Computational Modelling of Galaxy Formation using FLAME GPU

Laurence William James

Supervised by Dr. Mike Stannett

COM3600

February 11, 2012

This report is submittted in partial fulfilment of the requirement for the degree of Master of Computing with Honours in Computer Science by Laurence William James

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s) (where possible). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Laurence James

Signature:

Date: February 11, 2012

Abstract

As hardware has become increasingly powerful, the doors have been opened for a wide range of more computationally intensive simulation procedures. In particular, agent-based modelling has seen a recent surge of interest in the fields of Biology and Economics. For this project we propose using an agent-based model to create an implementation of the classic physics problem of cosmological N-body simulation, in order to investigate the process of galaxy formation. Our simulation should be displayed as an evolving 3D graphic, to allow for easy interpretation of the data.

The GPU, with its massively-parallel architecture is well suited to the computation of N-body systems, and allows us to create high-fidelity simulations using mid-level consumer hardware. We therefore also investigate methods for performing our model's calculations on the GPU.

The primary aims of the project are to determine whether agent-based models are suitable for the investigation of physical phenomena, and to produce a realistic model of galaxy formation in doing so.

Contents

List of Figures

1 Introduction

From Aristarchus of Samos[1] to Fritz Zwicky[2], humans have been watching and analysing the movement of the stars. As science and technology progress we use ever more complex instruments to investigate the universe, and the resulting theories become increasingly precise. Recently, our models of how the universe formed under the influence of physical laws have been rigorously tested with a barrage of new data. Results from the *COBE*[3] and, more recently, *WMAP*[4] showed us that the Cosmic Microwave Background Radiation[5] (CMBR) isn't isotropic; we find tiny fluctuations in the levels of radiation (figure 1). These fluctuations comprise some of the best evidence we have for the idea that large-scale structures such as globular clusters, galaxies and galaxy clusters formed bottom-up around a small 'seed' of slightly higher matter density in the very early universe. This data led to a surge of interest and a revamping of our models, which are now being further tested with supercomputer simulations.

In Isaac Newton's *Principia*, he laid out the famous Laws of Classical Motion, from which the *Universal Law of Gravitational Attraction*, as well as Kepler's laws of planetary motion (which previously had only been shown empirically) he was able to derive. The relatively simple law of gravitational attraction is amongst the most profoundly important of classical physics; in words, it states that two masses are attracted towards each other across *any distance* with a force proportional to the product of their masses, and inversely proportional to the square of their distance. The constant of proportionality, G, is termed the *Gravitational Constant*, and $G \approx 6.674 \times 10^{-11}\text{N(m/kg)}^2$. Within a programming context, this is just a fixed global constant 'magic number'.

Comparing the force exerted by gravity to that of other fundamental forces (electromagnetism, for example), we find that gravity is, in fact, incredibly weak; if we take two electrons then the ratio of (attraction due to gravity):(repulsion due to electromagnetism) is given as 1.41×10^{-42}[Fey64]. This is at the microscopic scale, however. On the scale at which we live, our experience of the world is dominated by the effects of gravity doing what it does best — acting on a macroscopic level. In the universe, large collections of matter are electromagnetically neutral, so this force doesn't dominate their structure. Gravity however, is less discerning; it affects all matter, and its infinite range allows it to slowly pull objects across vast distances into the structures we see now.

We observe gravitationally bound systems in vastly different scales; from tiny satellites orbiting asteroids to galaxy clusters (superclusters are not gravitationally bound systems; the distances between their component clusters are much too vast). Those which are most relevant to the project include star groups/clusters, galaxies, and galaxy groups/clusters.

The formation of these large-scale structures, the expansion of the universe and the existence of the Cosmic-Microwave-Background radiation noted earlier is explained by the Λ-CDM[6] model, which this study aims to investigate with reference to the formation of Galaxies. To do this, we will be simulating a universe of particles under the effect of physical laws. For the sake of simplicity we will be working with a finite, 3D universe of Euclidean geometry, and so Λ — the Cosmological Constant (the 'dark energy' which controls the universe's rate of expansion) — is not relevant.

Structure formation has been already been extensively tested via simulation. Here, we populate a 'universe' with a set of particles and then compute the effects of the physical laws (gravity, electromagnetism, etc...) on these particles, and watch as the system evolves. Simulations of this nature are known as *Cosmological N-body* simulations, and are usually solved directly by integration on the systems of differential equations representing the particles' movement under the effect the relevant forces (usually just gravity; section 2.8). As such, the aim of this project is not merely to replicate preexisting studies with less time, resources and knowledge, but to use a technique previously unused in this field — *Agent-Based Modelling* (ABM) — where particles (in the N-body sense; we're not simulating individual particles of matter) are represented as individual autonomous *agents*. ABM is well suited to the simulation of systems where the location of agents is important, and where large systems are composed of many simpler

[1] Aristarchus was the first to propose a helio-centric model of the Solar-System; `www.astro.cornell.edu/academics/courses/atro201/aristarchus.htm`

[2] Zwicky first realised a large proportion of matter must be "unseen"[Tys01] (now referred to as 'Dark Matter')

[3] COsmic Background Explorer; `http://lambda.gsfc.nasa.gov/produce/cobe/`

[4] Wilkinson Microwave Anisotropy Probe; `http://lambda.gsfc.nasa.gov/produce/map/current`

[5] Radiation 'left over' from the recombination phase of the early universe.

[6] Lambda-Cold Dark Matter; See section 2.3

ones [MN06]; phenomena which we see in galaxies built of many star systems and globular clusters, as well as galaxy clusters comprising many mutually bound galaxies.

A final advantage of ABM is that it allows us to use unique agents at separate levels of abstraction in the same system. We therefore have the potential to have a set of agents of stars, some number of agents as gas clouds, others representing black holes, and so on, each with different rules rules governing their interaction with other objects.

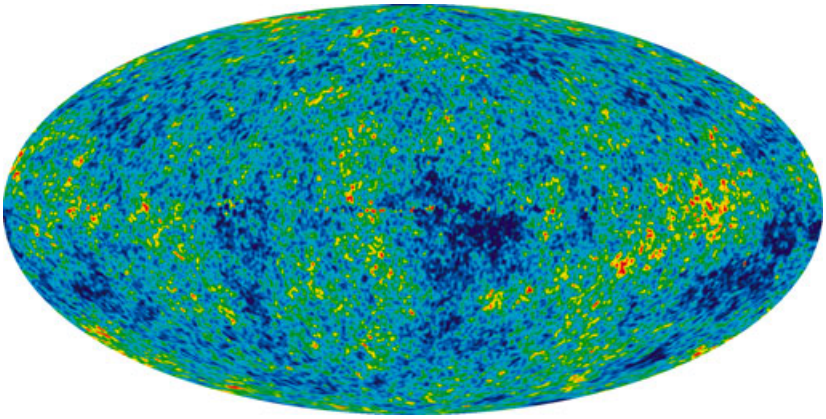

Figure 1: The Cosmic-Microwave-Background Radiation, showing the very slight temperature anisotropies in the early universe. The widest range of temperatures is $\pm 200 \times 10^{-6}$ Kelvin [St07]

2 Literature Review

2.1 Overview

Since this project spans multiple fields (modelling/simulation, computer science and physics), this literature review aims to cover the material relevant to each in a logical and structured manner. First, we cover information relating to the cosmological aspects of the project, mainly with reference to structure formation. Next, an explanation of some astronomy and the various smaller-scale astrophysical phenomena likely to have a bearing on the implementation. We then look at the aspects more directly related to computation; discussing Newtonian gravity specifically with reference to the calculations required for N-body simulations, and the different algorithms we can use for this. Next, we discuss existing tools and modelling frameworks likely to aid the project's implementation, before finally looking at previous studies which have carried out similar investigations.

2.2 The Early Universe

In order to fully grasp the background and extent of the project, it is important to have a basic understanding of the evolution of the universe as a whole, as described by the Λ-CDM 'standard model'. Not only does this provide us with a good mental framework from which to visualise what's going on in the simulation, but we can also appreciate the extent to which we have to simplify the physics in order to make this project feasible given the time constraints.

At the very beginning of time immediately after the Big Bang, we enter what is known as the Planck epoch, followed immediately by the grand unification and electroweak epochs. Not particularly relevant to this project, but of great interest to theoretical physicists, these are the time periods where, if our models are correct, the fundamental forces were combined; gravity broke away at the start of the grand unification, and the strong nuclear force broke away at the start of the electroweak epochs, respectively.

The inflationary epoch is the first with importance to structure formation. During this period space expanded rapidly — much faster than the speed of light. It is thought that during this period, quantum fluctuations 'froze in' [Lin95] and became *primordial fluctuations*; the density variations which we now (indirectly) observe as the anisotropies in the CMBMap results from COBE and WMAP (figure 1).

With the inflationary period concluded at $t \approx 10^{-32}$s, the universe is filled with an extremely high temperature *quark-gluon plasma* ("quark soup") — an extremely regular distribution of high-energy quarks[7] and gluons[8]. Over the following twenty minutes, the universe undergoes various changes responsible for the relative distributions of particles we see today; the plasma cools, and hadrons[9] form. During this period nucleosynthesis begins, and Hydrogen and Helium-4 nuclei[10] are produced as the protons and neutrons undergo nuclear fusion.

This period is extremely important for understanding where to start the model. With further cooling of the universe, cold dark matter around the primordial fluctuations provides a gravitational seed for matter density to increase around. However, for about $377,000$ years after nucleosynthesis the universe still only contains ionised plasma. It is only here, at recombination, that hydrogen and helium *atoms* start to form as free electrons are caught by their nuclei.

With the presence of Hydrogen and Helium gas, as well as small areas of increased density, we are finally in a position to observe structure formation. The amplitude of density perturbations grows as small quantities of gas are added until their rate of growth becomes non-linear, at which point gravitationally bound groups grow at an increasing rate via mergers due to collisions with others [CASS+94]. It is important to note that the dark matter around which these formed coalesces also, and remains around these objects as a *dark matter halo* [G+05]. As these clouds increase in mass and energy, the first stars form within them. Over time, as more mass is accrued and more stars form, these objects become the first dwarf and proto-galaxies. With further collisions and mergers between dwarf galaxies the system's mass and energy continue to increase, as star-formation continues in giant clouds of hydrogen and helium. Eventually, the first galaxies are formed, now observed by us as the very distant *quasars*.

2.3 Λ-CDM

The 'Standard Model' of Cosmology is known as Λ-CDM. Assuming time and space were created by a 'Big Bang' event, the Λ-CDM attempts to explain how the universe transitioned from this initial hot, dense, practically isotropic state through to the complex, structured Universe which we now observe. The model of structure formation was roughly explained in section 2.2, and so this section is more concerned with the assumptions and requirements of the Λ-CDM which are directly relevant to our simulation.

The mathematics of the Λ-CDM (the FLRW metric, Friedmann equations and cosmological equations of state) require a set of parameters, some of which will be incorporated into our simulation. These are:

Parameter	Symbol	Value
Hubble parameter	h	0.72 ± 0.03
Total matter density	Ω_m	$\Omega_\mathrm{m}h^2 = 0.133 \pm 0.006$
Baryon density	Ω_b	$\Omega_\mathrm{b}h^2 = 0.0227 \pm 0.0006$
Cosmological constant	Ω_Λ	$\Omega_\Lambda = 0.74 \pm 0.03$
Radiation density	Ω_r	$\Omega_\mathrm{r}h^2 = 2.47 \times 10^{-5}$
Neutrino density	Ω_ν	See Sec. 1.1.2
Density perturbation amplitude	$\Delta^2_\mathcal{R}(k = 0.002\,\mathrm{Mpc})$	$(2.41 \pm 0.11) \times 10^{-9}$
Density perturbation spectral index	n	$n = 0.963^{+0.014}_{-0.015}$
Tensor to scalar ratio	r	$r < 0.43$ (95% conf.)
Ionization optical depth	τ	$\tau = 0.087 \pm 0.017$
Bias parameter	b	See Sec. 1.3.4

Figure 2: Table of Λ-CDM parameters estimated from the most recent WMAP readings, taken from 'The Cosmological Parameters' Page 5 [LL10]

[7] Quarks are elementary particles, and form hadrons when two or three are held together by the strong nuclear force.

[8] Gluons are exchange particles which mediate the strong nuclear force.

[9] Formed from quarks, hadrons are split into baryons (including protons and neutrons), and mesons, which are extremely unstable.

[10] Protons and alpha particles, respectively.

Thankfully, our simulation simplifies most of these away. Those which will be relevant are the baryon density (the quantity of baryons in the universe); the dark matter density (the quantity of dark matter), and the density perturbation amplitude, which when used in conjunction with a Gaussian random field, should give an appropriate early-universe matter distribution.

2.3.1 Λ — The Cosmological constant & dark energy

First included in Einstein's 'Field equations for General Relativity'[Ein17], the *Cosmological constant* controls the rate of the expansion of space. The Λ-CDM model includes the concept of *dark-energy* — a physical realisation of the cosmological constant. This dark energy drives the expansion of the universe, and so every point becomes further away from every other point. This becomes quite an important simplification for our project — if we wish to deal with a fixed-space universe, we must ignore the universe's Dark Energy component. Many previous N-body simulations have modelled in this manner, and GADGET (section 2.7.4) includes options to use non-expanding universes.

This simplification helps us in a number of ways. Firstly, the mathematics dealing with particles' positions at each timestep is much simpler; we don't need to alter them to account for the universe's expansion. Secondly, the expansion would act in opposition to gravity — as gravity brings objects together, the expansion pulls them apart. In this case, much care would need to be taken in balancing these phenomena to still see structure formation and avoid universes dominated by entropy.

2.3.2 CDM – Cold Dark Matter

When we look at the rotation curves of galaxies we find that their rotational velocity does not decrease with distance from the centre as it should, given the galaxy's apparent mass distribution. To fix this discrepancy, additional mass is required in the form of a 'halo' around the galaxy [G$^+$05]. Since we can't observe this matter interacting with electromagnetic radiation we term it *dark*.

The variant of dark matter in the Λ-CDM is believed to be cold, that is, moving at slow (non-relativistic) speeds, and accounts for 23% of the mass-energy[11] in the Universe, or 83% of the matter in the universe (ordinary 'baryonic'[12] matter making up for the remaining 17%).

Unlike dark energy, dark matter is vital to our simulation — essentially providing a gravitational frame around which structures such as dwarf and proto-galaxies can form. Indeed, the Millennium simulation (section 2.8.1) didn't model baryonic matter at all — it investigated structure formation on such a large scale ($\approx 2 \times 10^6$ galaxies), that *only* dark matter particles were used [S$^+$05].

2.4 Stars

2.4.1 Star formation

If we want to simulate realistic galaxies, we're going to want them to contain stars. If we want our simulation to include stars, and we don't want to write them directly into the t_0 file[13], we need some method of creating stars from agents which represent gas or molecular clouds[14].

Now, the physics of star formation are easily complex enough to make another entire simulation project out of, so we need to implement some simplifications. Generally, the process of star formation begins with the collapse of a gas cloud whose gravitational potential energy is greater in magnitude than its kinetic energy. That is, the force acting inward due to gravity is greater than the outward force of pressure.

[11] According to the mass-energy equivalence, energy can be expressed as a function of mass, and visa-versa. By using mass-energy, we normalise everything to one unit.

[12] Baryonic matter is that which is made up predominantly of baryons; particles built from three quarks.

[13] When modelling a system like this we specify some initial conditions for the simulation, these constitute the t_0 file (section 2.7.2).

[14] A molecular cloud is an interstellar cloud (a collection of gas, plasma and dust), whose density is high enough to permit the formation of molecules. Stars are believed to form from the collapse of these.

To find if a cloud is going to undergo gravitational collapse, we check that it's mass is greater than the limit set out by the Jean's criterion [Mih]:

$$M_{cloud} > (\frac{5kT}{Gm})^{\frac{3}{2}}(\frac{3}{4\pi\rho})^{\frac{1}{2}} \tag{2.1}$$

with M_{cloud} the cloud's mass; m, the mass per particle; k the Boltzmann constant; T the average temperature of atoms in the cloud; G the gravitational constant; and ρ the cloud's density.

Thankfully, all these values should be relatively simple to calculate or approximate. If we work with agents representing gas clouds, their masses can be instantiated in t_0 (or calculated as the sum of their component 'particle' agents) and recalculated appropriately as a result of mergers. Similarly ρ we can assume to be uniform, then calculate $\rho = \frac{m}{V}$, with volume also instantiated to some realistic value at t_0. Finally, T can be estimated from the velocities of particles within the cloud.

2.4.2 Supernovae & Black Holes

Including supernovae and black holes (the final stage of evolution for particularly massive stars) could be interesting. When a sufficiently massive star ($M_{star} \geq 9$ solar masses[15] [Gil04]) exhausts the supply of elements which it can fuse for energy, it collapses. If the remaining core exceeds about 1.44 solar masses (the 'Chandrasekhar limit'), it implodes. This implosion causes a huge shock wave which could affect the simulation in a number of ways; possibly by 'blowing out' nearby gas clouds, or by inducing high levels of star formation in gas-rich areas. It should also prove relatively simple to display them as a particularly bright/high-energy source in our visual representation.

Black holes can play an important part in galaxy structure — it has recently been confirmed that our Milky Way contains a particularly large 'supermassive' black hole at its centre, and that this is likely the case for most, if not all, others [S+02, Ant93]. Given that our system ignores relativistic effects (section 2.6), the particular physics of black holes aren't necessarily relevant. The formation of galaxies with or around extremely large, central masses would therefore add additional weight to the reliability of our model.

2.5 Galaxy Types

We categorise galaxies according to the *Hubble Sequence* (known colloquially as the 'Tuning Fork', on account of its shape). The tuning fork sorts galaxies into four types, outlined here.

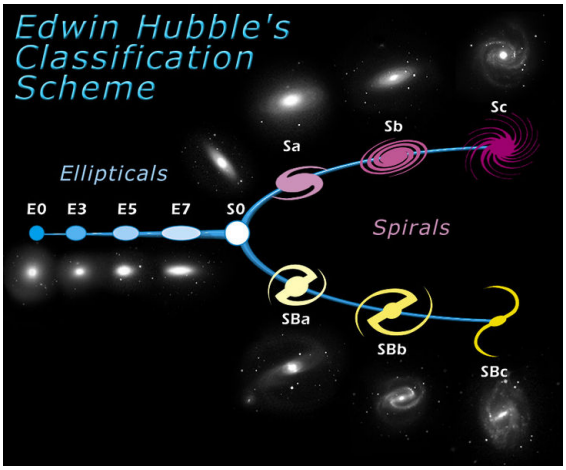

Figure 3: The Hubble sequence diagram [IN].

[15] Our sun is one solar mass.

2.5.1 Elliptical galaxies

Elliptical galaxies, located along the centre (the handle, if we're using the tuning fork analogy), are roughly spherical or ellipsoidal; essentially giant balls of stars and gas. Currently, we believe that elliptical galaxies are likely to be the result of collisions or mergers between other galaxies, where strong tidal forces disrupt the spiral. This theory is supported by other properties of elliptical galaxies; namely that the stars in them are much older, and that they appear to be less common in the early universe, which we observe with the Hubble deep field[16] [Mao97]. Additional support is given by the facts that we see them more often among dense galaxy clusters, and that ellipticals comprise the majority of the largest known galaxies (the huge ellipticals being the result of multiple galaxy collisions). Despite this however, we do see a great range of sizes in ellipticals — indeed, with most dwarf galaxies also classifying as such. [AM+08]

2.5.2 Spiral galaxies

Positioned on the two branches on the right of the diagram, spiral galaxies are extremely flat (relative to width) disks, with their matter rotating around a dense central bulge of older matter and stars, usually surrounding a supermassive black hole. We split spirals galaxies into two classes: (regular) spirals, and *barred* spirals. Barred spiral galaxies (the diagram's lower branch) contain a central bar of many stars.

Spiral galaxies take their name from the long 'arms' of matter we observe extending from and rotating around them, thus forming the distinctive structure. The spiral arms of different galaxies are 'wound' in varying degrees of tightness, from long and spread out (the most extreme cases found in barred spirals; SBc in figure 3) to those with many, tightly wound arms, such as the pinwheel galaxy (appendix A.1). The arms contain much higher levels of new star formation than other areas of galaxy, and so have a distinctive blue hue.

The majority of galaxies observed in our universe are spirals [Dre80] (our own Milky Way is believed to be a barred spiral). As such, if by the conclusion of this project we are able to observe the formation of a reasonably standard spiral galaxy, we could certainly consider this at least a partial success.

2.5.3 Lenticular galaxies

At the centre of the fork ($S0$), lenticular galaxies provide a bridging point between the elliptical and spiral classes. Like spiral galaxies, they're relatively flat disks with a central bulge. Like elliptical galaxies on the other hand, they don't appear to have any spiral arms, and there is very little active star formation — the majority of stars are old and dim — giving the galaxy a faintly reddish hue more akin to that of the ellipticals than the blue seen in spiral arms.

Lenticular galaxies are fairly uncommon [Dre80], and probably won't have much impact on our simulation. The main theory for their formation, as one might expect, is that they are simply spiral galaxies which have grown old and faded, with shock waves blowing out the interstellar matter used for star formation [BASM06]. This is somewhat contested by the notion that the bulges are too large and the galaxies too bright for this to be the case, and so they must have formed via mergers [BKW+07].

Whatever the outcome, both theories require us to accurately model the galaxy's evolution across long periods of time, and so we probably won't see any in the simulation; we're investigating the *formation* of galaxies, not the evolution of those which are extremely old.

2.5.4 Irregular galaxies

Included later as a catch-all for those galaxies which defy classification to any of the above three sections, irregular galaxies are rare and come in all shapes and sizes. These are generally thought to be the result of spiral or elliptical galaxies distorted by the gravitational pull of a near-neighbour. We may find that if our mathematical simplifications are too great, or our simulation's resolution is too low, then all galaxies simply appear irregular, with no governing spin or overarching structure.

[16] The Hubble deep field is a composite image of a section of sky, taken with an extremely long exposure. The objects observed in the Hubble deep field are incredibly distant, and therefore existed when the universe was very young.

2.6 Newtonian Gravity

Although technically superseded by Einstein's *General Theory of Relativity*, we can assume Newtonian gravity to be appropriate for use in our model as the mechanism behind which structure formation takes place. This assumption is used based on the success of a significant number of non-relativistic cosmological N-body simulations. Newtonian gravity will therefore govern the motion of our objects as they move through space, and so we need some accurate method of calculating this.

2.6.1 Formulae and calculations

For objects simplified to point-masses, Newton's Universal Law of Gravitational Attraction is given by:

$$F = \frac{Gm_1m_2}{r^2} \tag{2.2}$$

with F the force, G the Gravitational Constant, m_n the mass of body n, and r the distance between the two objects.
Substituting Newton's second law, $F = ma$, into 2.2, a body of mass m_1 accelerates towards a body of mass m_2 with acceleration:

$$a = \frac{Gm_1m_2}{r^2m_1} = \frac{Gm_2}{r^2} \tag{2.3}$$

Equation 2.3 therefore gives the acceleration due to gravity on some body from an object. When resolving the resultant force on a body, i for any number of j objects, we simply take the vector sum of all forces. Generalising 2.3 to many objects in three dimensions, we get:

$$\ddot{\mathbf{r}}_\mathbf{i} = G\sum_{i\neq j}\frac{m_j(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3} \tag{2.4}$$

with $\ddot{\mathbf{r}}_\mathbf{i}$ the acceleration of object i, and $\mathbf{r}_n$ the position in space of object n as a 3D Vector

From basic classical mechanics we know that $\int \mathbf{a}(t)\,\mathrm{d}t = \mathbf{v}(t)$ and $\int \mathbf{v}(t)\,\mathrm{d}t = \mathbf{r}(t)$ (with $\mathbf{a}(t)$ the acceleration at time t, $\mathbf{v}(t)$ the velocity at time t, and $\mathbf{r}(t)$ the displacement at time t)
Expanding these integrals $\mathrm{d}t$ with some small timestep $\Delta t = t - t_0$ gives:

$$\mathbf{v}_i(t) = \mathbf{a}_i\Delta t + \mathbf{v}_i(t_0) \tag{2.5}$$

$$\mathbf{r}_i(t) = \frac{1}{2}\mathbf{a}_i\Delta t^2 + \mathbf{v}_i(t_0)\Delta t + \mathbf{r}_i(t_0) \tag{2.6}$$

with $\mathbf{v}_i$ the velocity of object i; $\mathbf{a}_i$ the acceleration of object i as calculated by 2.4, and (t_0) the previous timestep.

2.6.2 The N-body problem

With Δt sufficiently small, we can assume that the acceleration remains constant for this period. This assumption is of fundamental importance to simulations of this type; for each timestep we calculate the acceleration for every particle. This acceleration is then used in 2.5 and 2.6 to find the vector displacement for each particle at the new time $t = t_0 + \Delta t$. The positions are then updated as a result of the displacement, and the iteration is concluded. We repeat this until some suitable number of timesteps has passed.

Any system which attempts to simulate the gravitational attraction between a set of bodies (usually with a method similar to that described above) is termed an *N-body Simulation*. (Examples of previous N-body simulations include the *Millennium run* and *Bolshoi Simulation*, and are given in section 2.8.)

N-body simulations become problematic, however, as we generalise 2.4 to find the resultant positions for *all* points at *all* times, given some initial positions and velocities. This problem stems from the fact that any N-body system becomes chaotic with $N > 2$ [BP98]. This is termed the *N-body problem*, and is expressed mathematically as when we wish to solve 2.4 for $i = 1, \ldots, N$

When performing N-body simulations rather than finding some general solution we opt to directly calculate the positions of each body (hereafter referred to as 'particles'), for a number of discrete timesteps. With enough timesteps the particles' movements appear continuous. A number of algorithms exist for computing these particle movements, outlined below.

Information on Barnes-Hut, particle-mesh and P^3M approaches (including algorithms 2 and 3) taken from Lindholm's N-body algorithm seminar: [Lin09].

All-pairs (also 'brute force', 'pairwise' or 'particle-particle') approaches are the most basic. As you might expect, with an all-pairs N-body computation algorithm we simply calculate 2.4 for every particle in the system. That is, for every particle we compute its acceleration due to every other particle. This is obviously rather computationally expensive — with N particles, the algorithm has $\mathrm{O}(N^2)$ complexity. This potentially restricts the number of particles we can model with, thus reducing the simulation's fidelity.

Tree-based methods treat groups of extremely distant objects as point masses located at their combined centre of mass. With attraction decreasing as the square of the distance, it makes little sense to put the same effort into treating a star in some far-off galaxy the same as one which might be only a few light-years away. Instead, we treat the whole *galaxy* the same as our hypothetical star, reducing it to only one body with its mass and position given as the total and centre of mass for all combined particles, respectively.

To do this, we split space into cells in a tree structure, and so create an octree of our simulation space (quadtree when working with 2 dimensions) by recursively subdividing our volume. We continue until each particle is enclosed in a leaf node (figure 4; more detailed version in appendix A.2).

With the tree constructed, we can generalise sets of particles to higher-level nodes. For example, in figure 4 when calculating the force for a particle such as g, we might combine all particles under 1.2 to just one; giving some new particle, say h with mass $M_h = M_b + M_c + M_d$, and position their centre of mass.

Obviously, we can do this for any node based on our distance threshold. Particle a, for example, might use e individually, but combine f and g to one. The set of particles 'belonging' to a node is those which are at any depth below that node. So, the set of particles in 1.3.3 is a subset of those in 1.3; is a subset of those in 1.

The *Barnes-Hut simulation* introduced the tree-based method described above, and optimally reduces complexity from $\mathrm{O}(N^2)$ to $\mathrm{O}(N \log N)$ [BH86]. It contains three steps:

1. Compute octree
2. Traverse from leaves to root, computing total mass and center of mass for each parent node
3. For each particle, traverse from root calculating the appropriate force

Algorithm 1 Function to recursively subdivide cells until all particles on leaves. (As in fig 4, appendix A.2)

```
split current cell into 8
for all new cells, x do
  if x contains >1 particle then
    Call this function recursively on x
  end if
end for
```

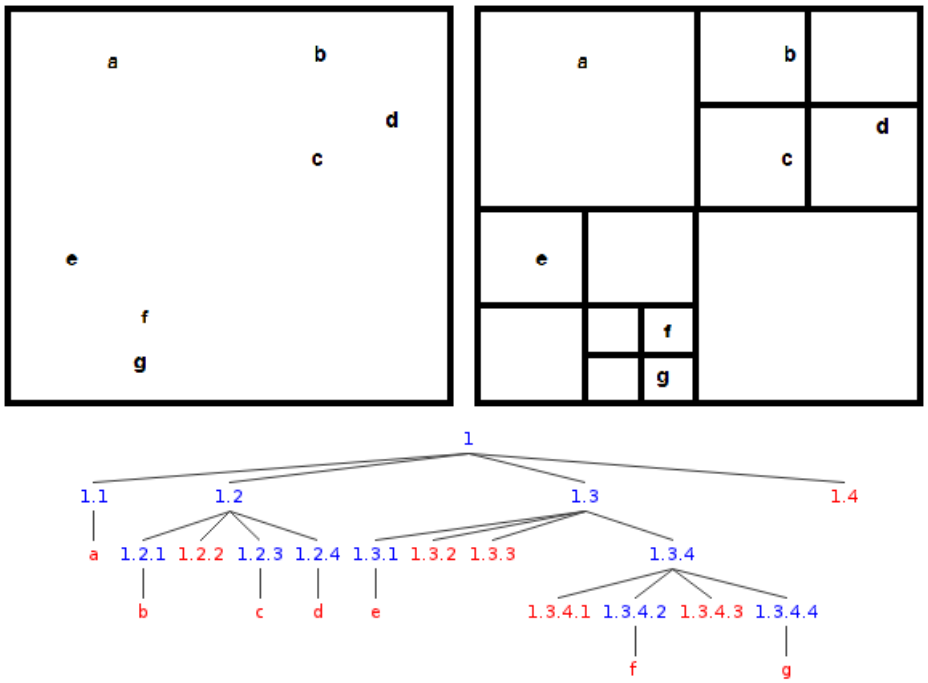

Figure 4: Diagram showing initial particle space (particles as a,b,c...); particle space once we've performed the subdividing, and the associated quadtree. Tree labels calculated top-left to bottom-right. Author's own work.

Algorithm 2 Function to compute step 2 of Barnes-Hut algorithm:

if leaf node **then**
  return
else
  for all children, x **do**
    call this recursively on x
  end for
end if
mass = 0
distance = 0
for all children, x **do**
  mass += mass of x
  distance += (mass of x $\times$ position of x)
end for
distance = distance/mass

When calculating the force for step 3 we find the ratio, θ, where $\theta = \frac{D}{r}$ (with D the size of the current node and r the distance to some other node B) If $\theta <$(some limit), we use B as a generalisation for its children. Otherwise, we compare again with each of B's children.

Algorithm 3 Function to compute step 3 of Barnes-Hut algorithm:

```
if (leaf node) OR (((size) / (distance(B))) < θ) then
  return force between this and B
else
  force=0
  for all children, x do
    force += (this recursively on x)
  end for
  return force
end if
```

Particle-Mesh based optimisation: While our previous descriptions use discrete timesteps to measure time, particle-mesh methods expand upon this idea by also measuring *position* discretely.

We first split the universe into a grid ('mesh'), which gives us our fixed positions, and then solving 2.7 at each meshpoint gives us the gravitational potential at that position:

$$\nabla^2 \Phi = c\rho \tag{2.7}$$

with c a constant; Φ the gravitational potential energy and ρ the mass density, we can now define some U as the force, with: $\nabla \cdot U = c\rho$;

For the purposes of calculating the density function, ρ, particle masses are assigned to cells by way of either the 'nearest gridpoint', where each particle's entire mass is given to the closest grid position, or by 'cloud-in-cell', where the particle's mass is distributed over the eight neighboring cells by a weighting function determined by the particle's proximity to each cell.

Particle-Particle/Particle-Mesh: The traditional particle-mesh algorithm is, however, a rather poor method[Lin09] for computing short-range interaction. Therefore, the *Particle-Particle/Particle-Mesh* (P^3M) algorithm was devised. (P^3M) essentially serves as a compromise by using a PM-based algorithm for long-range forces, and pairwise comparison for short-range forces. When using P^3M we first calculate the overall force on our particle with a standard PM method, and then subtract from this the total force given by objects within some radius around our particle. The forces for each particle within this radius are then individually calculated with pairwise comparison as described in section 2.6.2, and summed to give the final result.

2.7 Agent-models & software packages

The general concept behind agent-based modelling is to represent each individual actor in some situation as an autonomous *agent*. Each agent is an instance of some agent type, and their behaviour is governed by a set of rules and variables defining that type. By then populating a system with a number of these agents, their low-level interplay prompts changes at the macroscopic level; this generates the system's *emergent behaviour*. Furthermore, we can have higher-order agents defined as collections of individual, lower-level agents. The low-level agents will then still behave according to their own rules individually, but might have a different role in the context of some greater system. We find ABMs are becoming increasingly used in biology, economics and social sciences [MN06]; a key strength being that we only need to understand the behaviour of what we're simulating on an individual level. If the interaction of two cells is well understood, but the interaction of two-hundred thousand less so, then we only need to implement the low-level knowledge in our system and then observe the resultant behaviour.

By using an agent-based modelling package to control the actual execution of our simulation, we avoid having to invest a significant portion of time into creating bespoke software. This therefore allows us to focus all our efforts on producing a more accurate simulation, incorporating more phenomena or removing mathematical simplifications as appropriate.

Rob Allan's 2009 paper comparing various agent-based modelling and simulation tools [All09] outlined forty-three different packages, each with different approaches, capabilities and specialities. Unfortunately, there is very little general comparison available, and so choosing which package to use could

prove quite a task. We look in detail at two of these; 'Swarm' and 'FLAME'[17]. Swarm was the first generic software tool for creating and executing agent-based models, and is the most well-known and widely used [All09]. FLAME, on the other hand, was developed 'in-house' at Sheffield, and so plenty of resources and information are available. Importantly, the 'FLAME GPU' extension is also available to us, allowing for extremely efficient execution of models containing many agents.

2.7.1 Swarm

Much of this section is based on the Swarm 'documentation set' book [Gro04].

Swarm was developed in 1994 by the Santa Fe institute as the first re-usable agent-based modelling and simulation tool, and management and development has since been taken over by the 'Swarm Development Group' [All09]. Although described as being "Probably still the most powerful and flexible simulation platform" [All09], every aspect of the system to be modelled must be specified in a mixture of Java or Objective-C, and it relies heavily upon the modeller having an understanding of the concepts of object-orientation. The platform is therefore not particularly accessible to the scientific community as a whole, who may not necessarily have the computer science background required to get over the initial learning-curve.

General concepts: The most important concept is that of the 'Swarm'. Any swarm is composed of a collection of objects, and a 'schedule of activity' which runs over those objects. The objects we place in a swarm represent everything in our simulation which we use to model with. For a simple flocking model, therefore, our swarm objects may be a collection of 'boid'[18] agents, a representation of the world they're interacting in, and a store of their positions. The schedule of activity describes how time influences the collection of objects. The schedule of activity for flocking, therefore, would be to tell each boid to change its position/course, and to update the object storing positions.

Note at this point that Swarm is designed to be hierarchical; we can have swarms where one of the swarm's objects may be an entire, lower-level, swarm. Such a structure is extremely powerful and enables us to implement complex systems at various levels of abstraction. For example, we could model an ocean as a swarm composed of many schools of fish. Each school could then be its own swarm, containing individual fish agents.

Swarm simulations contain two swarms: the model swarm and the observer swarm. Combined, they contain all the code and information necessary for execution.

Model Swarms: The model swarm contains everything in the world which is involved in the simulation. All the computation relating to the actual model takes place here, so these are considered the heart of execution. When we wish to run a simulation, we pass the model swarm parameters upon its instantiation. These parameters control the model's initial setup, including any agent distributions or any simulation-specific variables (which type of N-body algorithm to use, for example).

With reference to this project, input parameters might be the number and distribution of 'star' agents, along with their masses and initial velocities. The output of a model swarm is the system's observables — how whatever we're investigating changes with time. In our project, therefore, the outputs from execution of a model swarm would be the agents' positions in space.

Observer Swarms: We use a second swarm; the 'observer swarm', to interpret the output of our model. The observer swarm contains the model swarm in its object collection. Other objects in the collection automatically pass parameters to the model swarm, and read the resultant output. The observer swarm's activity schedule handles processing of the generated data; taking results from the collection and drawing them to a graph or writing to a file, for example.

The inputs of an observer swarm control how we process the data — what information are we interested in and how should we deal with it. The outputs are obviously the processed results, and we have a choice between using *graphics mode* (where results are displayed to the user graphically, as graphs

[17] Flexible Large-scale Agent Modelling Environment; `www.flame.ac.uk`

[18] Craig Reynolds' initial work on flocking using agent-based algorithms is considered seminal work in ABM, here individual generic agents which could represent fish, birds, insects, etc... are termed 'boids'. Further information at `www.red3.com/cwr/boids`

or 2D visualisations) or *batch swarms*, where many simulations are run and we store all the raw data for later statistical analysis. Importantly, Swarm does not support 3D visualisations, and so we would need to add this functionality ourselves by extending the graphics mode.

Swarms and OOP: Swarm models are created with Objective-C, although an alternate version 'SwarmJava' uses a Java Objective-C library to allow the creation of models using Java with little deviation from original Swarm syntax. We can therefore create a complete Swarm model with either Objective-C, Java, or a combination of both.

Model and observer swarms are implemented as interfaces of the generic 'swarm' superclass. We define their parameters to be any user input relevant to the simulation, and write our own methods for getting data or returning output as required. Agents are interfaces of 'SwarmObject' — another generic superclass which has very little behaviour of its own. We specify all the behaviours for agent interaction in these classes, and so it is here that the majority of coding effort will have to be spent. As all agents are built as objects of their own class, we also define their variables here. The most important function is the *step* method, which is called by the model schedule to increment a timestep. Here we provide the code to calculate how the agent's variables change as a result of its state and/or environment during that period of execution, and these results are updated for the next timestep.

2.7.2 The FLAME framework

> *"FLAME is a generic agent-based modelling system which can be used to development applications in many areas. It generates a complete agent-based application which can be compiled and built on the majority of computing systems ranging from laptops to HPC super computers"*[19]

FLAME breaks the model down into a set of interlinked components; we use X-machines to model the agents' formal behaviour; XML to specify the agents' structure (including any variables), and an initial agent configuration file; and C to write the transition functions which compute agent interaction.

X-machines: We define the operation of our model with a formal model of computation known as *Stream X-machines* [Lay93]. A stream X-machine operates as an *extended finite state machine*, with the additional property that it has a memory which can be written to and read from via regular transition functions.

A full explanation of state machines is not required for the purposes of this literature review, but we will be using them to define our agents for the early stages of implementation. FLAME considers one timestep to have passed when every agent in the simulation has moved from its *start* to its *end* state.

XML: FLAME's models are written in XML[20], the particular schema for which is available at `http://flame.ac.uk/schema/xmml_v2.xsd`. By using XML for simple tasks such as the definition of agents, the system is significantly easier to manage and understand (the majority of information can be easily read by anyone who understands the XML 'tag' concept), and we don't need to worry about any obscure C syntax. On the other hand, the complex core of operations — the agent's transition functions — *are* written by us in C, allowing for far greater flexibility.

Input (t_0): In FLAME, we define the initial conditions of our system in an XML file named '0.xml'. This file contains each agent's initial variables, and is read by FLAME when we first execute the simulation. This file is formatted the same as the subsequent numbered timestep files, an example for which is shown in appendix A.3

Initial investigation suggests that the matter distribution of the early universe is well-represented by a 3D Gaussian distribution; if we use FLAME, one aspect of this project will be to create a small program which is capable of automatically producing an appropriate 0.xml file.

[19] `www.flame.ac.uk`

[20] Extensible Markup Language; `http://www.w3.org/XML/`

Output: FLAME runs its simulation for a set number of n timesteps, n defined by the user, with one timestep concluding when every agent has moved from its start to end state. After each timestep, FLAME outputs a numbered file x.xml, (x the iteration number) which represents all agents' variables in the same fashion as our initial 0.xml.

Obviously, we want a more meaningful representation of the evolving system than a sequence of tens of thousands of numbered XML files. The most obvious and intuitive method of doing this is to output the results as a 3D visualisation, redrawing the positions of agents after each timestep as they move across space. Various strategies to further improve the visual spectacle of this will be available; perhaps rendering agents in a variable colour from red–blue to represent different levels of energy.

To achieve this in FLAME a third-party program will need to be developed which takes the numbered XML files as input, parses and processes these, then draws all the information for each timestep to the screen, making use of some 3D graphics API such as openGL or Direct3D. Dr Simon Coakley, who initially developed FLAME, has recently produced the FLAMEVisualiser[21]; a tool written in C++ which does exactly this. FLAMEVisualiser takes FLAME output files and then draws each one as a frame in a video, representing agents with simple objects of different colours in their appropriate position in 3D space. FLAMEVisualiser provides a GUI with which to choose from various rendering options (figure 5), but given that this is fairly early-release software, the options for customisation aren't as extensive as we will require. It will therefore be necessary to make some adjustments to slightly tailor the video's appearance to exactly what we wish.

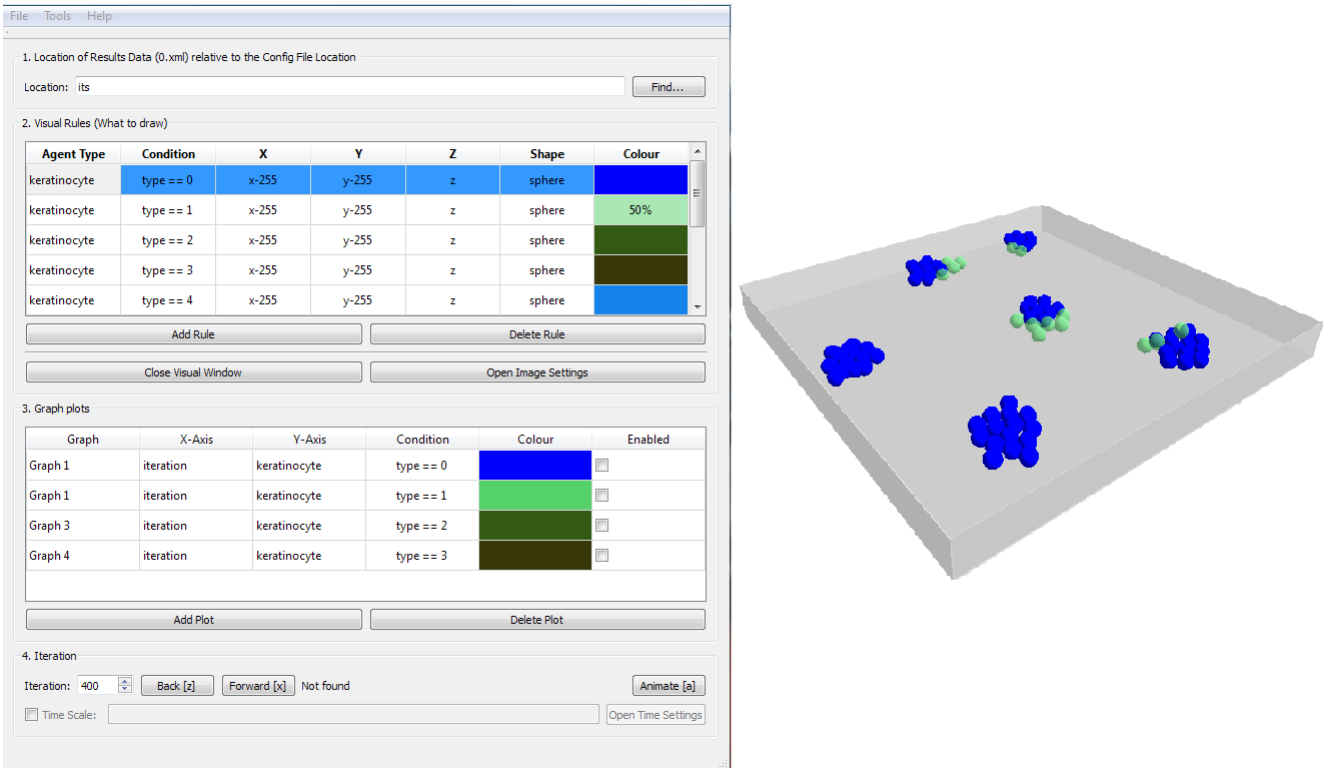

Figure 5: Left: FLAMEVisualiser config controlling rendering for an example visualisation (right). Shows the rules governing which agents are drawn, as well as how they're drawn. Shape options include sphere, cube and point (spheres shown). Agent colour can be any standard RGB value, with an optional alpha channel. Also shows the current iteration (timestep) being drawn, as well as rules for automated graph production. (Author's own work, using FLAMEVisualiser's provided example model).

[21] `http://ccpforge.cse.rl.ac.uk/gf/project/flame/frs/`

2.7.3 FLAME GPU

> *"FLAME GPU is a high performance Graphics Processing Unit (GPU) extension to the FLAME framework. It provides a mapping between a formal agent specifications with C based scripting and optimised CUDA code."*[22]

An extension to FLAME, FLAME GPU utilises NVIDIA's[23] CUDA[24] framework for parallel processing on the GPU to provide a more efficient method of computing agent-based simulations. Having been designed in hardware to perform simple calculations at an incredible pace (the ray tracing algorithm, for example, benefits greatly from parallelisation), the GPU is built with very many processing cores.

This (parallel) design fits perfectly with our specification of an ABM; a system composed of many *autonomous* agents. The fact that agents act autonomously is of key importance — if their operations don't rely on the results from previous calculations, then they can all be computed in parallel, reducing computation time potentially from tnN, (with t the time taken to perform one agent's calculation, n, the number of agents and N the number of timesteps), to just tN. Realistically this won't be the case; our simulation will contain hundreds of thousands of agents, and most consumer GPUs have between 200–500 cores.

The advantages of FLAME GPU (and GPU-based computation in general) are most substantial when we are dealing with very large numbers of agents performing simple calculations, and therefore the relative speedup over FLAME is highly dependent on the model we're simulating. Since our agents' interactions are simple mathematical operations, we can compare its likely performance well with that of the 'circles' model[25]; a graph of the performance of which over FLAME we show in figure 6. Here we see FLAME GPU consistently performing up to 100% faster than FLAME for population sizes of up-to 2^{16} on a GPU with single-precision, and up-to 300% faster on a double-precision GPU.

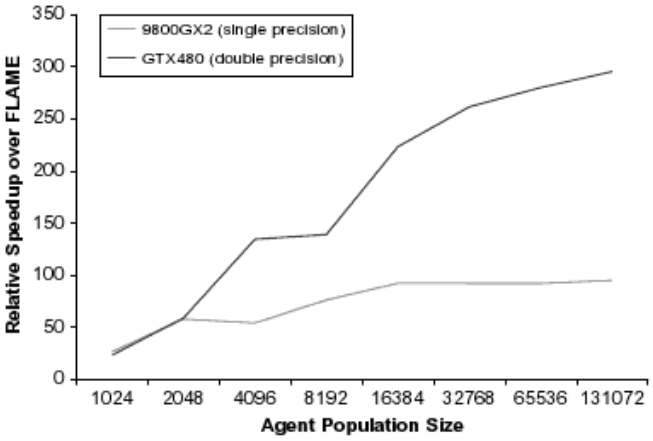

Figure 6: The relative speedup of FLAME GPU over FLAME for one timestep of the 'circles' model using FLAME GPU's standard 'brute force' message calculation. Computed using FLAME on a single dual-core AMD Athlon 2.51GHZ, and FLAME GPU with a 9800GX2, and GTX480 [RR11]

Output Since agent information is stored on the GPU during execution, visualisation of our output is much easier for real-time rendering than it would be were we using the CPU (where data would need to first be sent to the GPU). FLAME GPU provides two methods for dealing with output, chosen by command line parameter. In *Console Mode*, it behaves the same as FLAME; spitting out numbered xml timestep files, which can later be interpreted by a different program. In *Visualisation Mode*, FLAME

[22] `www.flamegpu.com`

[23] NVIDIA design GPUs and Graphics cards; `www.nvidia.co.uk`

[24] Compute Unified Device Architecture; `http://www.nvidia.com/object/cuda_home_new.html`

[25] 'Circles' is a basic force-resolution model included in FLAME GPU for benchmarking purposes.

GPU draws the results of the simulation in 3D space in real-time, representing agents as spheres [Ric11]. This eliminates our need to use the 'FLAME Visualiser', or to create our own program. When built in visualisation mode, entire FLAME GPU models are compiled down to an executable file which takes a command-line parameter defining the location of a '0.xml' initial setup file. It is therefore possible to launch models built for FLAME GPU by executing a single batch-file, whereupon the simulation will run indefinitely, rendering itself in real-time until the window is closed. This, combined with the intuitive data representation in visualisation mode, allows simulations to be easily distributed and the results observed on different machines.

FLAME GPU does not currently allow support for multi-GPU systems such as cards in SLI[26], or grid systems, although work is being undertaken to change this.

2.7.4 GADGET-2

GADGET-2 (GAlaxies with Dark matter and Gas intEracT (v2), hereafter referred to just as GADGET[27] is a suite of code designed to execute N-body simulations by modelling particles as collisionless fluids and using smoothed-particle hydrodynamics to calculate their motion. Although not an agent-based modelling package, GADGET is included in this discussion anyway due to its relevance to the project. Because it has been designed explicitly for N-body simulations, the software has been well optimised for the task in hand, and includes a combined Tree/PM optimisation algorithm (section 2.6.2), using a particle-mesh for anything longer-range than a certain tree distance.[Spr05]

GADGET also includes several different models of the universe we can use as our simulation-space. These are:

- 'Newtonian' space (as in our model);
- 'Cosmological' (expanding) space, with physical coordinates;
- 'Cosmological' space with co-moving coordinates.

We can also use any of these with a selection of optimisation strategies including TreePM and periodic boundaries [Spr05].

Output: GADGET, like FLAME, has been built for massively-parallel systems, and output is created as a series of 'snapshot' files, conceptually similar to FLAME's ordered timestep files. Unlike FLAME however, particle data is stored in a binary, rather than ASCII, representation [Spr05].

Two pieces of software are available to create visualisations of GADGET data, though their performance is unknown. These are SPLASH[28] and IFrIT[29].

Both the Millennium simulation and the Bolshoi simulation (section 2.8.1) used versions of the GADGET code which had been modified to improve efficiency when dealing with such vast models. GADGET basically provides a generic framework for developing N-body simulations; essentially performing the same function that FLAME and Swarm do for agent-based models.

2.7.5 Software summary

With a number of different software packages outlined, we need to compare and contrast their relative features to arrive at some form of conclusion.

FLAME, FLAME GPU and Swarm are all generic, agent-based frameworks which allow us greater flexibility at the cost of a larger memory and computational overhead per particle when compared to systems such as GADGET. GADGET, by contrast, models particles as collisionless fluids and calculates their positions by way of general formulae. Beneficial features of ABM for N-body simulations such as this include the ability to model with layers of hierarchical agents written directly into our code

[26] Scalable Link Interface; NVIDIA's consumer technology for connecting multiple graphics cards for single output; `http://www.nvidia.co.uk/object/sli_technology_overview_uk.html`

[27] `http://www.map-garching.mpg.de/gadget/`

[28] A generic visualisation tool for scaled particle hydrodynamics; `http://users.monash.edu.au/~dprice/splash/`

[29] Ionization FRont Interactive Tool — a generic tool for visualising any 3D data set; http://sites.google.com/site/ifrithome/

(galaxy agents composed of star agents composed of gas agents...), and that we don't have to worry about the mathematics of the large-scale system; by applying the rules of particle-interactions for agents on an individual level, the whole-system's behaviour should be *emergent*. GADGET, however, has the benefit of being designed specifically for our problem domain — cosmological N-body simulations. Such specialisation is obviously in direct contrast with FLAME and Swarm; while GADGET is highly specific, these are highly generic — they can be used to model any system, and the level of detail achieved is determined by the time available to create an appropriate model within the framework.

Of all our software tools, FLAME GPU is the only one which allows computation on the GPU. Currently, the software only supports processing on one device, although work is being undertaken to extend it to multi-GPU systems. Executing the project on the GPU is an extremely attractive idea; As the video-game industry drives the development of increasingly powerful GPUs at consumer-level prices, GPGPU[30] is rapidly being adopted as a new paradigm for scientific computing. As well as being a novel area of research, the type of calculations to be performed are perfectly-suited for the GPU, and it also allows us easier visualisation (section 2.7.3). FLAME and GADGET operate in a similar fashion; both will run on many architectures, including individual workstations as well as massively-parallel clusters, HPCs and supercomputers. Swarm, however, does not support parallel execution across distributed systems. This could pose a considerable problem given the potential time required to complete a timestep for large agent populations.

As a result of being developed in-house, we have access to significantly more documentation and direct support for FLAME and FLAME GPU than for any of the other packages. If, as is often the case when using fairly small, specific tools, the provided documentation doesn't adequately cover some obscure function-call or error message, then the software's developers are available just down the corridor! User guides for Swarm and GADGET are both available, although the Swarm guide is considerably more comprehensive.

Finally, the format of data output by the software varies between packages, and as producing a graphical visualisation of the simulation is a key objective, this is an important area of discussion. All tools have methods available for visually displaying their data, but to varying degrees of suitability. GADGET is the only tool described which does not have a first-party visualiser, and FLAME GPU is the only one which will render simulation results in real-time. Real-time rendering doesn't make a huge difference, but is helpful for debugging or improving the model — instead of having to run the simulation for a period of downtime and then build a video from the output, we can watch problems develop as they occur, and change them then and there. We also have the option to build some bespoke visualisation software by parsing the raw output data and drawing it with some 3D graphics API in a suitable language, such as Java or C. This gives us the greatest flexibility in exchange for the significant time-investment of actually coding it. GADGET, with its binary output, is likely to prove harder to do this for than the other packages (which use ASCII), since we'll need to perform all the necessary conversations first.

2.8 Similar investigations

2.8.1 N-body simulations

Many projects have used supercomputers to simulate the evolution of stars, galaxies, and even universes. For the sake of brevity, I shall only outline those which are of the largest scope, and those which are the most relevant.

The Millennium Simulation [S+05] (also known as the Millennium Run) can, in a way, be regarded the as 'poster child' of large-scale N-body simulation. Developed by the Virgo Consortium for Cosmological Supercomputer Simulations, the Millennium Run modelled with $\approx 1 \times 10^{10}$ dark matter particles, each with a mass of $8.6 \times 10^8 h^{-1} M_\odot$ [31] The project primarily modelled the evolution of the very large-scale distribution of matter, but galaxy formation obviously plays a role in this domain.

The results from the Millennium Run were released in 2005, and images of the simulation are widely used. The full data are available for scientific use by registered users via a publicly interfaceable SQL

[30] General-Purpose computation on Graphics Processing Units.

[31] $\odot$ is used to represent one solar mass, $\approx 1.99 \times 10^{30} kg$

database, or one $\frac{1}{512}$th of the data are available for immediate public use.[32] Data from the Millennium simulation has been used in over 400 publications to date.[33] One conclusion drawn from their initial study was that

> "N-body simulations of CDM Universes are now of such size and quality that realistic modelling of galaxy formation in volumes matches to modern surveys has become possible."

This ignores the computational effort required for such a simulation, however. We would hope that a smaller agent-based approach may yield similarly significant results, but only requiring a fraction of the CPU time.

The Bolshoi Simulation (2011) [KTGP11] is in many respects a larger, more up-to-date version of the Millennium simulation. The WMAP Seven-year data release in 2010 [J$^+$10] further refined the cosmological parameters outlined in figure 2, finding significant differences from those values released previously and used as the basis for the Millennium simulation. Because the Bolshoi simulation is so recent, very few papers have yet been published on their findings. Their initial paper describing the simulation investigated the properties of dark matter halos in detail, but presented very little information about structure-formation.

GPU-based N-body simulations are few and far-between. In their most recent book summarising the latest research efforts on 3D graphics and GPU algorithms, NVIDIA's research team published *'Fast N-Body Simulation with CUDA'* [Ngu07]. Since FLAME GPU is based on CUDA (using it to push calculations onto the GPU), any findings from this are extremely relevant to the project.

The team implemented an all-pairs algorithm to calculate acceleration for 2^{14} particles entirely on the GPU. The force for a particle-particle pair $F(p_i, p_j)$ was computed in serial, while the overall force for each particle was computed in parallel, so $\Sigma F(p_i, p_j)$ runs in a separate thread for each i.

The team found that:

> "The result is an algorithm that runs more than 50 times as fast as a highly tuned serial implementation (Elsen et al. 2006) or 250 times faster than our portable C implementation."
> (Elsen et al's paper: [E$^+$07])

And state that:

> "It is difficult to imagine a real-world algorithm that is better suited to execution on the G80 architecture[34] than the all-pairs N-body algorithm."

These findings are particularly encouraging; to demonstrate such dramatic changes in performance using an all-pairs algorithm makes the idea of running an optimised N-body solution (such as the Barnes-Hut simulation) on the GPU an extremely inviting idea. Indeed, the team even stated that:

> "In future work we hope to implement the Barnes-Hut or Fourier-Mesh-Method algorithms, to evaluate the savings of more-efficient algorithms."

Finally, this study is even more relevant the project than is initially apparent. FLAME GPU includes a number of different agent-messaging protocols for computing interactions on different levels of efficiency. The most basic one of these — the brute-force communication mode where every agent communicates with every other agent — was developed based off the algorithms given in the NVIDIA team's publication[35] ([Ngu07]). As a result, the simplest agent communication method available to us in FLAME GPU has already been designed for N-body simulation!

2.8.2 Agent-Based Models

The following studies were carried out with the use of a generic ABM package; either FLAME or FLAME GPU, as indicated.

[32] `http://gavo.mpa-garching.mpg.de/Millennium/`

[33] `http://www.mpa-garching.mpg.de/millennium/#PUBLICATIONS`

[34] NVIDIA's GeForce 8 series, the only GPUs commercially available which supported the CUDA framework at the time of publishing.

[35] From personal correspondence with Paul Richmond.

EURACE (EURopean Agent-based Computational Economics) [DvdHD08] used FLAME to build a large-scale model of the European economy. Their model included up to 10^7 agents representing *households*, up to 10^5 agents representing *firms*, and up to 100 *banks*. These agents were able to learn and adapt, while those representing governments and the central bank did not; they simply followed predefined rules. A final set of agents collated and redistributed information regarding the agents (financial ratings, for example). EURACE modelled the financial interactions between each of these, and was computed in a massively parallel environment.

In the conclusions of their preliminary report, the team state that they are confident that the project will reproduce currently unexplained phenomena of real economies, and will therefore help to explain how these emerge from simple interactions. Furthermore, their results from some preliminary investigations supported this statement[DvdHD08].

NARCSim [RLR09] used FLAME GPU to simulate the illegal drug market as a serious game. Agent types included drug addicts, recreational drug users, drug-dealers (optionally also users), treatment workers, non-users and police officers. The system then modelled the interactions of these agents as they went about their lives. Agents had their own variables, and their interactions with others were controlled by the interplay of these. Example variables for drug addicts would include: 'timeSincelastFix', 'addictionLevel', 'totalMoney', etc...

The model proved accurate, in that real-world trends (for example, as the population density increases, so does the ability for both drug-users and non-drug users to acquire drugs) were replicated within the simulation.

Some of the system's emergent behaviour proved particularly interesting; if a dealer doesn't have some certain type of drug requested by a user, then they may present an alternative (provided the user can afford it). If a user then takes the alternative drug some sufficient number of times, that user may become addicted to the new drug, thus mirroring the real-world concept of 'gateway drugs'.

By using FLAME GPU rather than FLAME, the team was able to run NARCSim on a single machine rather than a grid, and could provide real-time video feedback relating to the effects of any changes to the market.

2.8.3 Summary of Previous Studies

The Millennium Run and Bolshoi Simulation were both very large-scale studies, designed to really stretch the boundaries of our models and test their validity. Consequently, their publications focus almost entirely on the *results* of the simulation, rather than its implementation. This is an important distinction to make; for these studies, the simulation is simply the *tool* used to generate the results. On the other hand, NVIDIA's investigation into N-body simulation using CUDA is more of a 'tech-demo' — the working simulation and its algorithms were the *results*, rather than just the tool. Here, replicating the physics in minute-detail wasn't as relevant; this was a computer science study, not a physics one. The Millennium Run and Bolshoi simulation are undoubtedly the largest cosmological N-body simulations ever created, and demonstrate the extent to which this type of simulation is possible when we have access to supercomputers.

2.9 Literature Review Summary

This is quite a novel area of research; very little work has been previously undertaken using agent-based frameworks within the realm of physics or cosmology. The level of accuracy of our model is essentially just a function of how much time is available — we would be wise to start with a simple model of gravitation, and then evolve to a more complex simulation over time, ensuring that additional phenomena and mathematical constructs integrate well with existing work as we go.

We have learnt that galaxies are formed as a result of repeated collisions and mergers with smaller bodies, and can be classified into one of several groups based on their physical characteristics. This knowledge will be relevant to evaluation of our system (section 3.4)

There are various algorithms for computing particle interactions in the context of N-body simulation, with complexity ranging from $O(N^2)$ to $O(N \log N)$. Pairwise comparison is the simplest, but also the least efficient.

Topics which may prove to be particularly problematic include incorporating relativistic effects; an expanding universe; the mechanics of star formation; the methods by which density waves propagate through the interstellar gas to form structure; angular momentum, spin and the formation of spiral arms, and the flat, disk-like nature of spiral and lenticular galaxies despite their elliptical dark matter halos.

There are various software packages available to us, and care must be taken to select the right one. FLAME is a helpful framework for ABMs; XML makes agent definitions simple, while C allows functions to be arbitrarily complex. By compiling numbered output files to a video *after* the simulation has concluded, the time taken to compute a single timestep has no effect on the video's performance (Frames Per Second). FLAME GPU is an extension to FLAME which allows it to run on the massively-parallel GPU; making complex simulations accessible to much lower-level hardware. Entire FLAME GPU models can be compiled down to a single file, then executed and visualised in real-time. This allows for easy redistribution of the model.

SWARM and GADGET are other software options. GADGET is an advanced N-body simulation tool, designed for use on grids for physics investigations. The algorithms have already been well optimised for N-body calculation, and incorporate more advanced physics than we could realistically achieve otherwise, including models with a non-zero rate of expansion of the universe, and comoving coordinates. SWARM is similar to the FLAME framework, but does not allow for parallelisation, and less documentation is available.

3 Requirements & Analysis

3.1 Project aims

This project aims to simulate the process of galaxy formation using an agent-based model. This will involve the creation of a cosmological N-body simulation, where a large number of massive 'particles' are placed in 3D space and their interactions modelled as per physical laws.

In order to initialise the simulation, we need to define the system's starting configuration. Doing so requires specifying the number of agents of each type, as well as the starting values of any variables these agents may carry. Obviously, manually specifying the positions and velocities for large numbers of agents isn't an option. We therefore aim to develop a small program which, when given basic input parameters, generates a file containing a relevant distribution of agents ready to load into our framework. From the literature review we learnt that the conditions of the early universe are well modelled with a Gaussian random field, and so this will form the basis for one of the agent distribution algorithms.

In order to observe the evolution of our simulation, and thus determine the validity of our model, we need some method of taking output files and representing them in a more intuitive format. Another aim of the project is therefore to provide a visualisation of the system as an evolving 3D graphic. All potential software candidates outlined in section 2.7 have tools to do this automatically, with varying degrees of customisability. FLAME GPU, in particular, provides a very basic but extensible visualisation shell, which we would expect to have to extend for the purposes of this project.

3.2 Requirements

We here turn our project aims into a set of specific requirements.

Requirements which are *'Necessary'* are deemed critical to the successful implementation of the system. Without these in place, the project is unlikely to yield any meaningful result. *'Desirable'* requirements are those which are likely to significantly increase the accuracy/performance of the project, but are not essential. Finally, *'Optional'* requirements are to be attempted if we are left with additional time after all necessary and desirable requirements are completed.

1. Necessary
   (a) Our model operates in a universe of fixed, three-dimensional Euclidean space
   (b) Our model contains some number of 'particle' agents each with their own variables, at the very least including:
      - Position

- Velocity
- Mass

(c) As well as agents representing baryonic matter, we should also be able to instantiate agents as 'dark' matter, the differences being:

- Dark matter only interacts with other particles by gravitation
- Visualisation of dark matter particles can be toggled on/off
- The quantity of dark matter particles is determined by the relationship between the total matter density Ω_m and the baryon density Ω_b, such that $\Omega_{cdm} = (\Omega_m - \Omega_b)$ (section 2.3)

(d) We must create some program which automatically generates a file ready for use by our choice of ABM package. At the least, must take input of:

- N; The number of 'particle' agents in the system.
- $N_{dark} \leq N$; The number of dark matter particles.

(e) There must exist some method for visualising simulation output with 3D graphics. At the least, this must take a set of pre-generated output files and render the resultant video when each file is drawn as a frame.

(f) The visualisation runs smoothly. (It has an acceptable FPS[36])

(g) The time elapsed from simulation start should be displayed on the visualisation. This is found from (timestep number $\times$ time per timestep), and should be given in Myr[37]. (From time scales of galaxy formation in 2.2.)

(h) Software/Hardware

- FLAME is platform-independent, but must be developed in C.
- FLAME can be executed on an extremely wide range of hardware, but simulations with this number of agents are unlikely to be able to be computed on machines with single (multi-core) CPUs (such as my own, or those in the Lewin Lab). As such, access to a grid service such as CICS's 'Iceberg' cluster would be required.
- FLAMEGPU requires development in C in Visual Studio, and, therefore, Windows (is not platform independent).
- FLAMEGPU requires access to CUDA hardware, which means a single CUDA compatible NVIDIA graphics card[38] is required.
- FLAMEGPU only supports the *Double* variable type as a simulation constant in hardware with CUDA 2.0 or above. Similarly, the *Double* type can only be used as an agent variable with CUDA ≥ 1.3. My development hardware is CUDA 1.3, and so we will be restricted to global variables of the 'float' type. This shouldn't pose a problem as our global constant — the gravitational constant — $G \approx 6.67384 \times 10^{-11}$, and the smallest float is given as $\approx 1.1755 \times 10^{-38}$
- SWARM is platform-independent, but intended for use with UNIX-based systems. CPU grids are not supported.
- GADGET is distributed for use with UNIX-based systems.

2. Desirable

(a) In addition to the all-pairs algorithm, we should be able to optionally select use of the more efficient Barnes-Hut algorithm (section 2.6.2).

(b) Our visualisation tool should have the capability to run in real-time with the simulation.

(c) When generating initial model files, we should be able to choose between different particle distributions. One such distribution should be a Gaussian random field, controlled by the cosmological parameters as given in section 2.3.

[36] FPS refers to the number of frames drawn per second of video. Film is shown at 24 Frames Per Second, and before this point, video can appear choppy. Our visualisation is unlikely to have a high contrast or a requirement to be perfectly smooth (unlike video games, for example), and we can therefore lower this threshold somewhat.

[37] 1×10^6 years

[38] A list of all CUDA compatible GPUs can be found at: `http://developer.nvidia.com/cuda-gpus`

(d) As well as an agent type for low-level particles, we should implement hierarchical agents. A higher-level agent which would fit well in our system would be a gas cloud composed of many individual 'particle' agents.

(e) Rules for star formation given the Jean's Criterion (section 2.4.1) should be implemented within said gas clouds.

(f) Inclusion of supernovae and/or black holes for stars above a certain mass and age threshold.

(g) Inclusion of particle collisions and/or spin (most N-body simulations treat particles as collisionless).

3. Optional

(a) Packaging all tools and executables to a single distributable application, allowing users to generate, execute and observe models specified by their own parameters.

(b) The incorporation of Relativity into the system.

(c) The use of other, more accurate, models of the universe. Particularly, modelling with an expanding universe.

3.3 Analysis

3.3.1 Project components

The implementation of the basic gravitational N-body simulation can be considered the core of this project. A number of the 'necessary' requirements (1a, 1b, 1c) are therefore components of this, and methods of approaching each have been discussed extensively in the literature survey. To complete the most basic N-body system, we need to design a single 'particle' agent type, and create the code to compute its gravitational attraction as an agent-rule within some ABM framework. In the case of FLAME, this means specifying an agent in XML, and coding its transition functions in C (section 2.7.2). In Swarm, we must write the entire class representing our agent and use this to form a model swarm (section 2.7). The Swarm equivalent of FLAME's transition functions is the 'step' method, necessary for every agent class. In such a simple system, the only force acting upon a particle is its acceleration due to the gravitational attraction of every other particle, and so the transition/step functions need to implement this accurately.

In the basic implementation using a single agent type, the only difference between particles of dark and baryonic matter is that 'dark' particles aren't shown in the visualisation of our simulation (requirement 1c). All we need to do in order to implement this in conjunction with requirement 1e is add some boolean variable 'isDark' to the definition of a particle agent, and then include an GUI option on the visualisation to toggle dark matter visualisation on/off. When off, the visualiser will only draw agents with a false 'isDark' flag.

Our tool to create a model's initial conditions (requirement 1d) should be fairly simple. Upon execution, the application should take input controlling the number of particles, N, as well as a ratio of baryonic matter:dark matter. A sensible default for this ratio would be that given by the cosmological parameters found in section 2.3. Appropriately handling other inputs such as distribution type and amplitude of matter perturbations are to be implemented as and when we move on to the 'desirable' requirements (in particular, requirement 2c). The control structure for this program is simple. In FLAME, initial files are written in XML, with a structure as shown in appendix A.3. To automate production of these we therefore need some function which writes the appropriate XML for one agent given the agent's variables as parameters. We then loop on this function N times, with a separate set of logic determining the appropriate variables for each agent as per our requirements (wholly random distribution, Gaussian distribution, uniform distribution, fixed masses, variable masses, no dark matter, etc...). Since this tool doesn't need to run in conjunction with our modelling framework, it can be built as a standalone application. We therefore have a choice of any appropriate programming languages to write it in. Sensible options are those with good capabilities for writing to files. With this in mind, Java, C/C++ or Python are all acceptable.

The project's third main component is the visualisation software. The extent to which we have to code this ourselves is determined by which modelling package we use, as no preexisting software suits

our purposes exactly. To write the software from scratch we must create a standalone application which reads simulation output files, and parses these to find the information relevant to the simulation's status (agent positions, for example). These attributes are then drawn to the screen as a single frame for each timestep. To render the scene we need to use some 3D Graphics API, the most notable of which are Microsoft's Direct3D, and OpenGL. Requirement 1f states that our video should run smoothly. For scientific computing such as this, a frame-rate of $\geq$ 15FPS (Frames Per Second) should be acceptable. If possible however, we should aim for $\geq$ 24FPS. FLAME and FLAME GPU offer the most substantial inbuilt support for 3D visualisation, and using either of these packages would only require slightly customising preexisting tools; the open-source 'FLAMEVisualiser' for FLAME, or extending FLAME GPU's relevant inbuilt classes. One example of such a modification will be requirement 1g, overlaying model information onto the video.

3.3.2 Choice of software

We compared the selection of software available to us in detail in section 2.7. With this information, and in light of my requirements, I have chosen to use the FLAME framework — specifically FLAME GPU — as the modelling environment on which this project will be based. The reasons for this being:

- Both FLAME and FLAME GPU have been developed in-house. As such, plenty of help, information and documentation is available beyond that of the other options.
- I am already reasonably familiar with the FLAME framework, having experimented with and attended a day's workshop on it before[39].
- Visualisation tools for FLAME (and particularly FLAME GPU) are more advanced than those available for the other software choices.
- The FLAME framework is explicitly an agent modelling package, in contrast to GADGET. This allows us to have full-reign over the types of objects simulated.
- The FLAME GPU visualiser runs in real-time, making model development and debugging easier.
- By working on the GPU, the simulation should run faster (the time taken for each timestep will be reduced).
- FLAME GPU requires a NVIDIA Graphics card with CUDA capability. My own desktop PC has a powerful CUDA compatible card for PC gaming. FLAME, on the other hand, will work on far more diverse hardware, but my machine's CPU is unlikely to perform as well as its GPU for this type of operation. In the case of FLAME, we may have been required to obtain time on one of the university's grid computing services, which would limit testing and devlopment.
- When demonstrating the model (for example, at the poster session), it will be possible to run in real-time rather than running a video of execution. (provided that either Lewin machines have suitable GPUs, or, more likely, that I bring my own development box.)
- Few studies have used the GPU for N-body simulation; by doing this we're experimenting in a novel area of Computer Science.

3.3.3 Potential Problems

Because Agent-Based modelling works on the principle of accurately modelling low-level interaction and then observing the resulting emergent behaviour, we must be very careful about the extent to which we simplify fundamental interactions. Within the context of this project, the emergent behaviour we wish to observe is galaxy formation, and we may find that with only a basic model of gravity this simply won't occur. To help deal with this, I expect to have to spend a significant amount of time improving the model based on the observed results. Galaxy spin is an example of a property that might be particularly difficult to replicate with this approach; the galaxy spins as a whole, but on the agent-level, a particle's spin has no effect on the higher system.

[39]The workshop, organised by Daniella Romano, contained presentations on FLAME and FLAME GPU, as well as hands-on programming exercises with FLAME GPU, 14/10/2010

Another problem is encountered in knowing how much we can reasonably simulate given system resource constraints. As the fidelity of the simulation increases (we add more agents), so does the quantity of calculations we have to perform per timestep. We must here strike an acceptable trade-off between accuracy and performance. With many agents, our simulation becomes more realistic and we can remove lossy abstractions. An extreme case; simulating a galaxy with atoms-as-agents would provide an incredibly high-fidelity model, but would clearly be computationally infeasible[40]. By using FLAME GPU rather than FLAME, Swarm or GADGET, we mitigate this problem somewhat by moving to the more efficient GPU. However, when visualising model results in real-time, as with FLAME GPU, we have to worry about framerates. For the video to not appear overly choppy, we would wish to render no less than around fifteen Frames Per Second. We must therefore not use a model that takes $> \frac{1}{15}$s to calculate one timestep. We can approach this problem by increasing the simulation size until the computation time is not worth the benefits from a larger simulation.

3.4 Evaluation

The obvious approach to evaluate a simulation of something is to compare its accuracy against what we observe in the real world. That is, if a simulation's output replicates the actual observed phenomena, then this can be considered an accurate representation of whatever it's modelling. Since the timescales involved obviously don't allow us to observe the process of galaxy formation directly, we must instead compare our simulation against the collection of theory which explains how we *believe* it to happen (the Λ-CDM). Specifically, we would hope to observe small 'dwarf' galaxies forming from initial non-uniformities in the early matter distribution, and then galaxies forming as a result of collisions and mergers between dwarf galaxies.

We can also evaluate the system's accuracy with the Hubble tuning fork (figure 3). If the galaxies observed in our simulation can be classified using these groups, then this would also imply that we've done a good job of modelling the real world.

4 Conclusions and project plan

The majority of work so far has been invested into learning the physics and other background knowledge related to the project, including those similar studies which have been previously undertaken. I have experimented with FLAME, and have a good basic understanding of the framework.

Planning specifically how to approach such a task is fairly difficult. Since implementing the basic system should be relatively easy, the majority of development time will be spent on incremental improvements, the timescales for which may be vastly different.

The plan of action is outlined below:

1. The first task will be to generate a formal specification for our agents using X-machines. This is particularly important, since it will have a significant bearing on how we model in FLAME GPU, and therefore the system as a whole.
2. With the theory concluded, we next do the simple task of defining our agents' XML structure, with reference to the X-machines produced earlier.
3. We now start writing the agents' structure and transition functions into FLAME GPU. This is building the 'model proper' and is the main section of implementation. During this stage, we should test with a few agents initialised by hand, and with the inbuilt FLAME GPU visualisation classes.
4. The application to generate t_0 files should be developed in parallel with the previous step. Once the basic N-body simulation appears to work for a few agents, we can use this tool to test for many bodies.

[40] A typical star contains 1×10^{57} hydrogen atoms. With a typical galaxy containing 4×10^{11} stars we have 1×10^{68} agents to consider. Given the $O(N^2)$ complexity of pairwise N-body computation, that's 1×10^{136} calculations per timestep.

5. Finally, we can work on adapting the visualisation software to produce a more appropriate rendering.
6. With any remaining time we should address the desirable and optional requirements with an aim to refine the simulation's accuracy

   We have formed a Gantt chart of this plan, shown on the next side:

Figure 7: Project Gantt chart. Task 1 particularly long because it spans the Christmas break and exam period. Tasks 3 and 5 are to be completed in parallel, and task 4 once enough of task 3 is implemented. Based on testing results we build further. Once tasks 3 and 5 are completed, we can start testing with many agents. When we are satisfied with this, we begin extending the visualiser (task 7). Once the visualiser has been suitably developed the necessary requirements are finished, so we extend the system by implementing as many desired/optional requirements as possible, testing while we go.

References

[All09] Rob Allan. Survey of Agent Based Modelling and Simulation Tools. Technical report, Computational Science and Engineering Dpt, STFC Daresbury Laboratory, 2009.

[AM+08] Jennifer K Adelman-McCarthy et al. The Sixth Data Release of the Sloan Digital Sky Survey. *The Astrophysical Journal*, 175, 2008.

[Ant93] Robert Antonucci. Unified Models for Active Galactic Nuclei and Quasars. *Annual Review of Astronomy and Astrophysics*, 31:473–521, 1993.

[BASM06] A.G. Bedregal, A Aragn-Salamanca, and M.R. Merrifield. The Tully-Fisher relation for S0 galaxies. *Monthly Notices of the Royal Astronomical Society*, 373:1125–1140, 2006.

[BH86] Josh Barnes and Piet Hut. A hierachical O(N log N) force-calculation algorithm. *Nature*, 324, 1986.

[BKW+07] Sudhanshu Barway, Ajit Kembhavi, Yogesh Wadadekar, C.D Ravikumar, and Y.D Mayya. Lenticular Galaxy Formation: Possible Luminosity Dependence. *The Astrophysical Journal*, 661, 2007.

[BP98] Dino Boccaletti and Giuseppe Pucacco. Chaos in N-body Systems. *Planetary and Space Science*, 46, 1998.

[CASS+94] Shaun Cole, Alfonso Aragn-Salamanca, Carlos S.Frenk, Julio F.Navarro, and Stephen E.Zepf. A recipe for galaxy formation. *Monthly Notices of the Royal Astronomical Society*, 271:781–806, 1994.

[Dre80] Alan Dressler. Galaxy, morphology in rich clusters: implications for the formation and evolution of galaxies. *The Astrophysical Journal*, 236:351–356, 1980.

[DvdHD08] Christophe Deissenberg, Sander van der Hoog, and Herbert Dawid. EURACE: A Massively Parallel Agent-Based Model of the European Economy. *Applied Mathematics and Computation*, 204:541–552, 2008.

[E+07] Erich Elsen et al. N-Body Simulations on GPUs. *CoRR*, abs/0706.3060, 2007.

[Ein17] Albert Einstein. Kosmologische Betrachtungen zur allgemeinen Relativittstheorie (Cosmological Considerations in the General Theory of Relativity). In *Koniglich Preuische Akademie der Wissenschaften, Sitzungsberichte (Berlin) (Royal Prussian Academy of Sciences, Proceedings (Berlin))*, pages 145–152, 1917.

[Fey64] Richard Feynman, 1964. The Messenger Lectures; Lecture 1; 48:00; `http://tinyurl.com/3uvnurv`.

[G+05] Bettagli Giuseppina et al. The radial velocity dispersion profile of the Galactic halo: constraining the density profile of the dark halo of the Milky Way. *Monthly Notices of the Royal Astronomical Society*, 364:433–442, 2005.

[Gil04] Gerry Gilmore. The Short Spectacular Life of a Superstar. *Science*, 304, 2004.

[Gro04] Swarm Development Group. Documentation Set of Swarm 2.2, 2004.

[IN] Space Telescope Science Institute and NASA. `http://www.stsci.edu/portal/`.

[J+10] N Jarosik et al. Seven-Year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Sky Maps, Systematic Errors, and Basic Results. *Astrophysical Journal Supplement Series*, 2010.

[KTGP11] A Klypin, Sebastian Trujillo-Gomez, and Joel Primack. Dark Matter Halos in the Standard Cosmological Model: Results from the Bolshoi Simulation. *Astrophysical Journal*, Accepted for publication, 2011.

[Lay93] Gilbert Laycock. *The Theory and Practice of Specification Based Software Testing.* PhD thesis, The University of Sheffield, 1993.

[Lin95] Andrei Linde. Quantum Cosmology and the Structure of Inflationary Universe, 1995. Invited talk at the joint Johns Hopkins Workshop - PASCOS meeting: http://xxx.lanl.gov/abs/gr-qc/9508019.

[Lin09] Tancred Lindholm. N-body algorithms (seminar presentation), 2009. www.cs.hut.fi/~ctl/NBody.pdf.

[LL10] Ofer Lahav and Andrew Liddle. The Cosmological Parameters 2010, 2010. Article for The Review of Particle Physics; http://pdg.lbl.gov/.

[Mao97] Dan Maoz. Detectability of HIgh-Redshift Elliptical Galaxies in the Hubble Deep Field. *The Astrophysical Journal*, 490, 1997.

[Mih] Chris Mihos. Gravitational collapse of gas clouds. Astr 221 Lecture 36, Case Western Reserve University: http://burro.cwru.edu/Academics/Astr221/LifeCycle/jeans.html.

[MN06] Charles M Macal and Michael J North. Introduction to Agent-based Modeling and Simulation. Presentation, Center of Complex Adaptive Agent Systems Simulation, Agronne National Laboratory, 2006.

[Ngu07] Hubert Nguyen. *GPU Gems 3*. Addison-Wesley Professional, 2007.

[Ric11] Paul Richmond. *FLAME GPU Technical Report and User Guide*, 2011.

[RLR09] Daniela Romano, Lawrence Lomax, and Paul Richmond. NARCSim An Agent-Based Illegal Drug Market Simulation. In *Proc. of The International IEEE Consumer Electronics Society's Games Innovations Conference*, 2009.

[RR11] Paul Richmond and Daniella Romano. Template driven agent based modelling and simulation with CUDA. In *GPU Computing Gems Emerald Edition*, chapter 21. Morgan Kaufmann, 2011.

[S+02] R Schodel et al. A star in a 15.2-year orbit around the supermassive black hole at the centre of the Milky Way. *Nature*, 419:694–696, 2002.

[S+05] Volker Springel et al. Simulating the joint evolution of quasars, galaxies and their large-scale distribution. *Nature*, 435:629–636, 2005.

[Spr05] Volker Springel. User guide for GADGET-2, 2005. http://www.mpa-garching.mpg.de/gadget/users-guide.pdf.

[St07] NASA/WMAP Science team, 2007. Public Domain NASA image releases from the WMAP experiment: http://lambda.gsfc.nasa.gov/product/map/current/m_images.cfm.

[Tys01] Neil deGrasse Tyson, editor. *Cosmic Horizons: Astronomy at the Cutting Edge*. New Press, The, 2001.

A Appendices

A.1 Pinwheel galaxy

Figure 8: Image of the Pinwheel galaxy showing many, tightly wound spiral arms. Taken with the Hubble space telescope. Credit NASA, ESA. From: `http://hubblesite.org/newscenter/archive/releases/2006/10/`

A.2 Barnes-Hut subdivision

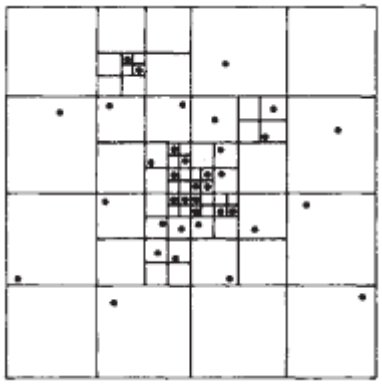

Figure 9: More detailed example of Barnes-Hut subdivision of particle space, taken from their original 1986 paper [BH86]

A.3 FLAME XML structure

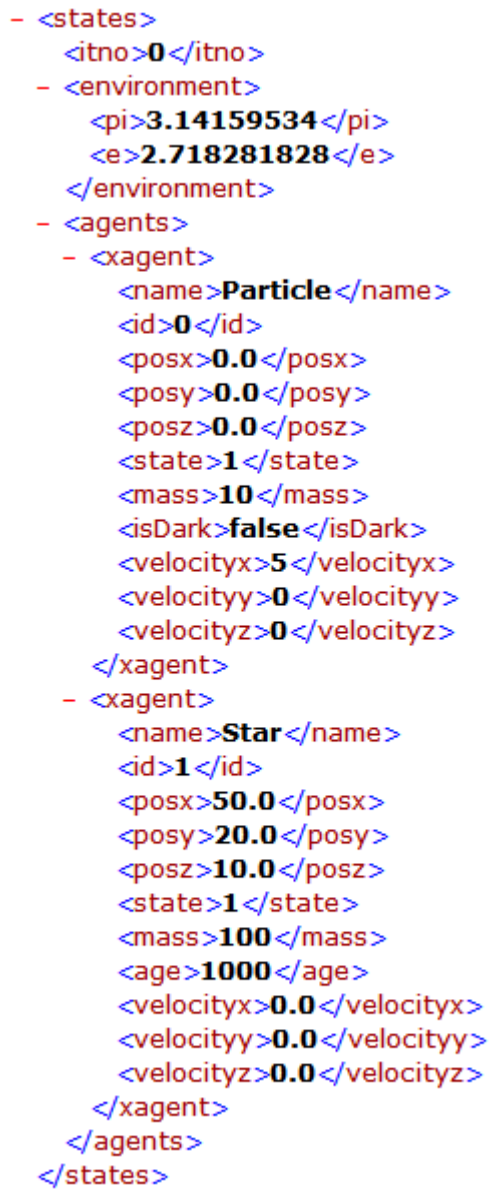

Figure 10: Example potential t0 file, defining the mathematical constants π and e, as well as one 'Particle' agent and its associated variables. Also contains one 'star' agent with its (different!) variables. Variables are just ideas of what agents may potentially require, and this is not indicative of the final agents' XML. Provided just to demonstrate the format. Author's own work.