University of Amsterdam

Anton Pannekoek instituut

Computational Astrophysics (CA) Simulating Ultra Compact Dwarf Galaxies with AMUSE

Author:
Timo Halbesma, 1603221

Supervisor:
Prof.dr. S.F. Portegies Zwart
Edwin van der Helm, MSc

Listing 1: TLRH's solution for CA / Gravitational Dynamics

```python
#!/usr/bin/env amuse
"""
computationalastrophysics_gravitationaldynamics_timohalbesma_s1603221.py

Computational Astrophysics / Gravitational Dynamics
Simulating Ultra Compact Dwarf Galaxies with AMUSE / assignment 1A.

Timo Halbesma, s1603221.
October 14, 2014. Version 1.0.
"""
from time import time

import numpy  # as np
from matplotlib import pyplot  # as plt
from amuse.units import units

from solve_nbody import nbody_integrator


def assignment_1a():
    options = dict()
    stars = numpy.array([1, 2, 4, 8, 16, 32, 65, 128,
                         256, 512, 1024], dtype=numpy.int16)
    t_end = numpy.array([1, 2, 4, 8, 16, 32], dtype=numpy.int8)

    runtime_of_N_and_t = numpy.zeros((len(stars), len(t_end)),
                                     dtype=numpy.float64)
    dE_of_N_and_t = numpy.zeros((len(stars), len(t_end)),
                                dtype=numpy.float64)

    options['mcl'] = 10**7 | units.MSun
    options['rcl'] = 10 | units.parsec
    options['n_steps'] = 100
    options['algorithm'] = "BHTree"
    options['assignment'] = "1b"

    for i, N in enumerate(stars):
        for j, t in enumerate(t_end):
            options['Ncl'] = N
            options['t_end'] = t | units.Myr
            t_start = time()
            dE_of_N_and_t[i][j] = nbody_integrator(**options)
            runtime_of_N_and_t[i][j] = (time() - t_start)
            print str(N) + ', ' + str(t) + ', ' + str(time() - t_start)\
                + ', ' + str(dE_of_N_and_t[i][j])
    return stars, t_end, dE_of_N_and_t, runtime_of_N_and_t


def make_plot(stars, t_end, to_plot, choice):
    colors = {0: 'r', 1: 'y', 2: 'g', 3: 'b', 4: 'c', 5: 'm'}

    labelled = False
    fig, ax = pyplot.subplots()
    for i, N in enumerate(stars):
        for j, integration_time in enumerate(t_end):
            print "N = {0}, t = {1}, dE = {2}, to_plot = {3}"\
                .format(N, integration_time, to_plot[i][j], dE[i][j])
            if not labelled:
                ax.scatter(N, to_plot[i][j],
                           color=colors.get(j, 'k'),
                           label="t={0} Myr".format(integration_time))
            else:
                ax.scatter(N, to_plot[i][j],
                           color=colors.get(j, 'k'))
        labelled = True

    if choice == "runtime":
        ax.set_title("Wall-clock runtime as a function of N and t_end")
        ax.set_ylabel("Wall-clock time (s)")
    elif choice == "dE":
        ax.set_title("Relative energy error as a function of N and t_end")
        ax.set_ylabel("relative energy error dE")

    ax.set_xlabel("N")
    ax.set_xscale('log', basex=2)
    ax.legend(loc=2)  # upper left
    pyplot.savefig("CA_GD_TLRH_s1603221_{0}".format(choice))
```

```
if __name__ in '__main__':
    stars, t_end, dE, runtime = assignment_1a()
    make_plot(stars, t_end, runtime, "runtime")
    make_plot(stars, t_end, dE, "dE")
    pyplot.show()
```

Listing 2: TLRH's solution for CA / Gravitational Dynamics

```
"""
    Visualization for simple N-body integration.
    Reads particle set from file (nbody.hdf5) and prints
        subsequent frames.
"""
from matplotlib import pyplot
from amuse.lab import read_set_from_file  # take a hike with your *
from amuse import plot as aplot # scatter, xlabel, ylabel


def plot_cluster(filename="nbody.hdf5"):
    """ Plot file nbody.hdf5 """
    pyplot.ion()  # Turn interactive mode on.
    stars = read_set_from_file(filename, format='hdf5')
    lim = 10*stars.center_of_mass().length().value_in(stars.x.unit)
    m = 1 + 3.0*stars.mass/min(stars.mass)

    for si in stars.history:
        time = si.get_timestamp()
        pyplot.title("Cluster at t="+str(time))
        print "time =", time
        aplot.scatter(si.x, si.y, s=m)  # s size (in point^2)
        aplot.xlabel("X")
        aplot.ylabel("Y")
        pyplot.xlim(-lim, lim)
        pyplot.ylim(-lim, lim)
        pyplot.draw()  # Redraw the current figure (interactive mode).
        pyplot.cla()  # Clear the current axes.


def new_option_parser():
    """ Set options """
    from optparse import OptionParser
    result = OptionParser()
    result.add_option("-f", dest="filename", default="nbody.hdf5",
                      help="output filename [nbody.hdf5]")
    return result

if __name__ in ('__main__', '__plot__'):
    o, arguments = new_option_parser().parse_args()
    plot_cluster(**o.__dict__)
```

Listing 3: TLRH's solution for CA / Gravitational Dynamics

```
# from amuse.lab import *  # meh...
from amuse.lab import BHTree, Hermite, zero
from amuse.lab import new_plummer_model, write_set_to_file
from amuse.lab import  set_printing_strategy
from amuse.units import units, nbody_system
from amuse.community.mmc.interface import mmc


def nbody_integrator(Ncl, mcl, rcl, t_end, n_steps,
                     algorithm, assignment):
    verbose = False
    converter = nbody_system.nbody_to_si(mcl, rcl)
    bodies = new_plummer_model(Ncl, convert_nbody=converter)

    # Allow selecting Stellar Dynamics code trough a function argument.
    if algorithm == "BHTree":
        gravity = BHTree(converter)
    elif algorithm == "Hermite":
        gravity = Hermite(converter)
    elif algorithm == "mmc":
        gravity = mmc(converter)
    if assignment == "1B":
        gravity.parameters.timestep = 0.03125
```

```
    gravity.particles.add_particles(bodies)
    channel_from_gravity_to_framework = gravity.particles.\
        new_channel_to(bodies)

    # If you were smart in assignment 1A". No, but I was smart enough
    # to read all assignments prior to writing the code.
    if assignment == "1A" and Ncl == 1024 and t_end == 32 | units.Myr:
        write_set_to_file(bodies.savepoint(0.0 | t_end.unit),
                          "nbody_Hermite_1024_32Myr.hdf5",
                          "hdf5", append_to_file=False)

    Etot_init = gravity.kinetic_energy + gravity.\
        potential_energy

    time = zero
    dt = t_end / float(n_steps)
    while time < t_end:
        Etot_prev = Etot_init
        time += dt

        gravity.evolve_model(time)
        channel_from_gravity_to_framework.copy()
        if assignment == "1A" and Ncl == 1024 and t_end == 32 | units.Myr:
            write_set_to_file(bodies.savepoint(time),
                              "nbody_Hermite_1024_32Myr.hdf5",
                              "hdf5")

        Ekin = gravity.kinetic_energy
        Epot = gravity.potential_energy
        Etot = Ekin + Epot

        # I had some concerns writing to stdout could lower the runtime
        # (wall-clock) because it tends to be slow.
        if verbose:
            print "T =", time, "M =", bodies.mass.sum(), "E =", Etot, "Q =",\
                Ekin / Epot,
            print "dE =", (Etot_init - Etot) / Etot, "ddE =", (Etot_prev -
                Etot) / Etot

    gravity.stop()

    # For assignment we are interested in dE, so we return it.
    if assignment == "1A" or assignment == "1b":
        return (Etot_init - Etot) / Etot


def new_option_parser():
    from amuse.units.optparse import OptionParser
    result = OptionParser()
    result.add_option("-N", dest="Ncl", type="int", default=100,
                      help="number of stars [%default]")
    result.add_option("-t", unit=units.Myr, dest="t_end", type="float",
                      default=1 | units.Myr,
                      help="end time of the simulation [%default]")
    result.add_option("-n", dest="n_steps", type="float", default=100,
                      help="number of output steps [%default]")
    result.add_option("-m", unit=units.parsec, dest="mcl", type="float",
                      default=10**7 | units.MSun,
                      help="cluster mass [%default]")
    result.add_option("-r", unit=units.parsec, dest="rcl", type="float",
                      default=10 | units.parsec,
                      help="cluster half-mass radius [%default")
    result.add_option("-A", dest="algorithm", type="string",
                      default="BHTree", help="algorithm choice [%default]")
    return result


if __name__ in '__main__':
    set_printing_strategy("custom",
                          preferred_units=[units.MSun, units.RSun, units.yr],
                          precision=4, prefix="", separator=" [",
                          suffix="]")
    o, arguments = new_option_parser().parse_args()
    main(**o.__dict__)
```

Assignment 1A

In Figure 1 and Figure 2 the required plots for this assignment can be found.

The cluster size r is passed to the nbody_integrator function as parameter named 'rcl'. This parameter is only used in amuse.units.nbbody_system, which is responsible for the creation of bodies within a convined cluster with half-mass radius 'rcl'. The distance between individual particles could increase as the cluster half mass radius increases. This maximum distance between particles scales linearly with the cluster half mass radius. The distance between particles can be found in the force, but the number of times the force is calculated does not depend on it. If the cluster size increases and the number of paricles is unchanged, then at a certain point in time more particles could be further away. In that case more particles will be bundles together in the same BHTree, thus, in principle the calculation time could decrease as the cluster size r increases if particles move further away.

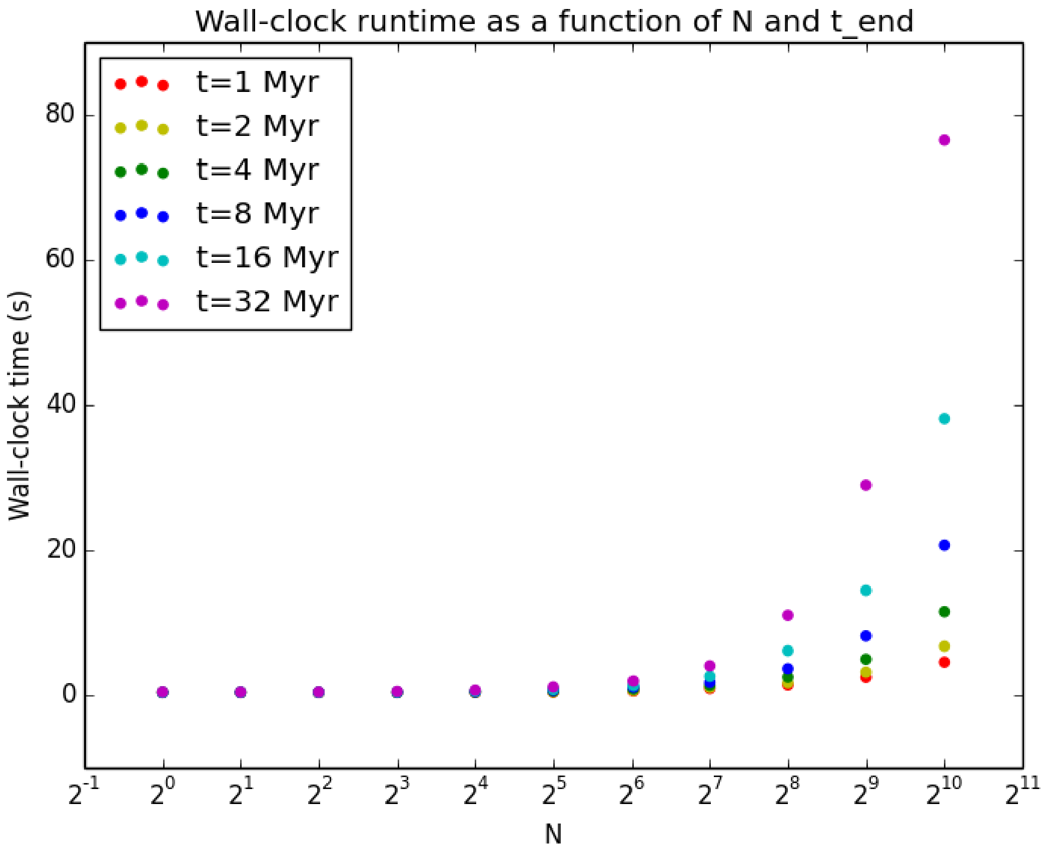

Figure 1: Wall-clock time as a function of both N and integration end time.

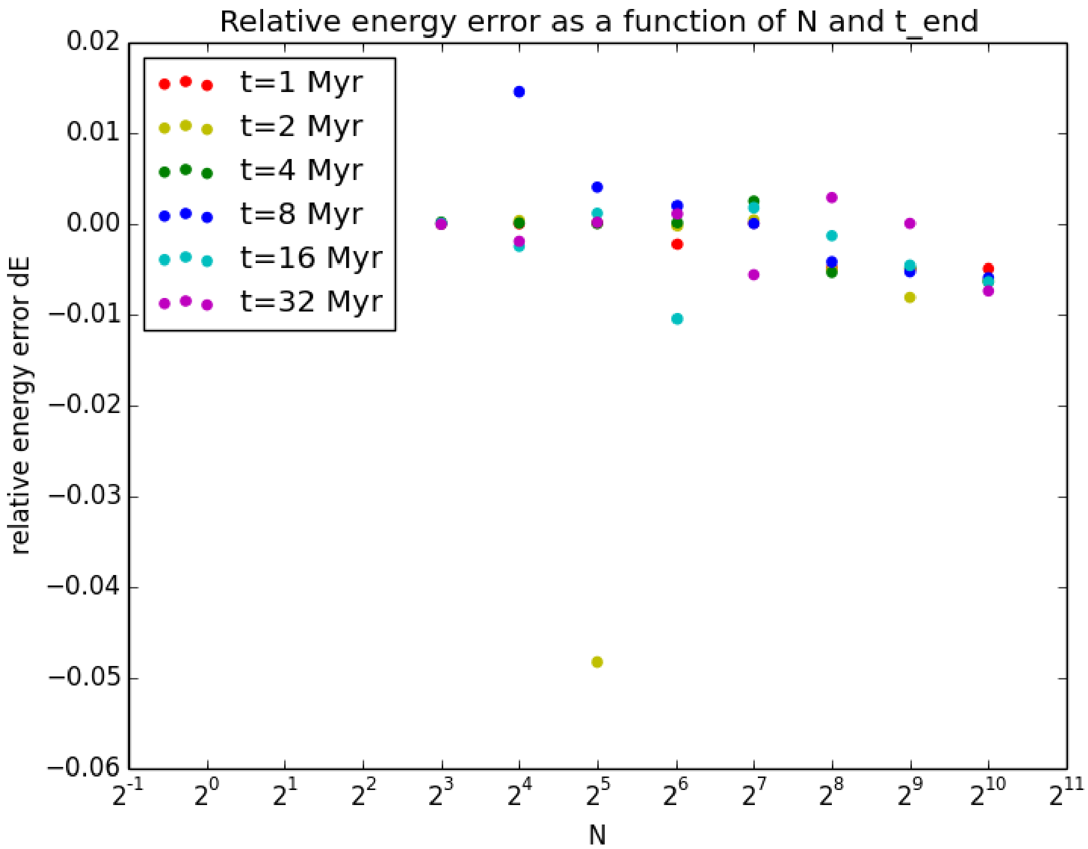

Figure 2: Relative energy error as a function of both N and integration end time.

Assignment 1B

Assignment 1C

Assignment 1D

Assignment 1E

Assignment 1F