GalaxSee Program Description

Compiling and running the code

The source code for GalaxSeeHPC requires a unix-like environment, such as MacOS X, Linux, or the Cygwin environment for Windows users. You will also need a C compiler, and a working copy of "make" (standard on most unix-like systems.) Additional libraries which are not required to use GalaxSeeHPC, but can extend the number of features available include the ability to display X11 graphics (top-down and side view) and to compile against the X11 development libraries for basic X11 display; GD libraries to create snapshots in jpg format (top-down and side-view); OpenGL, SDL, and pthreads for interactive graphics with perspective; and FFTW3 for the ability to use Particle-Particle-Particle-Mesh force calculation along with periodic boundary conditions. If you do not have these features on your system, you can disable them in the Makefile before compiling the code.

Download and unpack the source code for GalaxSeeHPC (galaxsee_hpc.tgz). You will notice a number of files ending in ".c" and ".h." These are the c code and header files that you will compile. You will see a sample input file, test.gal. Additionally, you will see some helper files, such as two "qsub" files that can be used to submit jobs on a cluster with a PBS scheduler, and a file "make_pov.perl" along with three POVRay include files that allow you to use the ray tracing program POVRay to make high resolution animations from snapshot files taken throughout a single simulation. Finally, notice the file "Makefile." This file contains all of the instructions and options that your compiler will need to build the code.

Makefile contains a series of options that will define what compiler is to be used, and defines special options for that compiler.

```
#CC            = /opt/mpich/intel/bin/mpicc
CC             = icc
#CC            = mpicc
#CC            = gcc
CFLAGS         = -Wall
LDFLAGS        = -o
```

```
LIBS          = -lm
```

You should set CC equal to the compiler you are using on your system. For most systems, this will be the GNU C Compiler, gcc, or if you are on a multi-core platform or a cluster and plan to run the code in parallel, you will want to specify your MPI C compiler, typically "mpicc." You can specify typical flags to pass to the compiler at compile time and link time (CFLAGS and LDFLAGS), as well as system libraries against which you will link your code ("LIBS = -lm" specifies that you will compile against the math library, for example.)

```
###### DEBUG OPTIONS
CFLAGS += -g

###### OPTIMIZATION OPTIONS
#CFLAGS += -O3
```

Additionally, you may choose to use additional debugging or optimization options. These will vary by compiler, but "-g" is a typical flag to tell your compiler that you want to run in "debug" mode, where additional information about the process of running the code is kept, in case you want to us a package like "gdb" (GNU Debugger) or "valgrind" (a memory error detection tool) to analyze the code. Running in debug mode will make the code run somewhat slower.

Optimization options vary greatly from one compiler to another and from one platform to another, but most compilers give you the option of a "shortcut" for common combinations of optimizations, where "O1," "O2," and "O3" are levels of increasing optimization, though you may find with some codes on some systems, too much optimization may result in unexpected results.

You will typically not use both optimization options and debug options at the same time, but would use one or the other. Note that in the example shown here, the optimization options in the Makefile are commented out by adding a hash symbol "#" at the front of the line.

```
###### LIBGD OPTIONS
#CFLAGS += -DHAS_LIBGD
#LIBS += -ljpeg -lpng -lz -lgd

###### X11 OPTIONS
CFLAGS += -DHAS_X11
```

```
LIBS += -L/usr/X11R6/lib64/
LIBS += -lX11

###### SDL OPTIONS
#CFLAGS += -DHAS_SDL -D_REENTRANT -D_USE_PTHREADS
#LIBS += -lGL -lGLU -lSDL
```

GalaxSeeHPC has a number of update options that allow you to view the results of each timestep in the simulation, some of which assume that you have additional software available on the machine. If you plan to write results as a series of jpg images directly to disk, you would want to ensure that LIBGD is installed on your system, and you would make sure that the LIBGD options are uncommented in your Makefile. If your version of LIBGD is installed in a non-typical location, you may need to use the '-L/path/to/library' LIBS option, and/or the '-I/path/to/include' CFLAGS option.

Similarly, if you plan to use the X11 or SDL visualization options, you will need to make sure that they are installed on your system, uncomment those CFLAGS and LIBS options before compiling, and if needed locate the actual library and include locations on your computer and add those to your Makefile options as well.

```
###### MPI OPTIONS
#CFLAGS += -DHAS_MPI
#LIBS += -lmpi

###### FFTW OPTIONS
CFLAGS += -DHAS_FFTW3 -DUSE_PPPM
CFLAGS += -I/opt/fftw3/gcc/include
LIBS += -L/opt/fftw3/gcc/lib
LIBS += -lfftw3
```

Your MPI and FFTW options allow you to use MPI to run the program in parallel and to use FFTW3 in order to use a Particle-Particle-Particle-Mesh periodic force calculation, respectively. If you plan to use these, make sure that they are installed on your system, and uncomment the appropriate Makefile lines.

```
OBJS          = nbody.o\
                text11.o\
                octtree.o\
                mem.o\
                rand_tools.o\
                quaternion.o\
```

```
                    sdlwindow.o\
                    fcr.o\
                    pppm.o\
                    cubeinterp.o\
                    readline.o\
                    galaxsee.o
```

The OBJS variable in your Makefile is your object code. Notice that for every ".c" file in your directory, there is a ".o" file listed here. This is the list of object files that you want the compiler to create, and then link together. If you extend the GalaxSee HPC program and create new ".c" files, make sure you add the corresponding ".o" file here.

Having the Makefile separate building each individual object from linking objects together allows for more efficiency when modifying large codes, as if you make a change in a single file, only that one file needs to be rebuilt.

** If you do choose to extend GalaxSee by adding additional code in new files, make sure to add them as ".o" and not ".c" in your Makefile, as it is possible to write over your code by doing this **

```
PROGRAM         = galaxsee

all:            $(PROGRAM)
```

PROGRAM gives the name of the executable to be built. "all" gives the default object to be built if make is executed with no arguments.

```
$(PROGRAM):     $(OBJS)
                $(CC) $(OBJS) $(LDFLAGS) $(PROGRAM) $(LIBS)
```

The $(PROGRAM) line tells make how to build the program, that it will first build all objects, and then use the compiler specified to link those objects with the given link flags, with the executable name provided, linking against the libraries specified. Note that Make does not need a specific rule on how to compile C files, but will automatically use your CFLAGS and CC options that you have provided.

```
clean:;         rm -f $(OBJS) $(PROGRAM)

vidclean:;      rm -rf out*.png out*.pov
```

```
povfiles:;      for f in *.dump; do make_pov.perl $$f >&
/dev/null; done

vidpov:;        for f in *.pov; do povray -H600 -W800 -D
$$f >& /dev/null; done

vidpovpbs:;    for f in *.pov; do qsub -v FILE=$$f
frames.qsub ; done

anim:;          convert -delay 1 -loop 0 -quality 100
out*png anim.gif
```

Finally, the Makefile ends with a few other common tasks that you may want to occasionally perform. “clean” removes the current executable and object code, so that the next build will rebuild everything from scratch. If you modify any Makefile options or anything in a header file, you want to build a clean executable. Vidclean, povfiles, vidpov, vidpovpuma, and anim are options specific to display. Vidclean removes files generated by the libgd and povray display options. Povfiles generates povray scen files from a sequence of “dump”ed snapshot files. Vidpov uses povray, if installed on your system, to process and render those scene files. Vidpovpbs can be used on a cluster with a PBS scheduler to schedule all of your povray scene renderings. Anim will use the convert program, provided by ImageMagick, to create an animated gif from a sequence of png files.

After you have made any modifications needed for your Makefile, run it on your computer using the “make” command. (I recommend making clean first to ensure a clean build.)

```
make
make clean
```

If everything works properly, you will see a sequence of comments and possibly warnings for your compiler, and eventually the program will finish compiling. The last line of your compilation step might look something like

```
icc nbody.o text11.o octtree.o mem.o rand_tools.o
quaternion.o sdlwindow.o fcr.o pppm.o cubeinterp.o
readline.o galaxsee.o -o  galaxsee -lm  -L/usr/X11R6/lib64/
-lX11 -L/opt/fftw3/gcc/lib -lfftw3
```

and when you type “ls” at the command prompt you will see an executable file “galaxsee” that has been created.

```
basicView.inc  galaxsee.c      nbody.c
pppm_structs.h  sdlwindow.c
cubeinterp.c   galaxsee.o      nbody.h
quaternion.c    sdlwindow.h
cubeinterp.h   galaxsee.qsub   nbody.o
quaternion.h    sdlwindow.o
cubeinterp.o   Makefile        octtree.c
quaternion.o    test.gal
CVS            make_pov.perl   octtree.h
rand_tools.c    text11.c
fcr.c          mem.c           octtree.o
rand_tools.h    text11.h
fcr.h          mem.h           octtree_structs.h
rand_tools.o    text11.o
fcr.o          mem.o           pppm.c
readline.c
frames.qsub    myShapes.inc    pppm.h
readline.h
galaxsee       myTextures.inc  pppm.o
readline.o
```

Check to see that the executable will run using the file "simple.gal"

```
galaxsee simple.gal
```

If there are errors, try compiling with fewer options. If that does not resolve the problem, buy your local guru a Mountain Dew™ and ask for help.

Display options

The UPDATE_METHOD variable controls how you will display each update, which is executed every X timesteps (X is set by the variable SKIP_UPDATES). Options include

- 1 Write a hash mark to the command window
- 2 Write current model time to the command window
- 4 Write all position information to the command window
- 8 Create a jpeg using GD libraries on disk
- 16 Display ASCII Art image to the command window
- 32 Display image using X11 to the screen

- 64 Display statistics to the command window
- 128 Display SDL/OpenGL window to the screen
- 256 Write all position information to a file on disk

You may ask yourself, why are the UPDATE_METHOD values ordered as powers of 2 instead of counting numbers? The UPDATE_METHOD variable is what is known as a bitmask, which allows you to set multiple values at the same time by adding them together. A value of UPDATE_METHOD equal to 34, for example, would be read as 32 + 2, or both display using X11 and write the current time to the command window.

If you expect to see a display as a result of setting the UPDATE_METHOD command and you do not, yet the code is running otherwise, make sure that you are not requesting a feature which has not been compiled into the code.

If you choose to extend the code and add your own update methods, you should follow the example used in nbody.g and nbody.h, and define each update method flag as a new power of 2, and use the code in nbody.c as an example of how to add a new update option.

Sample input file (test.gal)

If you open test.gal, you will see a large variety of input option. "Reasonable" defaults exist for these for a stellar cluster, and are presented with the test.gal file included in the distribution.

Model definitions

```
N 500                          # number of masses
TFINAL 1000.0                  # final time in time units
TIMESTEP 0.1                   # timestep
INITIAL_V 0.0                  # initial random velocity
in velocity units
ROTATION_FACTOR 0.0            # unitless rotation factor
(equilibrium ~ 1.0)
DRAG_COEFFICIENT 0.0           # coefficient of dynamical
friction
SCALE  13.0                    # 1/2 the "box" side length
MASS 800.0  a                  # total system mass
```

```
G 0.00449                       # Gravitational Constant
EXPANSION 0.0                   # expansion constant in
velocity units
```

Regarding units, the user chooses what units to use for time, length, and mass scales, and then should use derived units for velocity based on those, and should provide any constants, particularly the gravitational constant G and if used the universal expansion rate in a value appropriate to those units.

```
## INT_METHOD is the integration method
#  INT_METHOD_RK4          1
#  INT_METHOD_LEAPFROG     2
#  INT_METHOD_MPEULER      3
#  INT_METHOD_IEULER       4
#  INT_METHOD_EULER        5
#  INT_METHOD_ABM          6
##
INT_METHOD 1
```

A variety of choices are available for the integration method, the default is 4^{th} order Runge Kutta.

```
## FORCE METHOD determines how forces are calculated based
on position
#  FORCE_METHOD_DIRECT  1
#  FORCE_METHOD_TREE    2
#  FORCE_METHOD_PPPM    3
#
#   *NOTE   Some force calculation methods require
compilation
#           against external numerical libraries, and you
may
#           not be able to use all force methods if those
libraries
#           are not present and included in the Makefile at
compilation
##
FORCE_METHOD 1
```

Multiple force calculation methods are available, including a direct calculation, a tree-based Barnes-Hut calculation, and a FFT based Particle-Particle Particle-Mesh calculation. Note that PPPM methods require that you have FFTW3 installed on your system and include it in your compilation.

```
## FORCE_METHOD_TREE options
```

```
# TREE_RANGE COEFFICIENT is a scale factor determining how
far apart
#   two branches of the octtree must be at a given stage in
order
#   to allow an approximation to be made. This distance is
taken as
#   a constant multiple of the octtree node size at any
given stage.
##
TREE_RANGE_COEFFICIENT 1.2
```

The tree range coefficient is a parameter used in the Barnes-Hut force calculation.

```
## NGRID
#  The resolution of the grid used in PPPM force
calculation methods
##
NGRID 32

## KSIGMA and KNEAR
#  coefficients controlling a PPPM solution, they control
the
#  smoothing of point masses into a density distribution
and the
#  cutoffs for the Particle-Particle calculation for near
neighbors
#  For the purposes of mapping point particles to a density
distribution
#  on a grid the density function per point mass is assumed
to be
#  normal with a standard deviation given by
#     KSIGMA * (2 * SCALE) / NGRID
#  and nearest neighbors are any objects within
#     KNEAR * (2 * SCALE) / NGRID
#  To use a PM solution instead of a PPPM solution set
KNEAR to zero.
##
KSIGMA 2.0
KNEAR 1.0
```

The Particle-Particle Particle-Mesh method has three parameters, the resolution of the grid, and coefficients to calculate the expanded particle distribution and the cutoff for nearest neighbors.

```
## DISTRIBUTION controls the initial distribution of masses
```

```
#  DISTRIBUTION_SPHERICAL_RANDOM 1
#  DISTRIBUTION_RECTANGULAR_RANDOM 2
#  DISTRIBUTION_RECTANGULAR_UNIFORM 3
##
DISTRIBUTION 1

## ANISOTROPY is in distanced units, and is a random shift
given
#  to particle positions in distributions that are
initially uniform
ANISOTROPY 0.01
```

The initial distribution has a variety of options. Uniform distributions additionally can be given an anisotropy argument which adds a random shift to all particles.

```
## Softened potentials
#    two softening options are given, srad factor is a
unitless coefficient
#     for which a "shield radius" is calculated for each
point mass M
#     so that close encounters within that shield radius
are neglected
#     SRAD = SRAD_FACTOR * (G * M * TIMESTEP^2 )^(1/3)
#    the other option is an additive term in the
denominator of force
#     calculation terms so that instead of a force of G M1
M2 / r^2
#     one uses F = G M1 M2 / (r + SOFT_FACTOR)^2
##
SRAD_FACTOR 5.0
SOFT_FACTOR 0.0
```

Two options exist if you need to soften the potential to eliminate ejections of objects due to numerical instability when treating close interactions. A shield radius can be used in the code, and if used will result in all forces being neglected when objects are closer than that shield radius. In the code, it is calculated based both on the object causing the acceleration and the time step, and the user provides a unitless coefficient. Additionally, a traditional softened potential can also be used.

```
## UPDATE_METHOD is a bitmask allowing you to layer
different display options
#  UPDATEMETHOD_HASH_TEXT 1                  # display a
hash mark every update
```

```
#  UPDATEMETHOD_BRIEF_TEXT 2                  # display
model time
#  UPDATEMETHOD_VERBOSE_POSITIONS 4           # display all
positions at update
#  UPDATEMETHOD_GD_IMAGE 8                    # create image
files using GD
#  UPDATEMETHOD_TEXT11 16                     # display
ascii art animation
#  UPDATEMETHOD_X11 32                        # display X11
image
#  UPDATEMETHOD_VERBOSE_STATISTICS 64         # display
energy statistics
#  UPDATEMETHOD_SDL 128                       # display
SDL/OpenGL image
#  UPDATEMETHOD_DUMP 256                      # write raw
data to file at update
#
#  If you wanted to include both an X11 display and brief
text to the terminal
#  for example, you would use 32 + 2 = 34
#  UPDATE_METHOD 34
#
#  *NOTE   Some update options require compilation against
other system
#          libraries, and you may not be able to use all
visualization options
#          if they were not included in the Makefile and
present on the
#          system at compilation.
##
UPDATE_METHOD 16
```

The update method controls how the user will see the results of the simulation, and is set as a bitmask so that multiple methods can be used simultaneously.

```
## SHOW_UPDATES is either zero or positive to allow a
display to be shown
SHOW_UPDATES 1

## SKIP_UPDATES allows the user to skip the number of
timesteps between
#  refreshed information about the model
##
SKIP_UPDATES 10
```

```
## FILE_PREFIX is a string denoting the prefix of all
output files
#  produced by the code
##
FILE_PREFIX out
```

Additional options exist to control how the output is processed, including a single flag that can be used to shut off all output.

galaxsee.c (main routine)

galaxsee.c contains the main routine that will run the program. It looks for command line input to check to see if there is an input file, and takes input from STDIN if there is not an input file listed. It parses the input file, sets reasonable defaults for the model, and overrides them using the input file when specified. It calls routines from *nbody.c* to execute the code in an iterative loop over the requested model time and timestep, and completes output when done. You will notice some additional options in the code if specific features, particularly MPI or SDL options, that have been blocked out with "#ifdef" statements. This allows you to define variables in the Makefile to include certain portions of the code at compile time.

The compute loop is broken out into its own routine so that, if an update method that requires a threaded event handling loop (such as the SDL display option) is needed, that provided that the user has pthreads on their machine, the infrastructure is in place for the code to be modified to add a different update method with threaded event handling.

nbody.c and nbody.h

nbody.h defines the basic memory structure of the galaxsee model. It includes many variables that are obviously what you would expect (arrays for x,y,z, vx,vy,vz, and mass, for example,) and some that are not so obvious (a integer counter abmCounter to determine how many past iterations have been performed when restarting the Adams-Bashfourth-Moutlon integration method, for example.) Comments have been included in *nbody.h* describing each of these variables.

nbody.c contains the library routines needed to drive the simulation. Most of the methods in the library can be categorized as "set" routines used to set default values, "step" routines used to define how to integrate the model one step

forward, or "calc" routines used to calculate the interparticle forces. Additional routines exist to allocate and free memory, to handle update methods, or to initialize the problem.

octtree.c and octtree.h

The *octtree.c* routines and memory structures are used to perform a Barnes-Hut force calculation. Routines focus primarily on building the tree (*populateOctTree*), updating a built tree while keeping the structure the same (*resetOctTree*), and calculating forces from the tree (*calculateForceOctTree*).

pppm files

The *pppm.c* routines and memory structure are used to perform a Particle-Particle-Particle-Mesh force calculation, and primarily include routines to create a density distribution based on a particle distribution (*populateDensityPPPM*), to calculate the potential from that density using a Fourier transform and interpolate forces onto a grid (*prepPotentialPPPM*), and to calculate forces (*calculateForcePPPM*). *cubeinterp.c* is a helper file to handle the interpolation.

Miscellaneous Tools

mem.c is a set of memory allocation routines to simplify allocation of arrays and matrices. *rand_tools.c* are a collection of random number creation routines. *frc.c* is a fast cube root approximation which can be used to speed up the calculation of the shield radius, as speed is more important than precision in the calculation of the shield radius. *readline.c* are a collection of routines for reading and parsing the input file, allowing for automatic removal of whitespace, blank lines, and comments. *sdlwindow.c* handles the event and drawing loop for the SDL update option. *quaternion.c* is a quaterion rotation memory construct used in the SDL update option. *text11.c* is a ascii art memory construct used in the ascii art update option.