pNbody Documentation

Release 4

Yves Revaz

September 01, 2011

CONTENTS

Contents:

CHAPTER

ONE

OVERVIEW

pNbody is a parallelized python module toolbox designed to manipulate and display interactively very lage N-body systems.

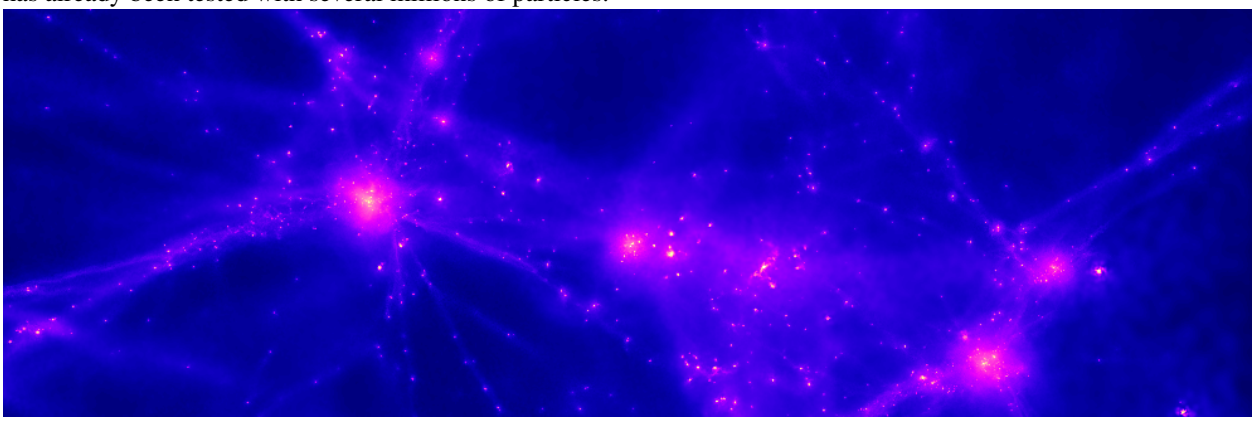
Its oriented object approche allows the user to perform complicate manipulation with only very few commands.

As python is an interpreted language, the user can load an N-body system and explore it interactively using the python interpreter. pNbody may also be used in python scripts.

The module also contains graphical facilities desinged to create maps of physical values of the system, like density maps, temperture maps, velocites maps, etc. Stereo capabilities are also implemented.

pNbody is not limited by file format. Each user may redefine in a parameter file how to read its prefered format.

Its new parallel (mpi) facilities make it works on computer cluster without being limitted by memory consumption. It has already been tested with several millions of particles.



CHAPTER

TWO

INSTALLATION

pNbody is curently only supported by linux.

2.1 Prerequiste

The basic module of pNbody needs python and additional packages :

1. Python 2.5.x, 2.6.x, 2.7.x

   http://www.python.org

2. a C compiler

   gcc is fine http://gcc.gnu.org/

3. numpy-1.0.4 or higher

   http://numpy.scipy.org/

4. Imaging 1.1.5 or higher

   http://www.pythonware.com/products/pil/

For additional but usefull special functions :

5. scipy 0.7 or higher

   http://www.scipy.org/

For the parallel capabilities, an mpi distribution is needed (ex. openmpi) as well as the additional python mpi wrapping:

6. mpi4py http://cheeseshop.python.org/pypi/mpi4py

In order to convert movies in standard format (gif or mpeg), the two following applications are needed :

1. convert (imagemagick)

   http://www.imagemagick.org/script/index.php

2. mencoder (mplayer)

   http://www.mplayerhq.hu/design7/news.html

2.2 Installing from source

2.2.1 Decompress the tarball

Decompress the tarball file:

```
tar -xzf pNbody-4.x.tar.gz
```

enter the directory:

```
cd pNbody-4.x
```

2.2.2 Compile

The compilation is performed using the standard command:

```
python setup.py build
```

If one wants to install in another directory than the default python one, it is possible to use the standard `--prefix` option:

```
python setup.py build  --prefix other_directory
```

2.2.3 Install

Now, depending on your python installation you need to be root. The module is installed with the following command:

```
python setup.py install
```

2.3 Check the installation

You can check the installation by simply running the following command:

```
pNbody_checkall
```

This command must of course be in your path. This will be the case if you did not specified any `--prefix`. On the contrary if `--prefix` is set to for example, *localdir* you should have your *PATH* environment variable should contains:

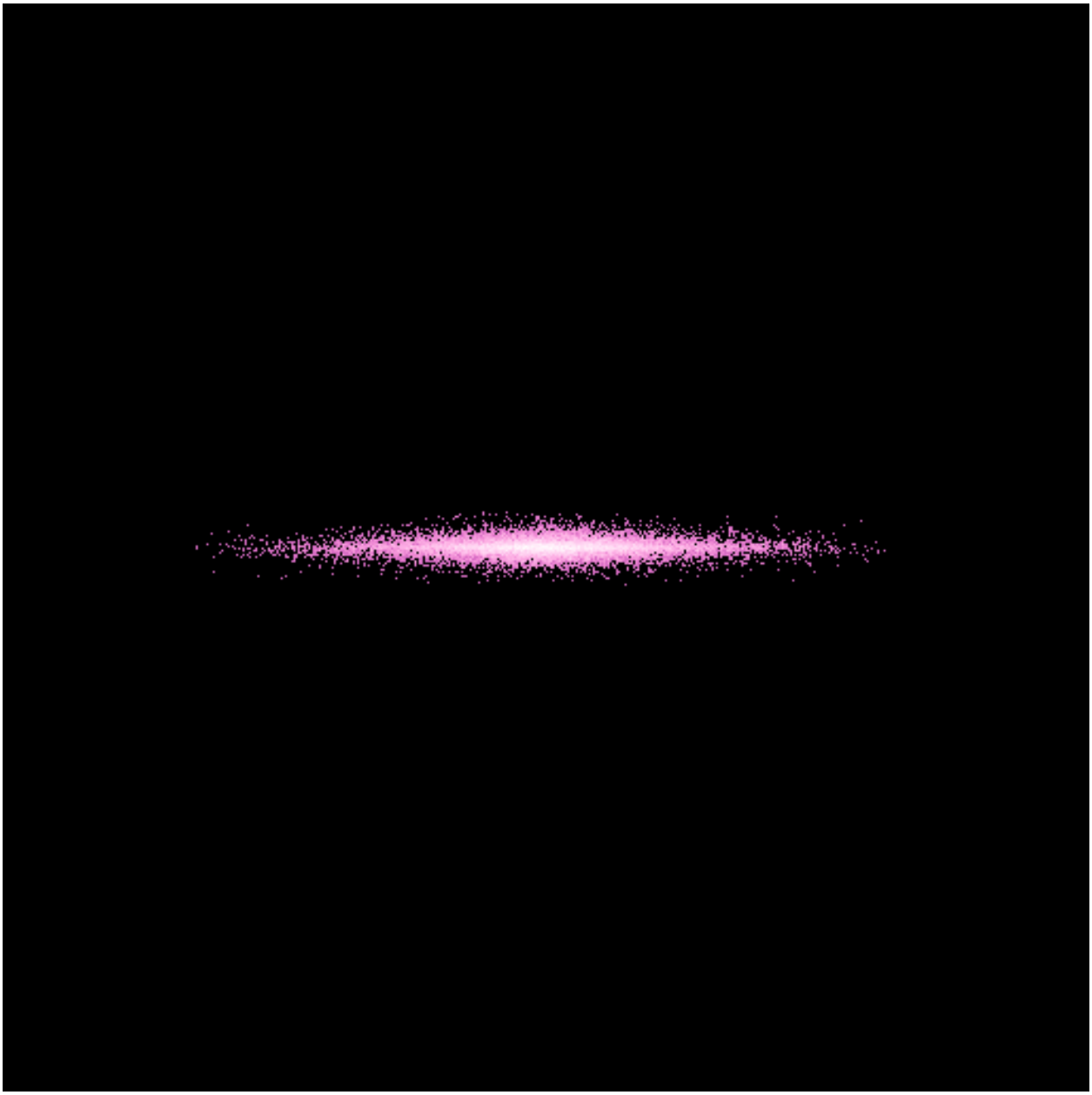
```
localdir/bin
```

and you *PYTHONPATH* environment should contains:

```
localdir/lib/python2.x/site-packages/
```

to ensure that the **pNbody** package will be found.

If everything goes well, you should see a lots of outputs on your screen, as well as a window displaying an edge-on disk.



Close it when you see it. The script should finally ends up with something like

```
#########################################################################
Good News ! pNbody with format gadget is working !
#########################################################################

You are currently using the following paths

HOME                : /home/leo
PNBODYPATH          : /home/leo/local/lib/python2.6/site-packages/pNbody
CONFIGDIR           : /home/leo/local/lib/python2.6/site-packages/pNbody/config
PARAMETERFILE       : /home/leo/local/lib/python2.6/site-packages/pNbody/config/defaultparameters
UNITSPARAMETERFILE : /home/leo/local/lib/python2.6/site-packages/pNbody/config/unitsparameters
PALETTEDIR          : /home/leo/local/lib/python2.6/site-packages/pNbody/config/rgb_tables
PLUGINSDIR          : /home/leo/local/lib/python2.6/site-packages/pNbody/config/plugins
```

```
OPTDIR              : /home/leo/local/lib/python2.6/site-packages/pNbody/config/opt
FORMATSDIR          : /home/leo/local/lib/python2.6/site-packages/pNbody/config/formats
```

2.4 Default configuration

pNbody uses a set of parameters files, color tables and formats files. These files are provided by the installation and are by default stored in the directory `site-packages/pNbody/config`. To display where these files are taken from, you can use the command:

```
pNbody_show-path
```

It is recommanded that the user uses its own configuration files. To be automatically recongnized by **pNbody**, the latter must be in the user `~/.pNbody` directory. **pNbody** provides a simple command to copy all parameters in this directory. Simply type:

```
pNbody_copy-defaultconfig
```

and check the values of the new paths:

```
pNbody_show-path
```

You can now freely modify the files contains in the configuratio directory.

By default, the content of the configuration directory is:

name	type	Content
defaultparameters	file	the default graphical parameters used by **pNbody**
unitsparameters	file	the default units parameters used by **pNbody**
formats	directory	specific class definition files used to read different file formats
rgb_tables	directory	color tables
plugins	directory	optional plugins
opt	directory	optional files

2.5 Default parameters

To see what default parameters **pNbody** uses, type:

```
pNbody_show-parameters
```

The script returns the parameters taken from the files *defaultparameters* and *unitsparameters*. Their current values are displayed:

```
parameters in /home/leo/local/lib/python2.6/site-packages/pNbody/config/defaultparameters

------------------------------------------------------------------------------------------------
                          name                            meaning             value (type)
------------------------------------------------------------------------------------------------
                           obs :                           observer =            None (ArrayObs)
                            xp :                 observing position =            None (List)
                            x0 :              position of observer =            None (List)
                         alpha :                 angle of the head =            None (Float)
                          view :                               view =              xz (String)
                         r_obs :            dist. to the observer =   201732.223771 (Float)
                          clip :                        clip planes = (100866.11188556443, 403464.4475422
                           cut :                   cut clip planes =              no (String)
```

```
                        eye :               name of the eye =            None (String)
                   dist_eye :         distance between eyes =         -0.0005 (Float)
                        foc :                       focal =           300.0 (Float)
                      persp :                 perspective =             off (String)
                      shape :            shape of the image =      (512, 512) (Tuple)
                       size :                 pysical size =    (6000, 6000) (Tuple)
                       frsp :                         frsp =             0.0 (Float)
                      space :                        space =             pos (String)
                       mode :                         mode =               m (String)
                  rendering :             rendering mode =             map (String)
                filter_name :            name of the filter =            None (String)
                filter_opts :                filter options =  [10, 10, 2, 2] (List)
                      scale :                        scale =             log (String)
                         cd :                           cd =             0.0 (Float)
                         mn :                           mn =             0.0 (Float)
                         mx :                           mx =             0.0 (Float)
                        l_n :              number of levels =              15 (Int)
                      l_min :                     min level =             0.0 (Float)
                      l_max :                     max level =             0.0 (Float)
                       l_kx :                          l_kx =              10 (Int)
                       l_ky :                          l_ky =              10 (Int)
                    l_color :                   level color =               0 (Int)
                    l_crush :              crush background =              no (String)
                   b_weight :               box line weight =               0 (Int)
                    b_xopts :                x axis options =            None (Tuple)
                    b_yopts :                y axis options =            None (Tuple)
                    b_color :                    line color =             255 (Int)

parameters in /home/leo/local/lib/python2.6/site-packages/pNbody/config/unitsparameters

-------------------------------------------------------------------------------------------------------
                        name                          meaning             value (type)
-------------------------------------------------------------------------------------------------------
                          xi :       hydrogen mass fraction =              0.76 (Float)
                  ionisation :              ionisation flag =                 1 (Int)
                  metalicity :            metalicity index =                 4 (Int)
                        Nsph :      number of sph neighbors =                50 (Int)
                       gamma :               adiabatic index =    1.66666666667 (Float)
                 coolingfile :                  Cooling file = ~/.Nbody/cooling.dat (String)
                 HubbleParam :                   HubbleParam =              1.0 (Float)
            UnitLength_in_cm :              UnitLength in cm =        3.085e+21 (Float)
               UnitMass_in_g :                 UnitMass in g =     4.435693e+44 (Float)
  UnitVelocity_in_cm_per_s :     UnitVelocity in cm per s =    97824708.2699 (Float)
```

2.6 Examples

A series of examples is provided by **pNbody** in the `PNBODYPATH/examples`, where NBODYPATH is obtained with the command:

```
pNbody_show-path
```

CHAPTER

THREE

TUTORIAL

3.1 Using pNbody with the python interpreter

In order to use this tutorial, you first need to copy some examples provided with **pNbody**. This can be done by typing:

```
pNbody_copy-examples
```

by default, this create a directory in your home `~/pnbody_examples`. Move to this directory:

```
cd ~/pnbody_examples
```

Then you can simply follow the instructions below. First, start the python interpreter:

```
leo@obsrevaz:~/pnbody_examples python
Python 2.4.2 (#2, Jul 13 2006, 15:26:48)
[GCC 4.0.1 (4.0.1-5mdk for Mandriva Linux release 2006.0)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Now, you can load the **pNbody** module:

```
>>> from pNbody import *
```

3.1.1 Creating pNbody objects from scratch

We can first start by creating a default **pNbody** objet and get info about it

```
>>> nb = Nbody()
>>> nb.info()
------------------------------------
particle file       : ['file.dat']
ftype               : 'Nbody_default'
mxntpe              : 6
nbody               : 0
nbody_tot           : 0
npart               : [0, 0, 0, 0, 0, 0]
npart_tot           : [0, 0, 0, 0, 0, 0]
mass_tot            : 0.0
byteorder           : 'little'
pio                 : 'no'
>>>
```

All variables linked to the object nb are accesible by typing nb. followed by the associated variables :

```
>>> nb.nbody
0
>>> nb.mass_tot
0.0
>>> nb.pio
'no'
```

Now, you can create an object by giving the positions of particles:

```
>>> pos = ones((10,3),float32)
>>> nb = Nbody(pos=pos)
>>> nb.info()
-----------------------------------
particle file       : ['file.dat']
ftype               : 'Nbody_default'
mxntpe              : 6
nbody               : 10
nbody_tot           : 10
npart               : array([10,  0,  0,  0,  0,  0])
npart_tot           : array([10,  0,  0,  0,  0,  0])
mass_tot            : 1.00000011921
byteorder           : 'little'
pio                 : 'no'

len pos             : 10
pos[0]              : array([ 1.,  1.,  1.], dtype=float32)
pos[-1]             : array([ 1.,  1.,  1.], dtype=float32)
len vel             : 10
vel[0]              : array([ 0.,  0.,  0.], dtype=float32)
vel[-1]             : array([ 0.,  0.,  0.], dtype=float32)
len mass            : 10
mass[0]             : 0.10000000149
mass[-1]            : 0.10000000149
len num             : 10
num[0]              : 0
num[-1]             : 9
len tpe             : 10
tpe[0]              : 0
tpe[-1]             : 0
```

In this case, you can see that the class automatically intitialize other arrays variables (vel, mass, num and rsp) with default values. Only the first and the last element of each defined vector are displyed by the methode info. All defined arrays and array elements may be easily accessible using the numarray convensions. For exemple, to display and change the positions of the tree first particles, type:

```
>>> nb.pos[:3]
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]], type=float32)
>>> nb.pos[:3]=2*ones((3,3),float32)
>>> nb.pos[:3]
array([[ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.]], type=float32)
```

3.1.2 Open from existing file

Now, lets try to open the gadget snapshot gadget_z00.dat. This is achieved by typing:

```
>>> nb = Nbody('gadget_z00.dat',ftype='gadget')
```

Again, informatins on this snapshot may be obtained using the instance info():

```
>>> nb.info()
---------------------------------
particle file       : ['gadget_z00.dat']
ftype               : 'Nbody_gadget'
mxntpe              : 6
nbody               : 20560
nbody_tot           : 20560
npart               : array([ 9160, 10280,     0,     0,  1120,     0])
npart_tot           : array([ 9160, 10280,     0,     0,  1120,     0])
mass_tot            : 79.7066955566
byteorder           : 'little'
pio                 : 'no'

len pos             : 20560
pos[0]              : array([-1294.48828125, -2217.09765625, -9655.49609375], dtype=float32)
pos[-1]             : array([ -986.0625    , -2183.83203125,  4017.04296875], dtype=float32)
len vel             : 20560
vel[0]              : array([ -69.80491638,   60.56475067, -166.32981873], dtype=float32)
vel[-1]             : array([-140.59715271,  -66.44669342,  -37.01613235], dtype=float32)
len mass            : 20560
mass[0]             : 0.00108565215487
mass[-1]            : 0.00108565215487
len num             : 20560
num[0]              : 21488
num[-1]             : 1005192
len tpe             : 20560
tpe[0]              : 0
tpe[-1]             : 4

atime               : 1.0
redshift            : 2.22044604925e-16
flag_sfr            : 1
flag_feedback       : 1
nall                : [ 9160 10280     0     0  1120     0]
flag_cooling        : 1
num_files           : 1
boxsize             : 100000.0
omega0              : 0.3
omegalambda         : 0.7
hubbleparam         : 0.7
flag_age            : 0
flag_metals         : 0
nallhw              : [0 0 0 0 0 0]
flag_entr_ic        : 0
critical_energy_spec: 0.0

len u               : 20560
u[0]                : 6606.63037109
u[-1]               : 0.0
len rho             : 20560
rho[0]              : 7.05811936674e-11
```

```
rho[-1]             : 0.0
len rsp             : 20560
rsp[0]              : 909.027587891
rsp[-1]             : 0.0
len opt             : 20560
opt[0]              : 446292.5625
opt[-1]             : 0.0
```

You can obtain informations on physical values, like the center of mass or the total angular momentum vector by typing:

```
>>> nb.cm()
array([-1649.92651346,   609.98256428, -1689.04011033])
>>> nb.Ltot()
array([-1112078.125 ,  -755964.1875, -1536667.125 ], dtype=float32)
```

In order to visualise the model in position space, it is possible to generate a surface density map of it using the display instance:

```
>>> nb.display(size=(10000,10000),shape=(256,256),palette='light')
```

You can now performe some operations on the model in order to explore a specific region. First, translate the model in position space:

```
>>> nb.translate([3125,-4690,1720])
>>> nb.display(size=(10000,10000),shape=(256,256),palette='light')
>>> nb.display(size=(1000,1000),shape=(256,256),palette='light')
```

Ou can now rotate around:

```
>>> nb.rotate(angle=pi)
>>> nb.display(size=(1000,1000),shape=(256,256),palette='light')
```

You can now display a temperature map of the model. First, create a new object with only the gas particles:

```
>>> nb_gas = nb.select('gas')
>>> nb_gas.display(size=(1000,1000),shape=(256,256),palette='light')
```

now, display the temperture mass-weighted map:

```
>>> nb_gas.display(size=(1000,1000),shape=(256,256),palette='rainbow4',mode='T',filter_name='gaussian
```

3.1.3 Selection of particles

You can select only particles within a radius smaller tha 500 (in user units) with respect to the center:

```
>>> nb_sub = nb.selectc((nb.rxyz()<500))
>>> nb_sub.display(size=(1000,1000),shape=(256,256),palette='light')
```

Now, rename the new model and save it:

```
>>> nb_sub.rename('gadget_z00_sub.dat')
>>> nb_sub.write()
```

A new gadget file has been created and saved in the current directory. We can now select particles as a function of the temperature. First, display the maximum temperature among all gas particles, then selectc particles and finally save in 'T11.num' the identifier (variable num) of these particles:

```
>>> log10(max(nb_gas.T()))
12.8707923889
>>> nb_sub = nb_gas.selectc( (nb_gas.T()>1e11) )
>>> nb_sub.write_num('T11.num')
```

Now open a new snapshot, from the same simulation, but at different redshift and find the particles in previous snapshot with temperature higher than 10^{11}:

```
>>> nb = Nbody('gadget_z40.dat',ftype='gadget')
>>> nb.display(size=(10000,10000),shape=(256,256),palette='light')
>>> nb_sub = nb.selectp(file='T11.num')
>>> nb_sub.display(size=(10000,10000),shape=(256,256),palette='light')
```

Now, instead of saving it in a gadget file, save it in a binary file type. You simply need to call the set_ftype instance before saving it:

```
>>> nb = nb.set_ftype('binary')
>>> nb.rename('binary.dat')
>>> nb.write()
```

3.1.4 Merging two models

As a last example, we show how two **pNbody** models can be easyly merged with only 11 lines:

```
>>> nb1 = Nbody('disk.dat',ftype='gadget')
>>> nb2 = Nbody('disk.dat',ftype='gadget')
>>> nb1.rotate(angle=pi/4,axis=[0,1,0])
>>> nb1.translate([-150,0,0])
>>> nb1.vel = nb1.vel + [50,0,0]
>>> nb2.rotate(angle=pi/4,axis=[1,0,0])
>>> nb2.translate([+150,0,50])
>>> nb2.vel = nb2.vel - [50,0,0]
>>> nb3 = nb1 + nb2
>>> nb3.rename('merge.dat')
>>> nb3.write()
```

Now display the result from different point of view:

```
>>> nb3.display(size=(300,300),shape=(256,256),palette='lut2')
>>> nb3 = nb3.select('disk')
>>> nb3.display(size=(300,300),shape=(256,256),palette='lut2',view='xz')
>>> nb3.display(size=(300,300),shape=(256,256),palette='lut2',view='xy')
>>> nb3.display(size=(300,300),shape=(256,256),palette='lut2',view='yz')
>>> nb3.display(size=(300,300),shape=(256,256),palette='lut2',xp=[-100,0,0])
```

or save it into a gif file:

```
>>> nb3.display(size=(300,300),shape=(256,256),palette='lut2',xp=[-100,0,0],save='image.gif')
```

3.2 Using pNbody with scripts

In addition to using **pNbody** in the python interpreter, it is very useful to use **pNbody** in python scripts. Usually a python script begin by the line #!/usr/bin/env python and must be executable:

```
chmod a+x file.py
```

The following example (slice.py), we show how to write a script that opens a gadget file, select gas particles and cut a thin slice

$$-1000 < y < 1000$$

The new files are saved using the extension .slice.

```
#!/usr/bin/env python

import sys
from pNbody import *

files = sys.argv[1:]

for file in files:
  print "slicing",file
  nb = Nbody(file,ftype='gadget',pio='yes')
  nb = nb.select('gas')
  nb = nb.selectc((fabs(nb.pos[:,1])<1000))
  nb.rename(file+'.slice')
  nb.write()
```

In your `pnbody_example` directory, you can run this script with the command:

```
./scripts/slice.py gadget_z*0.dat
```

or:

```
python ./scripts/slice.py gadget_z*0.dat
```

3.3 Using pNbody in parallel

With **pNbody**, it is possible to run scripts in parallel, using the `mpi` libary. You need to have of course `mpi` and `mpi4py` installed. To check your installation, try:

```
mpirun -np 2 pNbody_mpi
```

you should get:

```
This is task 0 over 2
This is task 1 over 2
```

but if you get:

```
This is task 0 over 1
This is task 0 over 1
```

this means that something is not working correctly, and you should check your path or `mpi` and `mpi4py` installation before reading further.

The prevous scripts `scripts/slice.py` can diretely be run in paralle. This is simply obtained by calling the `mpirun` command:

```
mpirun -np 2 scripts/slice.py gadget_z*0.dat
```

In this simple script, only the processus of rank 0 (the master) open the file. The content of the file (particles) is then distributed among all the other processors. Eeach processor recives a fraction of the particles. Then, the selection of gas gas particles and the slice are preformed by all processors on their local particles. Finally, the `nb.write()` command, run by the master, gather all particles and write the output file.

3.3.1 Parallel output

With **pNbody**, its possible to write files in parallel, i.e., each task write its own file. We can do this in the previous script simply by adding the line `nb.set_pio('yes')`. This tells **pNbody** to write files in parallel when `nb.write()` is called. The content of the new scripts `scripts/slice-p1.py` is:

```
#!/usr/bin/env python

import sys
from pNbody import *

files = sys.argv[1:]

for file in files:
  print "slicing",file
  nb = Nbody(file,ftype='gadget')
  nb = nb.select('gas')
  nb = nb.selectc((fabs(nb.pos[:,1])<1000))
  nb.rename(file+'.slice')
  nb.set_pio='yes'
  nb.write()
```

We can now run it:

```
mpirun -np 2 scripts/slice-p1.py gadget_z00.dat
```

This creates two new files:

```
gadget_z00.dat.slice.1
gadget_z00.dat.slice.0
```

The files have the same name than the initial name given in `Nbody()` with an extention `.i` where `i` corresponds to the processus rank. Each file contains the particles attributed to the corresponding task.

3.3.2 Parallel input

Now, it possible to start by reading these two files in parallel instead of asking only the master to read one file:: In our script, we add the optional argument `pio='yes'` when creating the object with `Nbody()`:

Note also that we have used `nb.set_pio('no')`. This force at the end the file te be written only by the master.

> #!/usr/bin/env python
>
> import sys from pNbody import *
>
> files = sys.argv[1:]
>
> **for file in files:** print "slicing",file nb = Nbody(file,ftype='gadget',pio='yes') nb = nb.select('gas') nb = nb.selectc((fabs(nb.pos[:,1])<1000)) nb.rename(file+'.slice.new') nb.set_pio('no') nb.write()

When we lunch it:

```
mpirun -np 2 scripts/slice-p2.py gadget_z00.dat.slice
```

the two files `gadget_z00.dat.slice.0` and `gadget_z00.dat.slice.1` are read each by one task, processed but at the end only the master write the final output : *gadget_z00.dat.slice.slice.new'*.

3.3.3 More on parallelisme

Lets try two other scripts. The first one (`findmax.py`) try to find the radial maximum distance among all particles and the center. It illustrate the difference between using `max()` wich gives the local maximum (maximum among particles of the node) and `mpi.mpi_max()` which gives the global maximum among all particles:

```
#!/usr/bin/env python

import sys
from pNbody import *

file = sys.argv[1]

nb = Nbody(file,ftype='gadget',pio='yes')
local_max  = max(nb.rxyz())
global_max = mpi.mpi_max(nb.rxyz())

print "proc %d local_max = %f global_max = %f"%(mpi.ThisTask,local_max,global_max)
```

When running it, you should get:

```
mpirun -np 2 ./scripts/findmax.py gadget_z00.dat.slice
proc 1 local_max = 8109.682129 global_max = 8109.682129
proc 0 local_max = 7733.846680 global_max = 8109.682129
```

which illustrate clearly the point. Finally, the latter script shows that even graphical functions support parallelisme. The script `showmap.py` illustrate this point by computing a map of the model:

```
#!/usr/bin/env python

import sys
from pNbody import *

file = sys.argv[1]

nb = Nbody(file,ftype='gadget',pio='yes')
nb.display(size=(10000,10000),shape=(256,256),palette='light')
```

When running

```
mpirun -np 2 ./scripts/showmap.py gadget_z00.dat.slice
```

you get an image of the model. The mapping has been performed independently by two processors.

CHAPTER

FOUR

SETTING A FORMAT FILE

CHAPTER

FIVE

DISPLAY MODELS