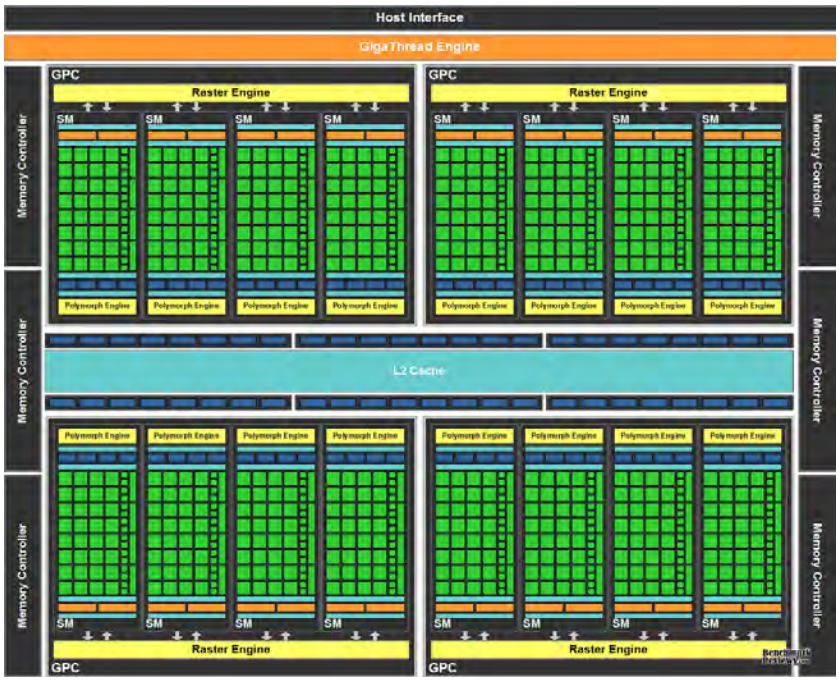


Figura C.6 - Arquitetura Fermi
Fonte: NVIDIA (2012)

A arquitetura Kepler é a ultima arquitetura projetada pela NVIDIA. A Figura C.7 mostra os diagramas de blocos de alto nível da arquitetura Kepler, a Figura C.8 mostra em detalhe, um bloco do SM da arquitetura Fermi e Kepler . As GPUs Kepler tem 192 núcleos CUDA por cada SM - seis vezes mais que a Fermi, e possuem até 16 SMs, gerando um total de 3072 núcleo CUDA por chip - seis vezes mais que a Fermi. Os SM da Kepler foram redesenhados para alcançar um melhor desempenho e eficiência, e ganharam uma nova denominação SMX.

Cada um dos núcleos do multiprocessador (SM) possui uma unidade de ponto flutuante (FP Unit, padrão IEEE 754-2008), uma unidade de inteiro (INT Unit, 64 bit), lógica para o envio de instruções e operandos para essas unidades. Cada SM (da Fermi) contem 32 núcleos SP, 16 unidades load-store para o envio de instruções e operandos para estas unidades, quatro unidades de funções especiais (SFUs, para lidar com operações transcendentais como seno,cosseno, exponencial e outras) e 64 KB de RAM com particionamento configurável de memória compartilhada e cache L1.

Por outro lado, cada multiprocessador SMX (da Kepler) contém 192 núcleos CUDA,

Figura C.7 - Arquitetura Kepler
Fonte: NVIDIA (2012)

32 unidades load-store, 32 unidades de funções especiais, 64 KB de RAM de memória compartilhada, cache L1 e 48KB de memória só para leitura. Cada SMX tem quatro unidades de escalonamento - duas vezes mais que a SM da Fermi, eles agrupam conjuntos 32 *threads*, chamados de *warps* e 8 unidades de busca/despacho de instruções (IDU-*Instruction Dispath units*)- quatro vezes mais que a SM da Fermi permitindo assim que quatro *warps* sejam executados concorrentemente(NVIDIA, 2012).

A Figura C.9 mostra uma sequência de instruções a serem distribuídas entre os blocos de execução disponíveis. Cada multiprocessador pode gerenciar 64 *warps*. Como cada *warp* tem 32 *threads*, um multiprocessador pode gerenciar 2048 *threads*. Com 16 SMs, uma GPU Kepler pode manipular até 32768 *threads* paralelos.

C.4 GPU 1: GeForce GTX 580(Fermi)

```
1 Device 0: "GeForce GTX 580"
2
3   CUDA Driver Version / Runtime Version          5.0 / 4.2
4
5   CUDA Capability Major/Minor version number:    2.0
6
7   Total amount of global memory:                 1536 MBytes
8   (16) Multiprocessors x ( 32) CUDA Cores/MP:    512 CUDA Cores
9
```

Figura C.8 - Um bloco multiprocessador (SM), (a) da arquitetura Fermi, (b) da arquitetura Kepler
Fonte: NVIDIA (2012)

```
  GPU Clock rate:                                1544 MHz (1.54 GHz)
  Memory Clock rate:                             2004 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536),
                                                 2D=(65536,65535),
                                                 3D=(2048,2048,2048)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048,
                                                 2D=(16384,16384) x ...
                                                    2048
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768

  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1536
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
```

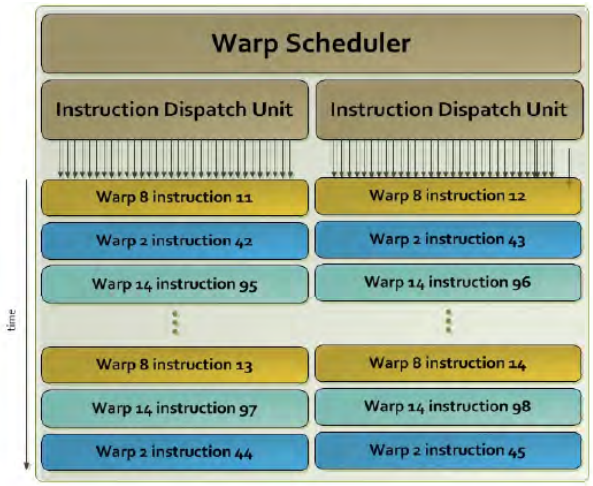


Figura C.9 - Unidade de escalonamento de *warps*-(*Warp Scheduler*)
Fonte: NVIDIA (2012)

```
30   Concurrent copy and execution:          Yes with 1 copy engine(s)
31
32   Run time limit on kernels:                     No
33   Integrated GPU sharing Host Memory:            No
34   Support host page-locked memory mapping:       Yes
35   Concurrent kernel execution:                   Yes
36   Alignment requirement for Surfaces:            Yes
37   Device has ECC support enabled:                No
38   Device is using TCC driver mode:               No
39   Device supports Unified Addressing (UVA):      Yes
40   Device PCI Bus ID / PCI location ID:           3 / 0
```

C.5 GPU 2: GeForce GTX 680(Kepler)

```
1  Device 0: "GeForce GTX 680"
2    CUDA Driver Version / Runtime Version          4.2 / 4.2
3    CUDA Capability Major/Minor version number:    3.0
4    Total amount of global memory:                 2048 MBytes ...
         (2147155968 bytes)
5    ( 8) Multiprocessors x (192) CUDA Cores/MP:    1536 CUDA Cores
6    GPU Clock rate:                                706 MHz (0.71 GHz)
7    Memory Clock rate:                             3004 Mhz
8    Memory Bus Width:                              256-bit
9    L2 Cache Size:                                 524288 bytes
10   Max Texture Dimension Size (x,y,z)             1D=(65536),
11                                                  2D=(65536,65536),
```

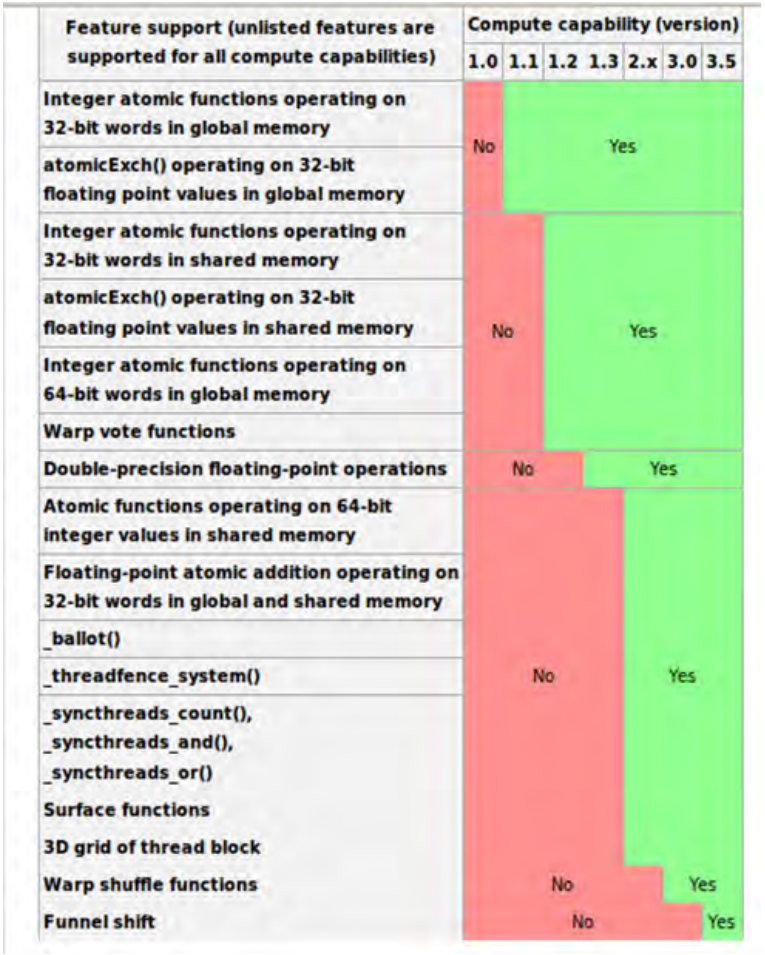
```
                                                    3D=(4096,4096,4096)
  Max Layered Texture Size (dim) x layers           1D=(16384) x 2048,
                                                   2D=(16384,16384) x 2048
  Total amount of constant memory:                  65536 bytes
  Total amount of shared memory per block:          49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                        32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:              1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid: 2147483647 x 65535 x 65535
  Maximum memory pitch:                             2147483647 bytes
  Texture alignment:                                512 bytes
  Concurrent copy and execution:             Yes with 1 copy engine(s)
  Run time limit on kernels:                        Yes
  Integrated GPU sharing Host Memory:               No
  Support host page-locked memory mapping:          Yes
  Concurrent kernel execution:                      Yes
  Alignment requirement for Surfaces:               Yes
  Device has ECC support enabled:                   No
  Device is using TCC driver mode:                  No
  Device supports Unified Addressing (UVA):         Yes
  Device PCI Bus ID / PCI location ID:              4 / 0
```

C.6 Capacidade de Computação (*Compute Capability*)

A capacidade de computação mostra as principais diferenças entre cada arquitetura da NVIDIA.

Architecture specifications	Compute capability (version)						
	1.0	1.1	1.2	1.3	2.0	2.1	3.0	3.5
Number of cores for integer and floating-point arithmetic functions operations	8[15]				32	48	192	192
Number of special function units for single-precision floating-point transcendental functions	2				4	8	32	32
Number of texture filtering units for every texture address unit or *render output unit* (ROP)	2				4	8	32	32
Number of warp schedulers	1				2	2	4	4
Number of instructions issued at once by scheduler	1				1	2[16]	2	2

Figura C.10 - Especificações da arquitetura
Fonte: CUDA C Programming Guide



Feature support (unlisted features are supported for all compute capabilities)	Compute capability (version)						
	1.0	1.1	1.2	1.3	2.x	3.0	3.5
Integer atomic functions operating on 32-bit words in global memory	No	Yes					
atomicExch() operating on 32-bit floating point values in global memory							
Integer atomic functions operating on 32-bit words in shared memory	No		Yes				
atomicExch() operating on 32-bit floating point values in shared memory							
Integer atomic functions operating on 64-bit words in global memory							
Warp vote functions							
Double-precision floating-point operations	No			Yes			
Atomic functions operating on 64-bit integer values in shared memory	No				Yes		
Floating-point atomic addition operating on 32-bit words in global and shared memory							
_ballot()							
_threadfence_system()							
_syncthreads_count(), _syncthreads_and(), _syncthreads_or()							
Surface functions							
3D grid of thread block							
Warp shuffle functions	No					Yes	
Funnel shift	No						Yes

Figura C.11 - Funções Suportadas.
Fonte: CUDA C Programming Guide

Technical specifications	Compute capability (version)						
	1.0	1.1	1.2	1.3	2.x	3.0	3.5
Maximum dimensionality of grid of thread blocks	2				3		
Maximum x-, y-, or z-dimension of a grid of thread blocks	65535				2^{31}-1		
Maximum dimensionality of thread block	3						
Maximum x- or y-dimension of a block	512				1024		
Maximum z-dimension of a block	64						
Maximum number of threads per block	512				1024		
Warp size	32						
Maximum number of resident blocks per multiprocessor	8				16		
Maximum number of resident warps per multiprocessor	24		32		48	64	
Maximum number of resident threads per multiprocessor	768		1024		1536	2048	
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K	64 K	
Maximum number of 32-bit registers per thread	128				63		255
Maximum amount of shared memory per multiprocessor	16 KB				48 KB		
Number of shared memory banks	16				32		
Amount of local memory per thread	16 KB				512 KB		
Constant memory size	64 KB						
Cache working set per multiprocessor for constant memory	8 KB						
Cache working set per multiprocessor for texture memory	Device dependent, between 6 KB and 8 KB						
Maximum width for 1D texture reference bound to a CUDA array	8192				65536		
Maximum width for 1D texture reference bound to linear memory	2^{27}						
Maximum width and number of layers for a 1D layered texture reference	8192 x 512				16384 x 2048		
Maximum width and height for 2D texture reference bound to a CUDA array	65536 x 32768				65536 x 65535		
Maximum width and height for 2D texture reference bound to a linear memory	65000 x 65000				65000 x 65000		
Maximum width and height for 2D texture reference bound to a CUDA array supporting texture gather	N/A				16384 x 16384		
Maximum width, height, and number of layers for a 2D layered texture reference	8192 x 8192 x 512				16384 x 16384 x 2048		
Maximum width, height and depth for a 3D texture reference bound to linear memory or a CUDA array	2048 x 2048 x 2048				4096 x 4096 x 4096		
Maximum width (and height) for a cubemap texture reference	N/A				16384		
Maximum width (and height) and number of layers for a cubemap layered texture reference	N/A				16384 x 2046		
Maximum number of textures that can be bound to a kernel	128				256		
Maximum width for a 1D surface reference bound to a CUDA array	Not supported				65536		
Maximum width and number of layers for a 1D layered surface reference					65536 x 2048		
Maximum width and height for a 2D surface reference bound to a CUDA array					65536 x 32768		
Maximum width, height, and number of layers for a 2D layered surface reference					65536 x 32768 x 2048		
Maximum width, height, and depth for a 3D surface reference bound to a CUDA array					65536 x 32768 x 2048		
Maximum width (and height) for a cubemap surface reference bound to a CUDA array					32768		
Maximum width (and height) and number of layers for a cubemap layered surface reference					32768 x 2046		

Figura C.12 - Especificações técnicas.
Fonte: CUDA C Programming Guide

APÊNDICE D - CÓDIGO

D.1 Gerador de Condições Iniciais N-GENIC

D.1.1 Parâmetros Cosmológicos do Modelo ΛCDM

WMAP Cosmological Parameters Model: lcdm Data: wmap9			
$10^9\Delta^2_{\mathcal{R}}$	2.41 ± 0.10	H_0	70.0 ± 2.2 km/s/Mpc
$\ell(\ell+1)C_{220}/(2\pi)$	$5746 \pm 35\ \mu\mathrm{K}^2$	$d_A(z_{\mathrm{eq}})$	14194 ± 117 Mpc
$d_A(z_*)$	14029 ± 119 Mpc	$D_v(z=0.57)/r_s(z_d)$	13.28 ± 0.31
η	$(6.19 \pm 0.14) \times 10^{-10}$	k_{eq}	0.00996 ± 0.00032
ℓ_{eq}	139.7 ± 3.5	ℓ_*	302.35 ± 0.65
n_b	$(2.542 \pm 0.056) \times 10^{-7}$ cm^{-3}	n_s	0.972 ± 0.013
n_s	$0.95 < n_s < 1.00$ (95% CL)	Ω_b	0.0463 ± 0.0024
$\Omega_b h^2$	0.02264 ± 0.00050	Ω_c	0.233 ± 0.023
$\Omega_c h^2$	0.1138 ± 0.0045	Ω_Λ	0.721 ± 0.025
Ω_m	0.279 ± 0.025	$\Omega_m h^2$	0.1364 ± 0.0044
$r_s(z_d)$	152.3 ± 1.3 Mpc	$r_s(z_d)/D_v(z=0.106)$	0.346 ± 0.012
$r_s(z_d)/D_v(z=0.2)$	0.1889 ± 0.0060	$r_s(z_d)/D_v(z=0.35)$	0.1135 ± 0.0032
$r_s(z_d)/D_v(z=0.44)$	0.0932 ± 0.0024	$r_s(z_d)/D_v(z=0.54)$	0.0787 ± 0.0019
$r_s(z_d)/D_v(z=0.57)$	$0.0753^{+0.0017}_{-0.0018}$	$r_s(z_d)/D_v(z=0.6)$	0.0724 ± 0.0016
$r_s(z_d)/D_v(z=0.73)$	0.0624 ± 0.0013	$r_s(z_*)$	145.8 ± 1.2
R	1.728 ± 0.016	σ_8	0.821 ± 0.023
$\sigma_8\Omega_m^{0.5}$	0.434 ± 0.029	$\sigma_8\Omega_m^{0.6}$	0.382 ± 0.029
A_{SZ}	< 2.0 (95% CL)	t_0	13.74 ± 0.11 Gyr
τ	0.089 ± 0.014	θ_*	0.010391 ± 0.000022
θ_*	0.5953 ± 0.0013 °	τ_{rec}	283.9 ± 2.4
t_{reion}	453^{+63}_{-64} Myr	t_*	376371^{+4115}_{-4111} yr
z_d	1020.7 ± 1.1	z_{eq}	3265^{+106}_{-105}
z_{rec}	1088.16 ± 0.79	z_{reion}	10.6 ± 1.1
z_*	$1090.97^{+0.85}_{-0.86}$		

Figura D.1 - Parâmetros cosmológicos
Fonte: Hinshaw et al. (2012)

D.1.2 Arquivos de Configuração

```
1 Nmesh             32
```

```
% This is the size of the FFT grid used to compute the
%displacement field. One should have Nmesh ≥ Nsample.
Nsample           16  % sets the maximum k that the code uses,
 %i.e. this effectively determines the Nyquist frequency that
 %the code assumes, k_Nyquist = 2*PI/Box * Nsample/2
 %Normally, one chooses Nsample such that Ntot =  Nsample^3,
 %where Ntot is the total number of particles
Box               16  % Periodic box size of simulation L/h
FileBase          ics % Base-filename of output files
OutputDir         ./ICs           % Directory for output
GlassFile         dummy_glass.dat
%File with unperturbed glass or Cartesian grid
TileFac           1    % Number of times the glass file is
%tiled in each dimension (must be an integer)
Omega             0.2793      % Total matter density  (at z=0)
OmegaLambda       0.721      % Cosmological constant (at z=0)
OmegaBaryon       0.0463     % Baryon density        (at z=0)
HubbleParam       0.70     % Hubble paramater (may be used
%for power spec parameterization)
Redshift          30        % Starting redshift
Sigma8            0.8       % power spectrum normalization
SphereMode        1         % if "1" only modes with
%|k| < k_Nyquist are used (i.e. a sphere in k-space),
%otherwise modes with |k_x|,|k_y|,|k_z| < k_Nyquist are used
%(i.e. a cube in k-space)
WhichSpectrum     0 % "1" selects Eisenstein & Hu spectrum,
                    % "2" selects a tabulated power spectrum in
                    % the file 'FileWithInputSpectrum'
                    % otherwise, Efstathiou parametrization is used
FileWithInputSpectrum   input_spectrum.txt
%filename of tabulated input spectrum (if used)
InputSpectrum_UnitLength_in_cm  3.085678e24
%defines length unit of tabulated input spectrum in cm/h.
ReNormalizeInputSpectrum   1  % if set to zero, the
%tabulated spectrum is assumed to be normalized already
%in its amplitude to the starting redshift, otherwise
%this is recomputed based on the specified sigma8
ShapeGamma        0.21   % only needed for Efstathiou power spectrum
PrimordialIndex   1.0    % may be used to tilt the primordial index,
                         % primordial spectrum is k^PrimordialIndex
Seed              123456    %  seed for IC-generator
NumFilesWrittenInParallel 1 % limits the number of files that are
                              % written in parallel when outputting
UnitLength_in_cm          3.085678e24
% defines length unit of output (in cm/h)
```

```
47 UnitMass_in_g             1.9885e43
48 % defines mass unit of output (in g/(cm/h))
49 UnitVelocity_in_cm_per_s  1e5
50 % defines velocity unit of output (in cm/sec)
```

D.1.3 Arquivos de Saida

```
1 <?xml version="1.0"?>
2 <N-GenIC NumBodies="4096" Redshift="30.000000" Box="16"
3 OmegaMatter="0.270000" OmegaLambda="0.730000" OmegaBaryon="0.045000"
4 HubbleParam_10km_s_Mpc="7.210000" WhichSpectrum="Efstathiou
5 parametrization" Sigma8="0.800000" ShapeGamma="0.210000"
6 PrimordialIndex="1.000000"  UnitLength_in_m_Mpc="3.085678e+22"
7 UnitMass_in_kg_1e10_Msol="1.988500e+40"
8 UnitVelocity_in_m_s_10Km_s="1.000000e+04"
9 Gravitational_Constant_m3_kg_s2="6.673840e-11"
10 Binary_file="Yes" Data_filename="./ICs/ics.bin"/>
```

D.2 Simulador N-Corpos em CUDA: COLATUS

D.2.1 Arquivos de Configuração

```
1 <?xml version="1.0"?>
2 <NBody_Sim_Config  Device_to_run="GPU" Compute_energy="Yes"
3 Save_snapshots="Yes" Max_Snapshots="40"
4 Use_Leap_Frog="Yes" Use_Comoving_Coordinates="Yes"
5 Comov_peculiar_velocities="Yes" Softering_epsilon="0.03"
6 Use_Variable_Time_step="Yes"Max_time_steps="10000" Eta="0.1"
7 Use_Parametrization_as_Time="No" Alpha="0.6" Redshift_final="1e-3"
8 Tracking_ID="128"IC_file="./ICs/ics.xml"
9 Binary_output="No" Out_file="./sim/out"/>
```

D.2.2 Integração das Equações do Movimento

D.2.2.1 Cálculo das Forças de Interação:bodyBodyInteraction

```
1 __device__ double bodyBodypot(double sum_mG_rij,double3 ri,double3 rj){
2     double3 r;
3     double L_2=L*0.5;
4     r.x = fmod(rj.x+L_2-ri.x,L) - L_2;
5     r.y = fmod(rj.y+L_2-ri.y,L)-  L_2;
6     r.z = fmod(rj.z+L_2-ri.z,L) - L_2;
```

```
7
8       double distsqrt = r.x*r.x + r.y*r.y + r.z*r.z;
9       double mg_rij = mG*rsqrtf(distsqrt);
10
11      sum_mG_rij += mg_rij;
12      return sum_mG_rij;
13  }}
```

D.2.2.2 Cálculo da Aceleração: ComputeAcceleration

```
1   __global__ void ComputeAcceletion(double C1,double C2,double ...
        eps_comov){
2       int bx      = blockIdx.x;     int tx      = threadIdx.x;
3       int dimX    = blockDim.x;     int idx     = bx * dimX + tx;
4       //printf("\n dim block %d blid %d txid %d i %d",dimX,bx,tx,idx);
5       double n_acc=0.0,n_vel=0.0;
6
7       if (idx <mNumBodies){
8           double3 sum_mGrji_dij2 =make_double3(0,0,0);
9           double3 acc =make_double3(0,0,0);
10          double3 pos = Pos[idx];      double3 vel = Vel[idx];
11          for (int j=0; j<idx; j++)
12              sum_mGrji_dij2= bodyBodyInteraction(sum_mGrji_dij2, pos,
13                                                  Pos[j], eps_comov);
14          for (int j=idx+1; j<mNumBodies; j++)
15              sum_mGrji_dij2= bodyBodyInteraction(sum_mGrji_dij2, pos,
16                                                  Pos[j], eps_comov);
17
18      acc.x =C1*sum_mGrji_dij2.x+C2*vel.x;
19      acc.y =C1*sum_mGrji_dij2.y+C2*vel.y;
20      acc.z =C1*sum_mGrji_dij2.z+C2*vel.z;
21      n_acc=(acc.x*acc.x+acc.y*acc.y+acc.z*acc.z);
22      n_vel=(vel.x*vel.x+vel.y*vel.y+vel.z*vel.z);
23
24      Acc[idx]=acc;
25      Norm_acc[idx]=n_acc;
26      Norm_vel[idx]=n_vel;
27          double nn_acc=0.0,nn_vel=0.0;
28      if(idx==(mNumBodies-1))// No keppler tirar este if!!!!
29          {
30              double acc_max=0.0,vel_max=0.0;
31              for (int j=0; j<mNumBodies; j++)
32                  {
33                  nn_acc=Norm_acc[j];
```

```
                nn_vel=Norm_vel[j];
                acc_max=nn_acc>acc_max?nn_acc:acc_max;
                vel_max=nn_vel>vel_max?nn_vel:vel_max;
                }
            d_g[0]=acc_max;
            d_g[1]=vel_max;
        }
```

D.2.2.3 Cálculo do Passo de Tempo

```
//Copia da GPU para CPU  a norma da acc max e da vel_max
cudaMemcpy(d,d_g,sizeof(double)*2, cudaMemcpyDeviceToHost);
cudaThreadSynchronize();
//dp≤eta/sqrt(acc_max);
d[0]=eta/sqrt(sqrt(d[0]));
//dp≤eta/(vel_max);
d[1]=eta/sqrt(d[1]);
//Escolhe o menor passo de tempo
dp=d[0]<d[1]?d[0]:d[1];

dp=dp<dp_min?dp_min:dp;
```

D.2.2.4 Integração: Integrate

```
__global__ void integrate(int it,double C1,double C2,double C3) {
    int bx     = blockIdx.x;    int tx     = threadIdx.x;
    int dimX   = blockDim.x;    int idx    = bx * dimX + tx;
    double3 acc =make_double3(0,0,0);
    if (idx < mNumBodies){
        double3 pos = Pos[idx];
        double3 vel = Vel[idx];
        double3 acc = Acc[idx];
        // Atualizar velocidade
        vel.x =vel.x*C1  +acc.x * C2;
        vel.y =vel.y*C1  +acc.y * C2;
        vel.z =vel.z*C1  +acc.z * C2;
        // Atualizar posicao
        pos.x = fmod (pos.x+vel.x*C3+L,L);
        pos.y = fmod (pos.y+vel.y*C3+L,L);
        pos.z = fmod (pos.z+vel.z*C3+L,L);

        // Atualizar posicao global
        nVel[idx] = vel;
```

```
20          nPos[idx] = pos;
21          //Salvando informacao de uma particula
22          if (idx==i_snap){
23              Pos_i[it]=pos;      Vel_i[it]=vel;
24          }
25  }
26  }
```

D.2.2.5 Integração das Equações de Friedmann

```
1  a[0]=1.0/(zin+1);      p=pow(a[0],alpha);
2  for (int it = 0; it < max_steps; it++){
3  if (use_p){
4      p=p+dp;
5      a[it+1]=pow(p,1.0/alpha); // lembrando que p=a^(alpha)
6  }else{
7      //Primeira equação de Friedmann
8      dota=sqrt(1+omegaM*(1/a[it]-1)+omegaR*(1/a[it]/a[it]-1)
9                                  +omegaL*(a[it]*a[it]-1));
10     //Segunda derivada do fator de escala
11     ddota=-dota*dota/a[it]*(omegaM/2-omegaL-omegaR);
12     a[it+1]=a[it]+dota*dtau+ddota*(dtau*dtau)/2;
13     //Observação:Usamos o tempo normalizado dtau=dt/H0
14 }}
```

D.2.3 Geração dos *Snapshots*

```
1      printf("\n Saving snapshot %d ....",snap);
2      a_up=1.02*z_s[snap+1];  a_dow=a_up/1.02*.98;   z_s[snap]=z;
3      double3 vel;
4       sprintf(buf,"%ssnap%d.dat",mFileOut.c_str(),snap);
5                  if (mBinOutput) f = fopen( buf, "wb");
6                  else f = fopen(  buf, "w");
7                  if (!mBinOutput)
8                      for (int i = 0; i < mNumBodies; i++){
9                          if (use_v2 and comov_on and !use_p){
10                             vel.x=Vel[i].x/a[it];
11                             vel.y= Vel[i].y/a[it];
12                             vel.z=a[it]*Vel[i].z/a[it];
13                         }else{
14                             vel.x=Vel[i].x;
15                             vel.y=Vel[i].y;
16                             vel.z=Vel[i].z;
```

```
                        }
            fprintf(f, "%f\t%f\t%f\t%f\t%f\t%f\t%f\t\n",
                 m,Pos[i].x, Pos[i].y, Pos[i].z,vel.x,vel.y,vel.z);
                }else{
                    for (int i = 0; i < mNumBodies; i++) {
                        if (use_v2 and comov_on and !use_p){
                            vel.x=Vel[i].x/a[it];
                            vel.y= Vel[i].y/a[it];
                            vel.z=a[it]*Vel[i].z/a[it];
                        }else{
                            vel.x=Vel[i].x;
                            vel.y=Vel[i].y;
                            vel.z=Vel[i].z;
                        }
                        fwrite((void*)&m, sizeof(double), 1, f);
                        fwrite((void*)&Pos[i], sizeof(double3), 1, f);
                        fwrite((void*)&vel, sizeof(double3), 1, f);
                    }
                }
                fclose(f);
                }
                printf("...done\n");
                snap++;}
```

D.2.4 Cálculo da Energia

```
função integrate_eΔ(){
    double ek_1=ek;
    printf("\n Integral a*aWda...");

    if(use_gpu)
    {
        printf(" Kinetic GPU");
        g_EKin(mNumBodies);
        cudaThreadSynchronize();
        cutilSafeCall(cudaMemcpy(pot, pot_g,sizeof(double) * ...
            mNumBodies, cudaMemcpyDeviceToHost));
        cudaThreadSynchronize();
        ek=0.0;
        for (int j = 0; j < mNumBodies; j++)
            ek += pot[j];
        ek=0.5*ek/mNumBodies;
        printf("..it %d snap %d..%f",it,snap,ek);
    }
```

```
18      else
19      {
20          printf(" Kinetic CPU");
21          ek=EKin();
22          printf("..it %d snap %d..%f",it,snap,ek);
23
24      }
```

```
1  ComputeEnergy()
2      double ek_1=ek,eΔ_it=0.0;
3      printf("\n Computing energy %d ...",snap);
4      printf("Potential gpu ...");
5      g_EPot(mNumBodies); //Calculo da energia potencial na GPU
6      cudaThreadSynchronize();
7      cutilSafeCall(cudaMemcpy(pot, pot_g,sizeof(double)
8                      * mNumBodies, cudaMemcpyDeviceToHost));
9      cudaThreadSynchronize();
10     epot[snap]=0.0;
11     for (int j = 0; j < mNumBodies; j++)
12         epot[snap] += pot[j];
13     epot[snap] =-0.5*epot[snap]/mNumBodies ;
14     printf("...ok");
15     ek=EKin();
16
17     printf("..kin %f  pot %f",ek,epot[snap]);
18     ekin[snap]=ek;
19     eΔ[snap+1]=eΔ[snap];
20     if(snap==0) a[it+1]=a[it];
21     if (comov_on){
22             if(use_v2 and !use_p){
23                 ekin[snap] = a[it+1]*ek;
24                 if(snap>0)
25                     eΔ_it=(ek+ek_1)*(a[it+1]-a[it])/2;
26
27             }else{
28                 ekin[snap] = a[it+1]*a[it+1]*a[it+1]*ek;
29                 if(snap>0)
30                 eΔ_it=(a[it+1]*a[it+1]*ek+a[it]*a[it]*ek_1)
31                                         *(a[it+1]-a[it])/2;
32             }
33             eΔ[snap]+=eΔ_it;    ek=ek_1;
34         }
35         else
36         {
```

```
37                  e∆[snap]=0.0;
38          }
39
40          etotal[snap] = epot[snap]+ekin[snap]+e∆[snap];
41      printf("...done \n");
42
43      }
```

D.2.4.1 Cálculo da Energia Potencial na GPU

```
1  __device__ double bodyBodypot(double sum_mG_rij, double3 ri, ...
       double3 rj){
2      double3 r;
3      double L_2=L*0.5;
4      r.x = fmod(rj.x+L_2-ri.x,L) - L_2;
5      r.y = fmod(rj.y+L_2-ri.y,L)-  L_2;
6      r.z = fmod(rj.z+L_2-ri.z,L) - L_2;
7      double distsqrt = r.x*r.x + r.y*r.y + r.z*r.z;
8      double mg_rij = mG*rsqrtf(distsqrt);
9      sum_mG_rij += mg_rij;
10     return sum_mG_rij;
11 }
12 }
```

D.2.5 Arquivos de Saida

```
1  <?xml version="1.0"?>
2  <NBody_Sim_out NumBodies="4096" Box="16"
3   Output_base_filename="./sim/outsnap"
4  Snapshots="41" File_binary="No"
5  Com_time="3.225312e+00" Proc_time="4.847838e+02"
6  Snap_0="30.0000" Snap_1="23.1598" Snap_2="18.1531"
7  Snap_3="14.6218" Snap_4="12.0346" Snap_5="10.0677"
8  Snap_6="8.5375" Snap_7="7.3156" Snap_8="6.3269"
9  Snap_9="5.5181" Snap_10="4.8400" Snap_11="4.2690"
10 Snap_12="3.7828" Snap_13="3.3629" Snap_14="3.0023"
11 Snap_15="2.6821" Snap_16="2.4063" Snap_17="2.1582"
12 Snap_18="1.9397" Snap_19="1.7433" Snap_20="1.5686"
13 Snap_21="1.4111" Snap_22="1.2675" Snap_23="1.1356"
14 Snap_24="1.0178" Snap_25="0.9094" Snap_26="0.8092"
15 Snap_27="0.7184" Snap_28="0.6343" Snap_29="0.5560"
16 Snap_30="0.4855" Snap_31="0.4179" Snap_32="0.3565"
17 Snap_33="0.2969" Snap_34="0.2425" Snap_35="0.1933"
```

```
18 Snap_36="0.1464" Snap_37="0.1005" Snap_38="0.0596"
19 Snap_39="0.0211" Snap_40="0.0028"
20 Out_file_energy="./sim/out.energy.dat"
21 Out_file_tracking="./sim/out.i.128.dat"
22 Out_file_scaler_factor="./sim/out.a.dat"
23 IC_info="./ICs/ics.bin" Simulation_config="sim/sim_cfg.xml"/>
```

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.