Gadget4 Crash Course

Bradley Kavanagh

https://github.com/bradkav/CrashCourse_Gadget4/

Installation and execution

Detailed instructions for compiling and running Gadget4 can be found here. Some simpler ‘tips’ are here.

In order to compile, you’ll need a **Config.sh** file, which you can obtain from the example folders, or by editing the **Template-Config.sh** file provided in the Gadget4 repo.

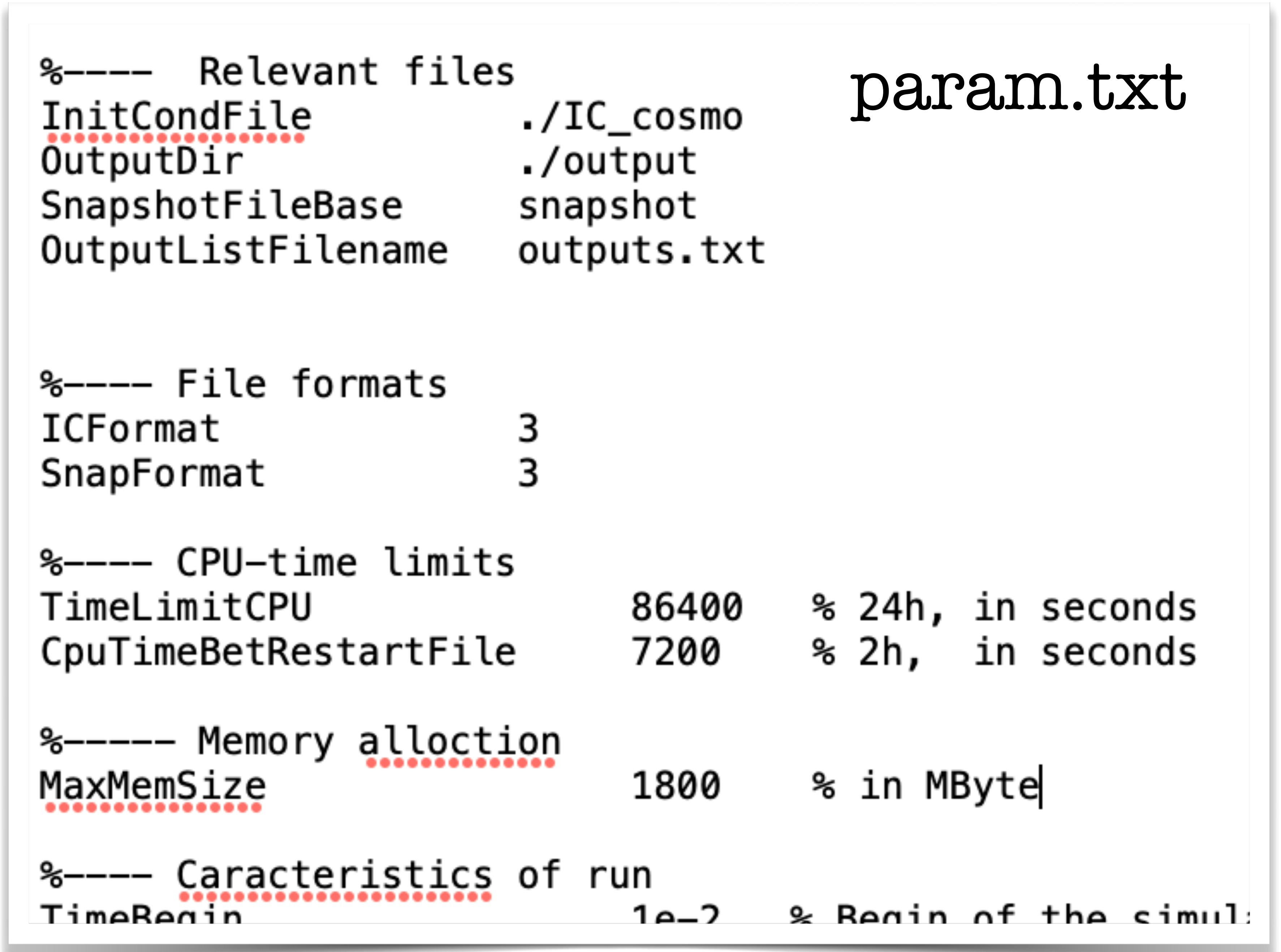
The **Config.sh** file specifies certain options in the code which are ‘baked in’ to the compilation. Such as whether the simulation region should be periodic, etc.

Once compiled, you can run Gadget4 (on **N** cores) using:

```
mpirun -np N ./Gadget4 param.txt
```

The file **param.txt** includes all of the runtime options (initial conditions file, units, integration and force calculation accuracy, etc) which specify how the code should run.

It is *highly* recommended to always keep the **Config.sh** and **param.txt** files alongside any output, so that you can always tell what the simulation was doing (and potentially reproduce it).

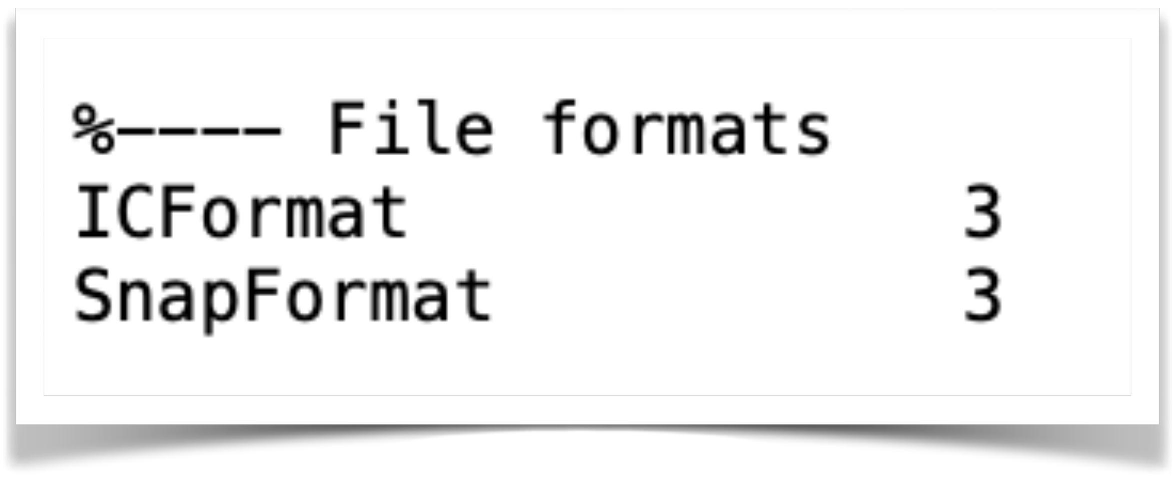


param.txt

```
%----  Relevant files
InitCondFile          ./IC_cosmo
OutputDir             ./output
SnapshotFileBase      snapshot
OutputListFilename    outputs.txt


%---- File formats
ICFormat              3
SnapFormat            3

%---- CPU-time limits
TimeLimitCPU              86400    % 24h, in seconds
CpuTimeBetRestartFile     7200     % 2h,  in seconds

%----- Memory alloction
MaxMemSize                1800     % in MByte

%---- Caracteristics of run
TimeBegin                 1e-2    % Begin of the simul:
```

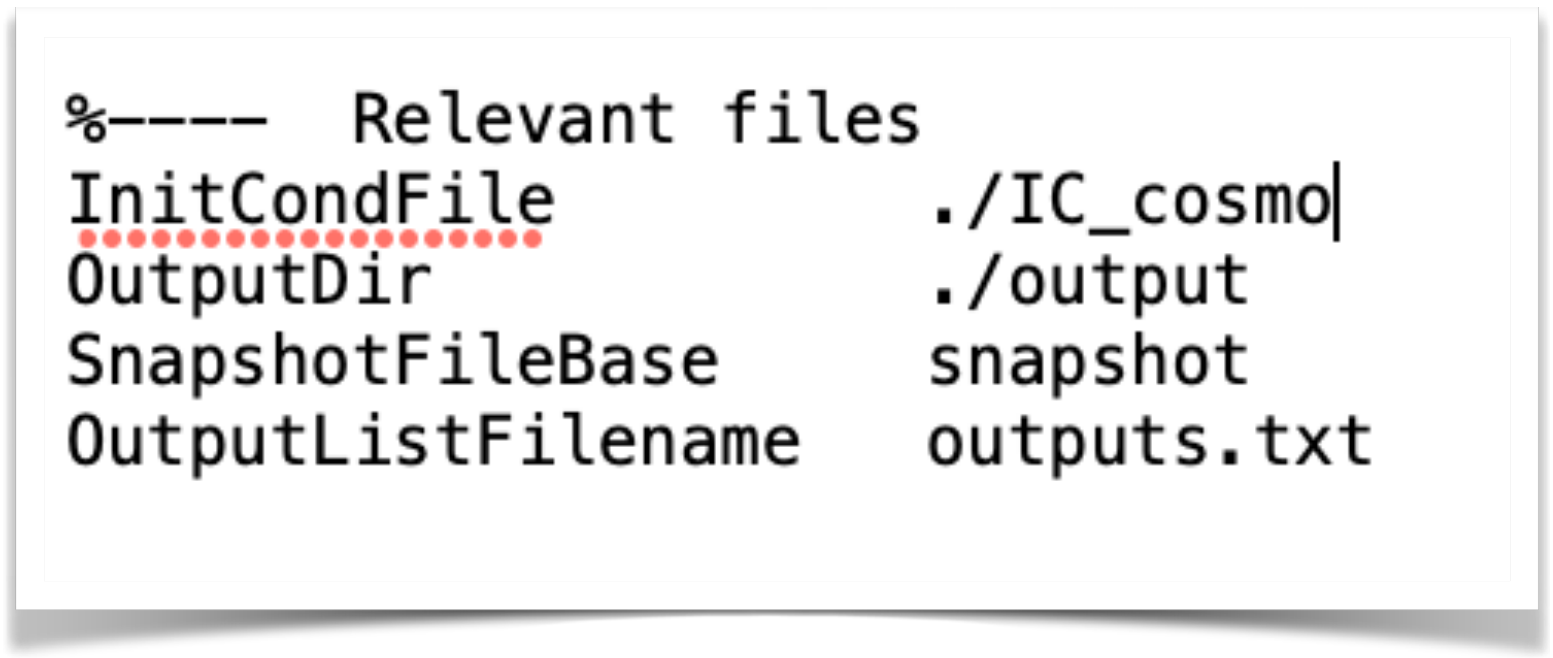
Initial Conditions

The best idea is to use hdf5 format for setting the initial conditions (option 3 for ICFormat and SnapFormat in **param.txt**)



```
%---- File formats
ICFormat                 3
SnapFormat               3
```

You can specify the file containing the initial conditions using the **InitCondFile** parameter (just drop the .hdf5 file extension)



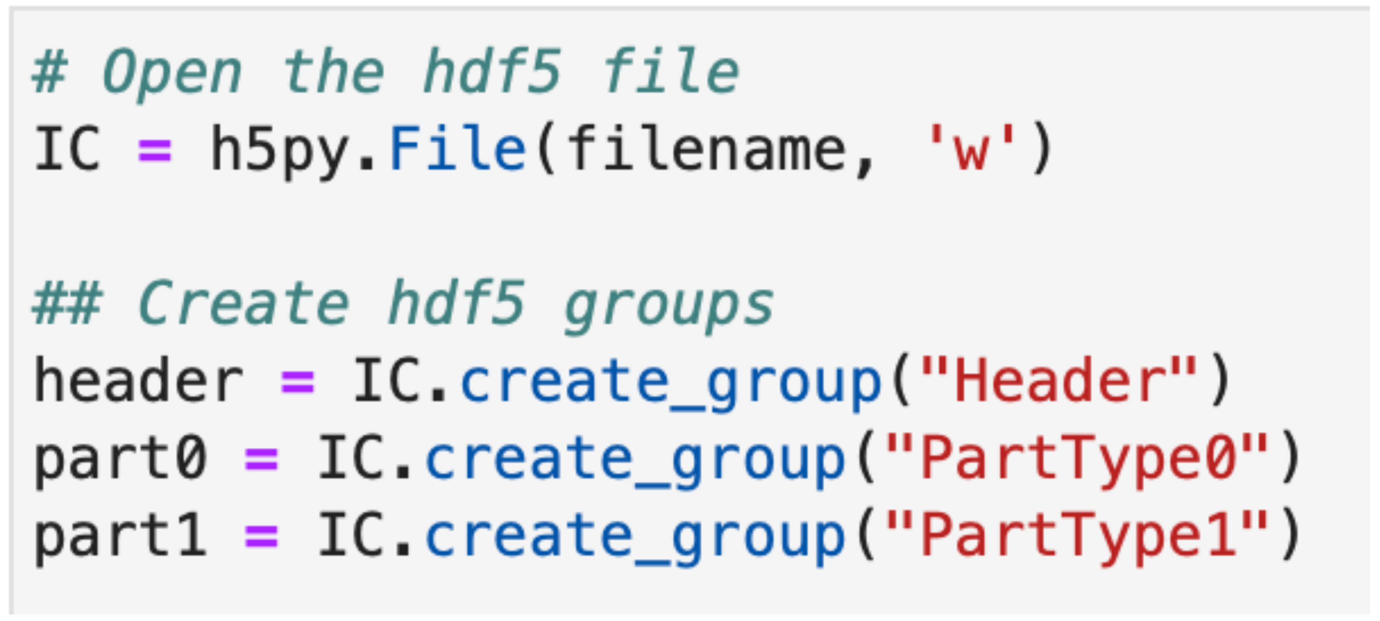
```
%----  Relevant files
InitCondFile          ./IC_cosmo
OutputDir             ./output
SnapshotFileBase      snapshot
OutputListFilename    outputs.txt
```

ICs and Particle Types

Gadget4 allows for a number of particle types, allowing us to keep track of distinct classes of particles.

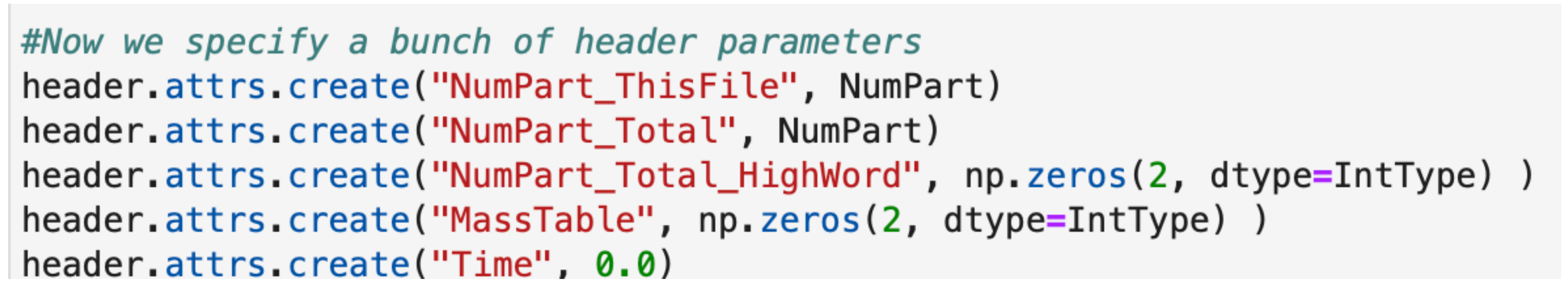
Type0 particles are *always* Smoothed Particle Hydrodynamics (SPH) — i.e. gas — particles.

We won’t want any of those, so we’ll only use Type1 (and other) particles.



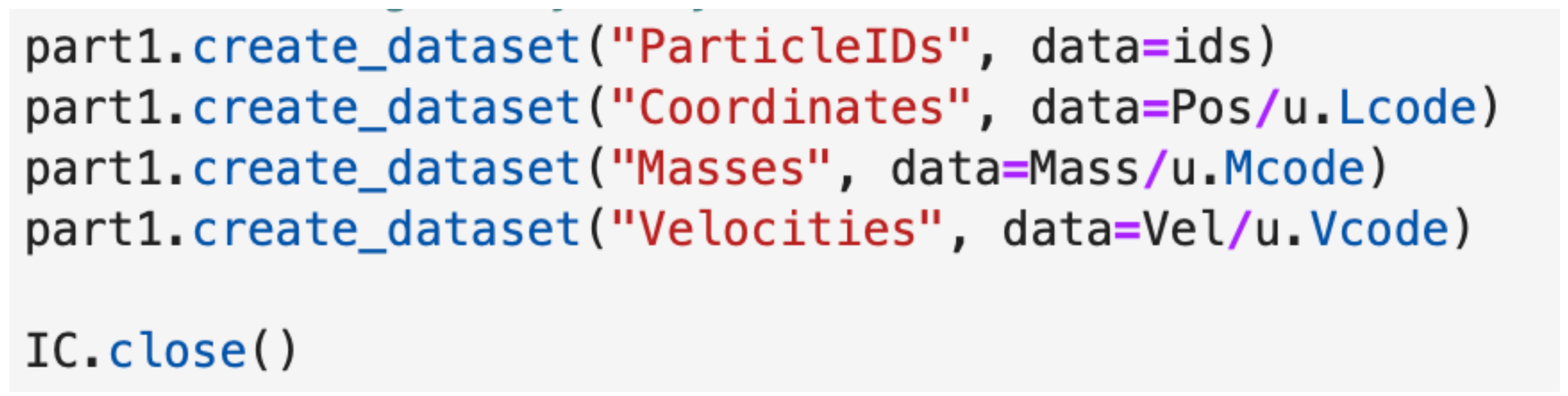
```
# Open the hdf5 file
IC = h5py.File(filename, 'w')

## Create hdf5 groups
header = IC.create_group("Header")
part0 = IC.create_group("PartType0")
part1 = IC.create_group("PartType1")
```

Specify a ‘header’ group and then a group for each particle type



```
#Now we specify a bunch of header parameters
header.attrs.create("NumPart_ThisFile", NumPart)
header.attrs.create("NumPart_Total", NumPart)
header.attrs.create("NumPart_Total_HighWord", np.zeros(2, dtype=IntType) )
header.attrs.create("MassTable", np.zeros(2, dtype=IntType) )
header.attrs.create("Time", 0.0)
```

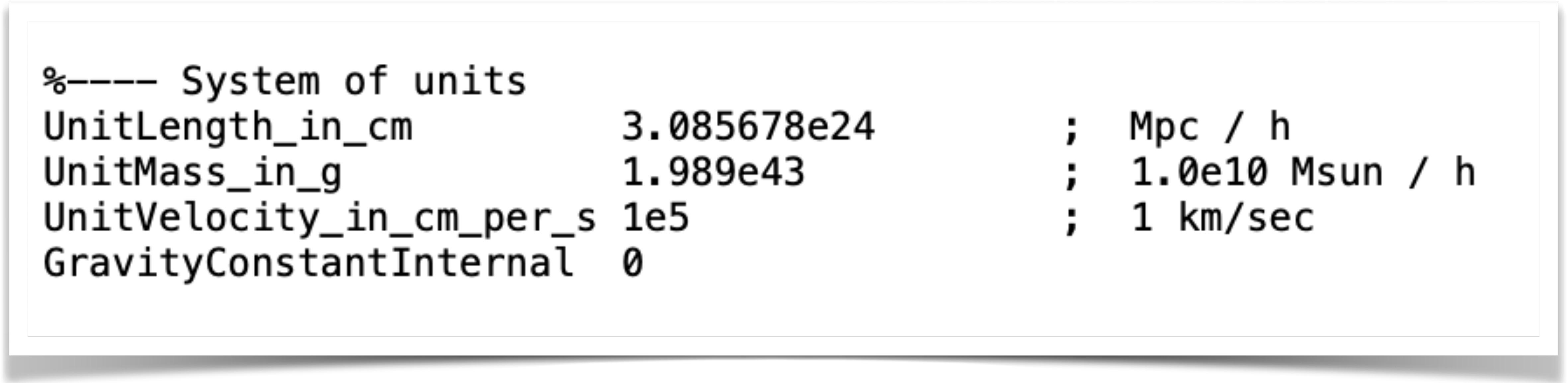
Specify some essential header information (numbers of particle etc)



```
part1.create_dataset("ParticleIDs", data=ids)
part1.create_dataset("Coordinates", data=Pos/u.Lcode)
part1.create_dataset("Masses", data=Mass/u.Mcode)
part1.create_dataset("Velocities", data=Vel/u.Vcode)

IC.close()
```

Copy over the positions, masses velocities for particles of each type.

Units



```
%---- System of units
UnitLength_in_cm         3.085678e24        ;  Mpc / h
UnitMass_in_g            1.989e43           ;  1.0e10 Msun / h
UnitVelocity_in_cm_per_s 1e5                ;  1 km/sec
GravityConstantInternal  0
```

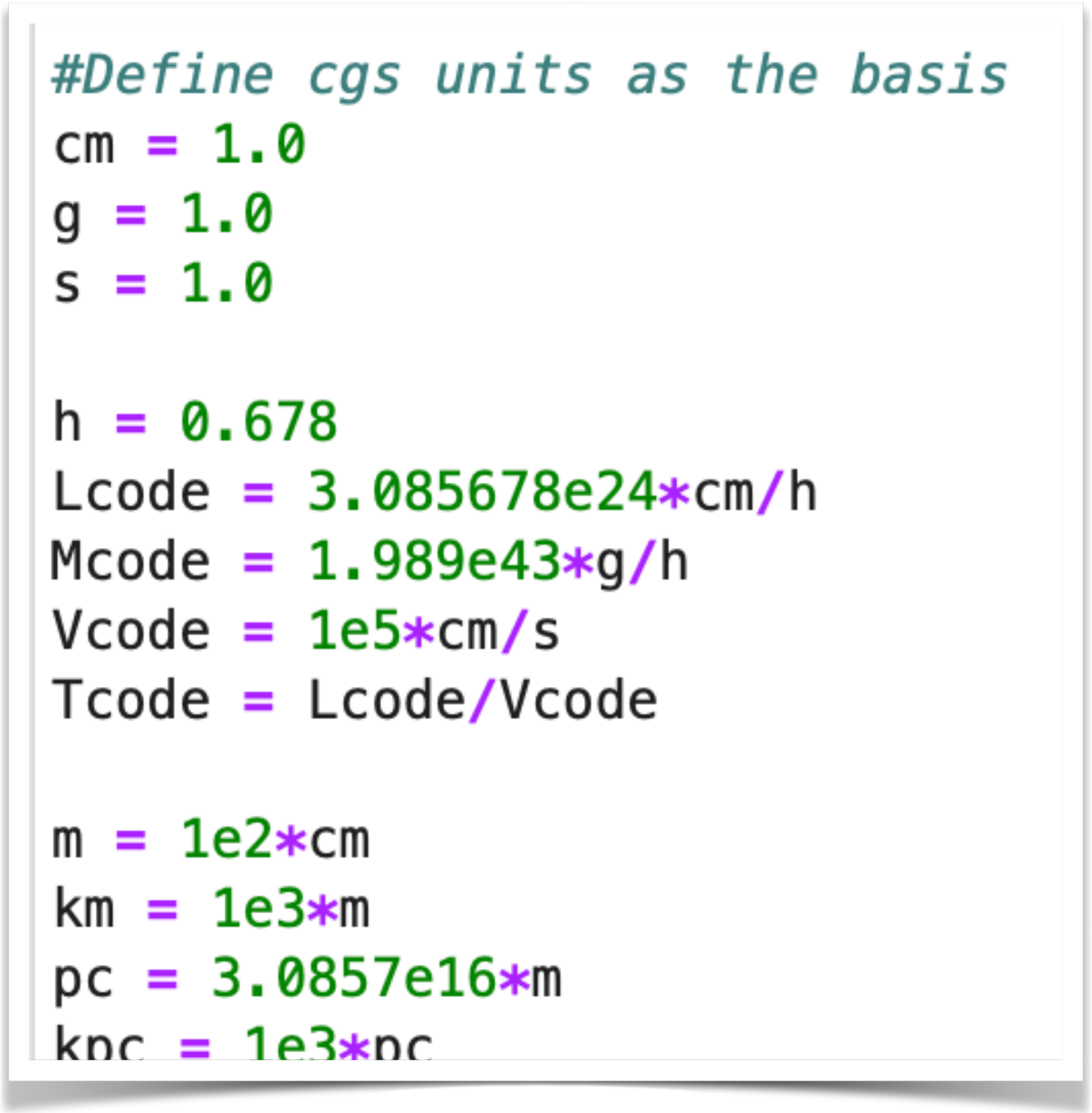
Gadget4 uses a system of internal code units, which you can change in the **param.txt** file.

Once you fix the internal code units for Length, Mass and Velocity, this fixes the internal value of G and also fixes the internal code units for Time (=Length/Velocity).

These units can get confusing, so I usually have a **units.py** module, where everything is defined in terms of cgs units (cm, g, s). Then it's easy to convert between everything.

If you use this approach, *don't forget to always specify the units!*

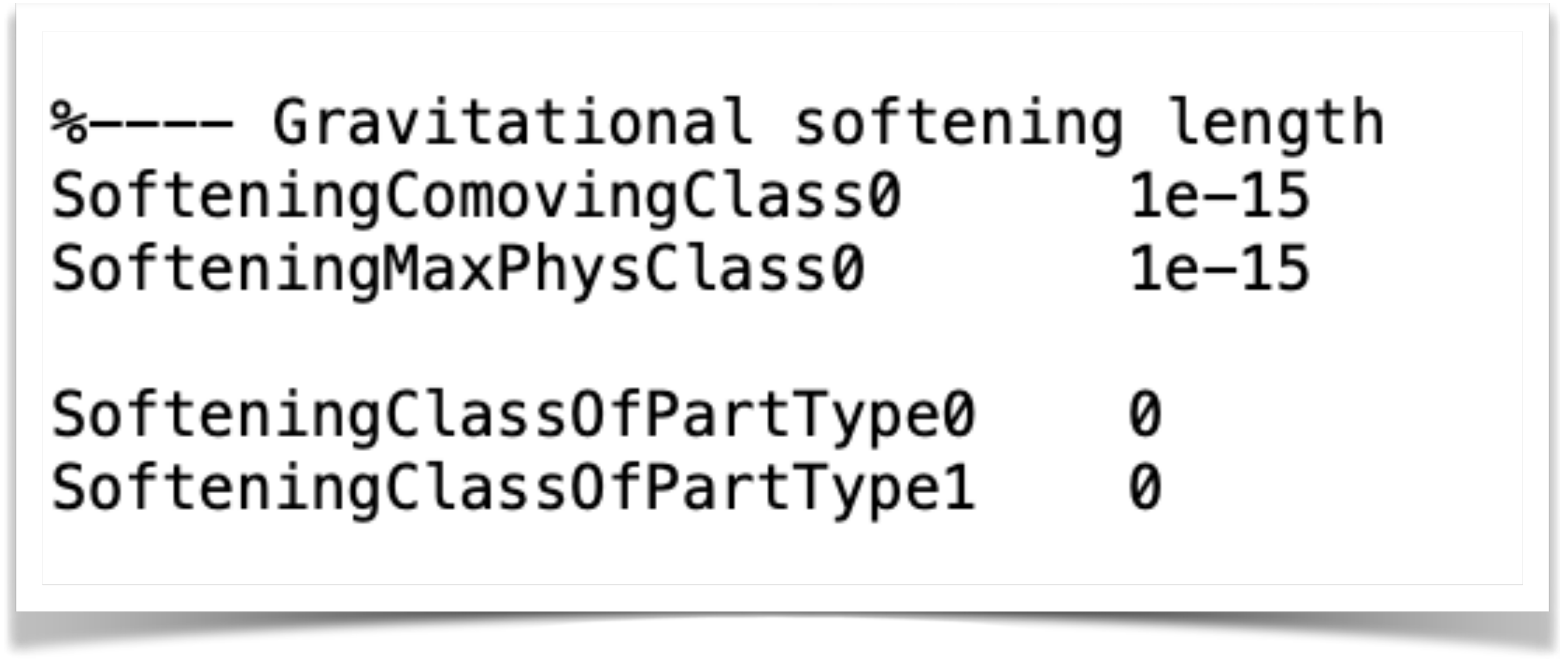
Then remember to divide by code units before you pass the initial conditions, and multiply by code units after you read in the snapshots.



```python
#Define cgs units as the basis
cm = 1.0
g = 1.0
s = 1.0

h = 0.678
Lcode = 3.085678e24*cm/h
Mcode = 1.989e43*g/h
Vcode = 1e5*cm/s
Tcode = Lcode/Vcode

m = 1e2*cm
km = 1e3*m
pc = 3.0857e16*m
kpc = 1e3*pc
```

Gravitational Softening

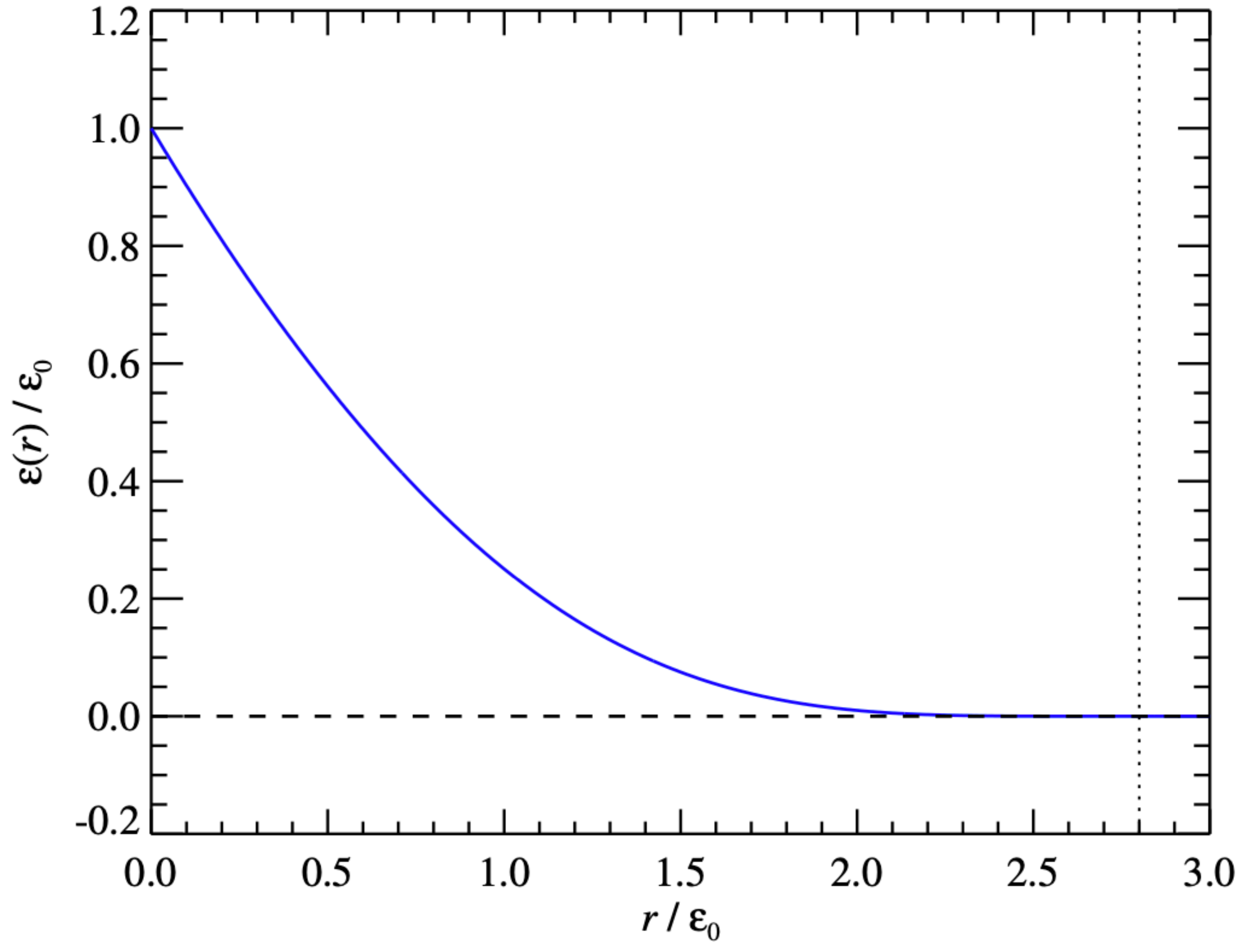
In Gadget4, the Newtonian gravitational potential is *softened* - it does not diverse as the particle separation goes to zero. This is to avoid very large accelerations, which are hard to resolve without tiny timesteps.



```
%---- Gravitational softening length
SofteningComovingClass0       1e-15
SofteningMaxPhysClass0        1e-15

SofteningClassOfPartType0     0
SofteningClassOfPartType1     0
```

In Gadget4, the potential at position $\mathbf{x}$, due to N point masses with positions $\{\mathbf{x}_j\}$, reads:

$$\phi(\mathbf{x}) = -\sum_{j=1}^{N} \frac{Gm_j}{|\mathbf{x}_j - \mathbf{x}| + \epsilon(|\mathbf{x}_j - \mathbf{x}|)}$$



[Gadget4 Paper]

The softening function $\epsilon(r)$ depends on the particle separation and the softening length ϵ_0. For $r > 2.8\,\epsilon_0$, the force is exactly Newtonian.

Note: it is possible to specify different softening lengths for different types of particle.

Gravitational Force Calculation

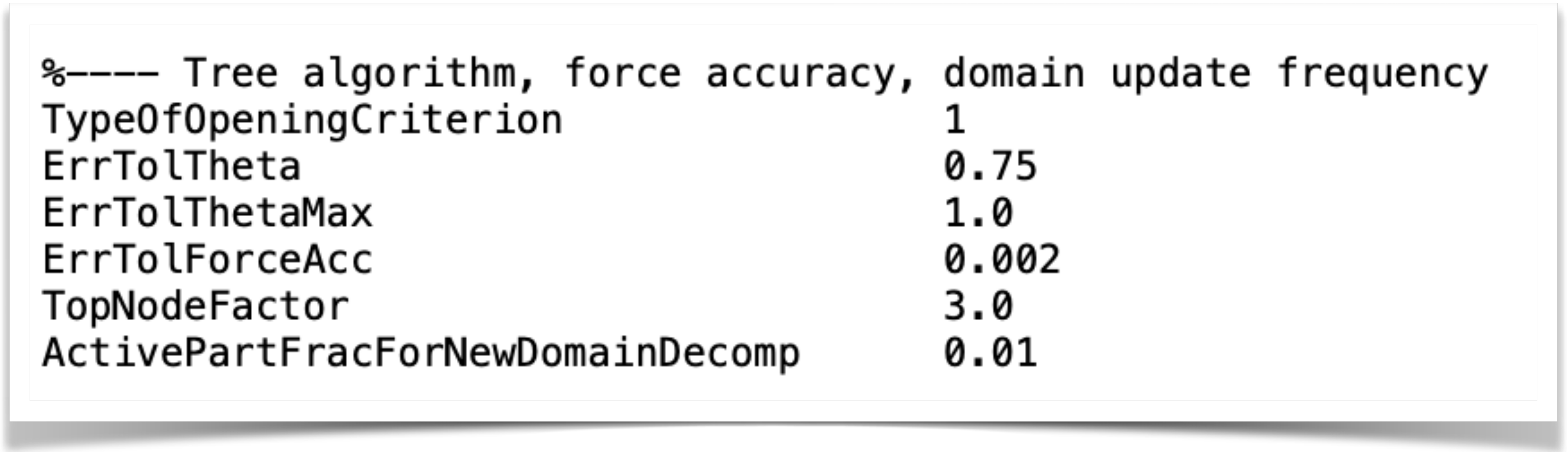
Gadget4 uses a hierarchical tree algorithm to calculate the gravitational forces.

It groups particles into cells (actually space-filling Peano-Hilbert regions).

For a cell of size ℓ at a distance D, if the opening angle $\theta = \ell / D$ is less than some critical value, we don't calculate the force from all the particles in the cell, we replace them with a single calculation - the force *due to the centre of mass of the shell.*

For TypeOfOpeningCriterion = 0, the critical opening angle is set by ErrTolTheta.

For TypeOfOpeningCriterion = 1, the critical opening angle is set by a relative criterion, designed to minimise the error.



```
%---- Tree algorithm, force accuracy, domain update frequency
TypeOfOpeningCriterion                1
ErrTolTheta                           0.75
ErrTolThetaMax                        1.0
ErrTolForceAcc                        0.002
TopNodeFactor                         3.0
ActivePartFracForNewDomainDecomp      0.01
```

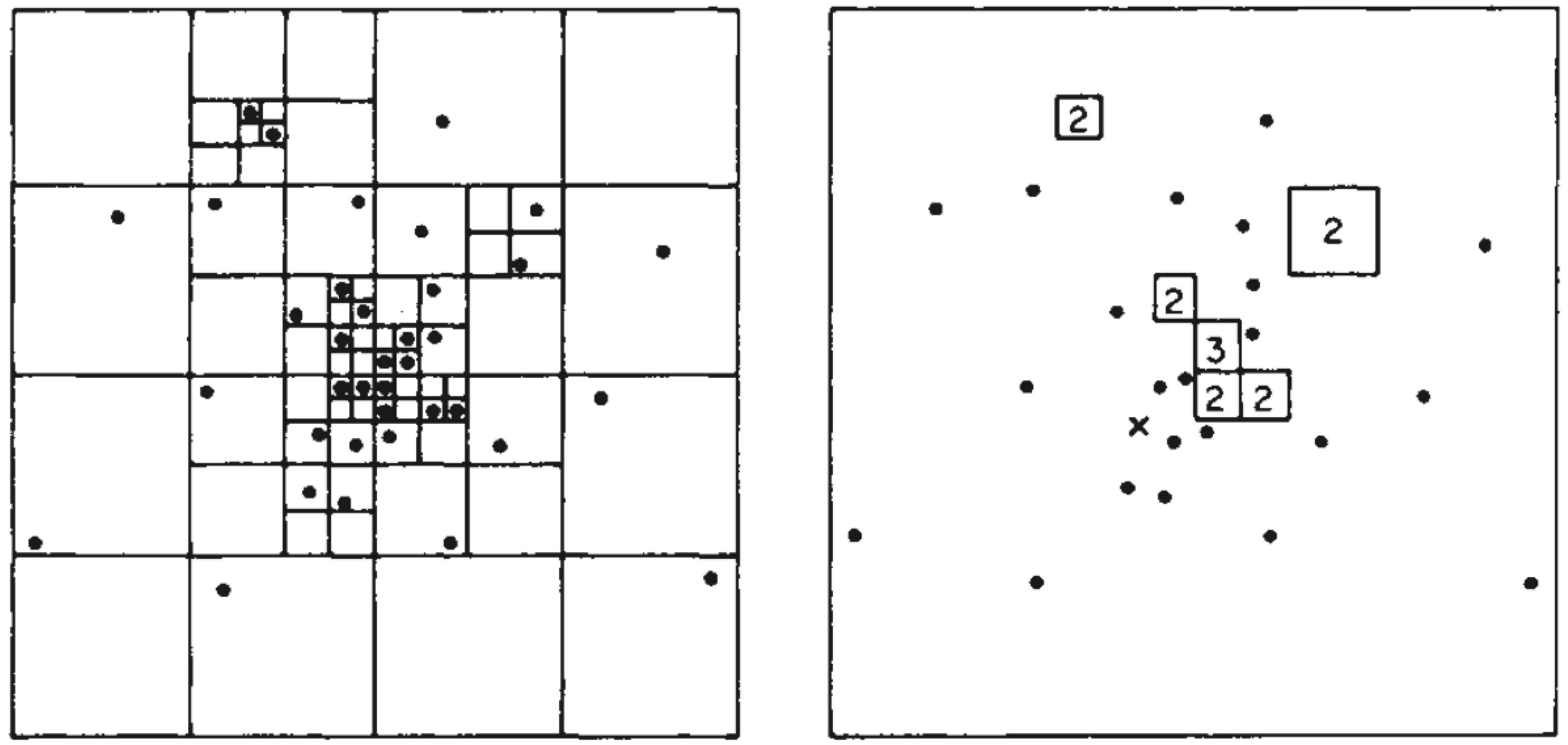
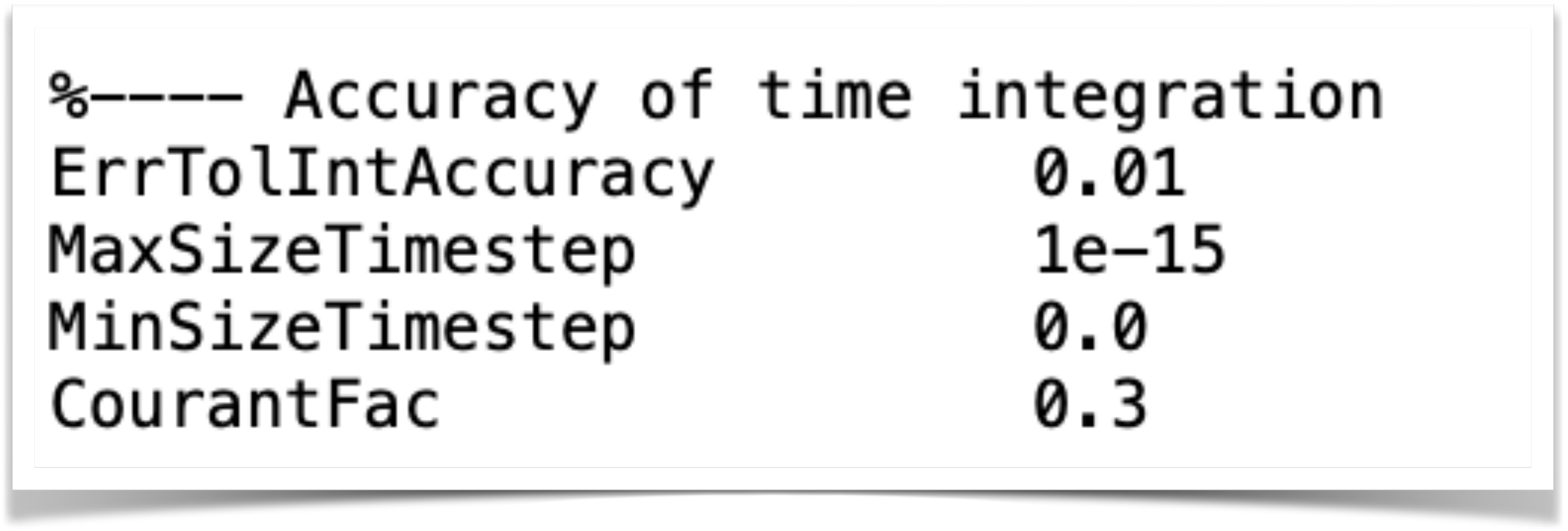


Fig. 1 Hierarchical boxing and force calculation, presented for simplicity in two dimensions. On the left, a system of particles and the recursive subdivision of system space induced by these particles. Our algorithm makes the minimum number of subdivisions necessary to isolate each particle. On the right, how the force on particle *x* is calculated. Fitted cells contain particles that have been lumped together by our 'opening angle' criterion; each such cell represents a single term in the force summation.

Time integration



```
%---- Accuracy of time integration
ErrTolIntAccuracy        0.01
MaxSizeTimestep          1e-15
MinSizeTimestep          0.0
CourantFac               0.3
```

The timestep of each particle is set according to:

$$\Delta t = \sqrt{\frac{2\eta\epsilon}{|\mathbf{a}|}}$$

where ϵ is the gravitational softening length and $\mathbf{a}$ is the gravitational acceleration.

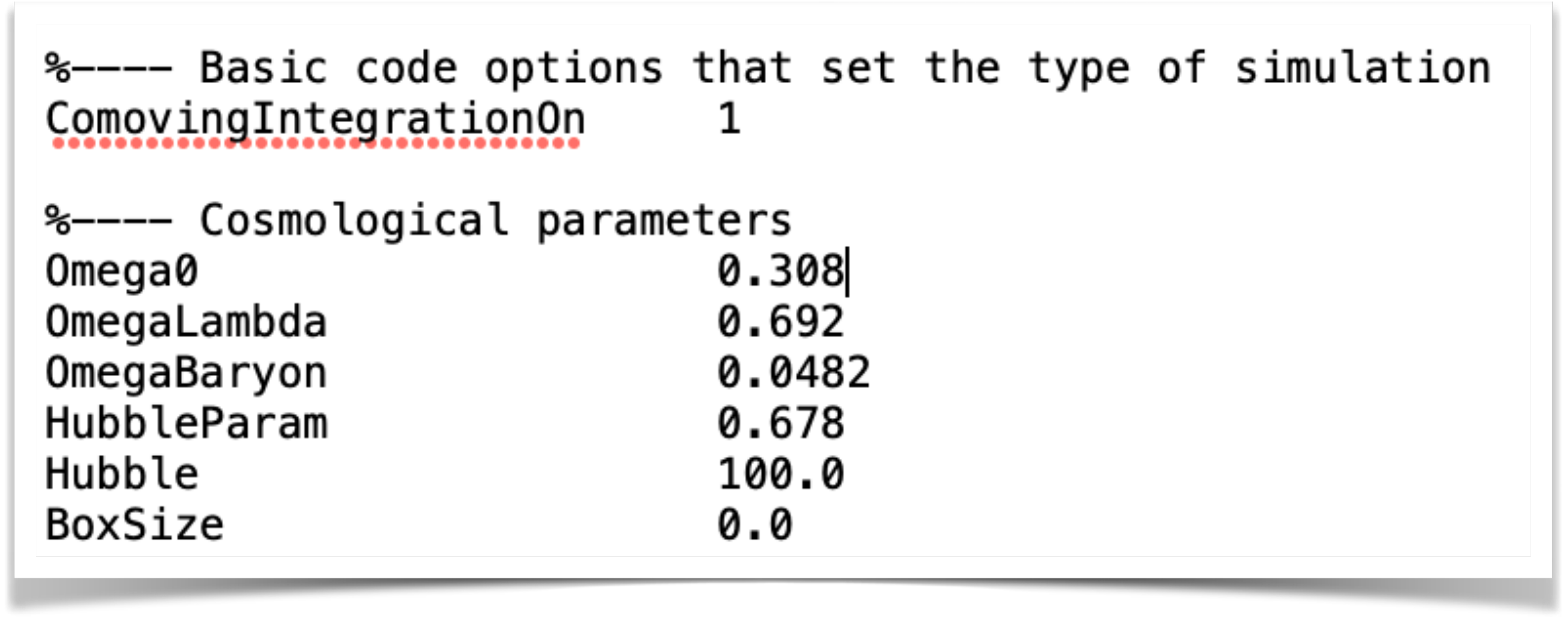
The parameter η is set by `ErrTolIntAccuracy` in the the parameter file.

Typical reasonable values are $\eta \sim 10^{-2} - 10^{-1}$

You can also set the minimum and maximum allowed timestep size.

Note that the size of the timestep depends both on the tolerance parameter η and also the softening length.

Cosmological integration



```
%---- Basic code options that set the type of simulation
ComovingIntegrationOn     1

%---- Cosmological parameters
Omega0                    0.308
OmegaLambda               0.692
OmegaBaryon               0.0482
HubbleParam               0.678
Hubble                    100.0
BoxSize                   0.0
```

Cosmological integration can be set using the `ComovingIntegrationOn` parameter.

In this case, Gadget4 treats **all coordinates are comoving** and the time parameter is no longer physical time but the scale factor a.

The parameter `MaxSizeTimestep` sets the maximum size of timestep in units of ln(a)

Output step size is *multiplicative*. *N*th snapshot is at `TimeOfFirstSnapshot`*(`TimeBetSnapshot`^N)

In this case, Gadget4 solves the dynamics for the gravitational potential $\nabla^2\phi = 4\pi G(\rho - \bar{\rho})$ and explicitly includes the Hubble expansion to the equations of motion. This means that if you simulate a region with density equal to the mean cosmic density ($\rho = \bar{\rho}$) it will not collapse and will simply expand under the Hubble flow (it will remain more or less static, in comoving coordinates).

WARNING: Gadget4 does *NOT* include radiation. That is, the Universe is matter dominated down to $t = 0$. It might be possible to include radiation, but I haven't checked the internals of the code too closely.