FleCSPH Wiki

August 31, 2020

This document contains static version of FleCSPH wiki page instruction for building FleCSPH and running several example tests. Full information about this wiki page can be found in https://github.com/laristra/flecsph/wiki This document is authorized for unlimited public release under LA-UR-20-26581

1 Building FleCSPH with Spack

FleCSPH can now be installed as a Spack package:

- Download spack at: github.com/spack/spack
- Follow installation instructions
- Use the following command to install `module` support for spack and load the module. The second line can be added in your `bash_profile.sh`

  ```
  spack install lmod
  . \$(spack location -i lmod)/lmod/lmod/init/bash
  ```

- Run:

  ```
  spack install flecsph
  ```

This will build all the dependencies, compile and install FleCSPH. In order to use FleCSPH executables simply run:

```
spack load flecsph
```

You will then have access to the generators and the drivers: - sodtube_{1-2-3}d_generator, sedov_{2-3}d_generator... - hydro_{1-2-3}d, newtonian_3d...
You can access to pre-configured parameter files and examples by downloading this repository:

```
git clone --recursive git@github.com:laristra/flecsph.git
cd flecsph
```

Sample parameter files and the intial data can be found in the `data` subdirectory.

1.1 Using Spack in the development workflow

If you have downloaded FleCSPH from github and working on a development branch, it is very convenient to use spack to automatically handle the dependencies:

1. Follow the steps above to install FleCSPH with spack. This will ensure that all the dependencies are satisfied.
2. To inspect the dependencies:

   ```
   spack module tcl loads --dependencies flecsph@refactor
   ```

If this command returns empty, use `spack bootstrap` for tcl.

3. Load the FleCSPH dependencies installed by spack into the `bash` environment:

   ```
   source <(spack module tcl loads --dependencies flecsph)
   ```

   Unload FleCSPH itself as you will be using your own custom built version:

   ```
   module unload $(spack module tcl find flecsph)
   ```

   Inspect your module environment to make sure dependencies have been loaded:

   ```
   module list
   ```

4. You can now build your development version with cmake as described below, skipping all the dependencies. cmake should find all the dependencies from what you loaded with spack:

   ```
   mkdir build; cd build
   cmake .. \
       -DCMAKE_BUILD_TYPE=debug \
       -DENABLE_UNIT_TESTS=ON   \
       -DENABLE_DEBUG=OFF       \
       -DLOG_STRIP_LEVEL=1
   ```

2 Building FleCSPH manually

Below we assume that FleCSPH is installed in FLECSPH root directory `${FLECSPH_ROOT}`.

2.1 Suggested directory structure

We recommend to use an isolated installation of FleCSPH and FleCSI, such that the software and all the dependencies in a separate directory, with the following directory structure:

```
${FLECSPH_ROOT}
 flecsi
  build
 flecsph
  build
  third-party-libraries
 local
     bin
     include
     lib
     lib64
     share
```

All the build happens in `build` subdirectories, and compiled dependencies are installed in `local` subdirectory. Make sure to set your CMAKE prefix to this location:

```
% export CMAKE_PREFIX_PATH=${FLECSPH_ROOT}/local
```

2.2 Prerequisites

You will need the following tools:

- C++17 - capable compiler, such as gcc version >= 7;
- git version > 2.14;
- MPI libraries;
- cmake version >= 3.15;
- boost library version > 1.59;
- Python version >= 3.6.
- HDF5 compiled with parallel flag version > 1.8
- GSL library

2.3 FleCSI

Clone FleCSI repo at the `stable/flecsph` branch. Checkout submodules recursively, then configure as below:

```
export CMAKE_PREFIX_PATH=${FLECSPH_ROOT}/local
cd ${FLECSPH_ROOT}
git clone --recursive git@github.com:laristra/flecsi.git
cd flecsi
git checkout stable/flecsph
git submodule update --recursive
mkdir build ; cd build
cmake .. \
    -DENABLE_OPENMP=OFF \
    -DCXX_CONFORMANCE_STANDARD=c++17 \
    -DENABLE_METIS=ON \
    -DENABLE_PARMETIS=ON \
    -DENABLE_COLORING=ON \
    -DENABLE_DEVEL_TARGETS=ON \
    -DENABLE_LOG=ON           \
    -DFLECSI_RUNTIME_MODEL=mpi
```

In this configuration, FleCSI is installed with the MPI backend.
Build as a final step:

```
% make -j
```

2.4 FleCSPH

Clone FleCSPH git repo:

```
cd ${FLECSPH_ROOT}
git clone --recursive git@github.com:laristra/flecsph.git
```

2.4.1 Building FleCSPH

Configure and build commands:

```
# in ${FLECSPH_ROOT}/build:
export CMAKE_PREFIX_PATH=${FLECSPH_ROOT}/local
cmake .. \
    -DCMAKE_BUILD_TYPE=debug \
    -DENABLE_UNIT_TESTS=ON   \
    -DENABLE_DEBUG=OFF       \
```

```
    -DLOG_STRIP_LEVEL=1
  make -j
  make install
```

3 Several Examples you may want to try

We provide several example tests to understand code structure, how you can run some problems. These examples will demonstrate our functionalities for hydrodynamics and gravity computation. Below link will show detailed descriptions on each example problems. To perform these examples, you must successfully build FleCSPH first.

3.1 1D Sod Shock Tube Test

3.1.1 Problem description

The Sod shock tubes provide standard tests with a classical Riemann problem for accuracy in computational fluid dynamics.

3.1.2 Generate initial data

First, we need to generate initial data. In this example, we use the standard test 1 and 10,000 particles. Copy the following parameter settings into a parameter file, e.g. `sod_test1_n10000.par`:

```
# Sodtube test #1 for 10000 particles in linear dimension

# initial data
  initial_data_prefix = "sod_test1_n10000"
  lattice_nx = 10000      # particle lattice linear dimension
  poly_gamma = 1.4        # polytropic index
  sodtest_num  = 1        # which test to generate (1..5)
  equal_mass = yes        # determines whether equal mass particles are used or equal separation
  sph_eta = 1.5
  lattice_type = 0        # only matters for dimensions > 1; in 3D: 0=rectangular, 1=hcp, 2=fcc lattice
  domain_type = 0         # 0:cube, 1:sphere
  box_length = 6.0

# evolution
  sph_kernel = "Wendland C6"
  initial_dt = 1e-12
  sph_variable_h = yes
  adaptive_timestep = yes
  timestep_cfl_factor = 0.75
  #thermokinetic_formulation = no
  final_iteration = 5000
  out_screen_every = 1
  out_scalar_every = 1
  out_h5data_every = 10
  out_diagnostic_every = 1
  output_h5data_prefix = "ev_sod_test1_n10000"
```

Generate initial data using the 1D generator:

```
mpirun -np 1 ./sodtube_1d_generator sod_test1_n10000.par
```

The file `sodtube_1d_generator` can be found in your build directory at `${PATH_TO_FLECSPH}/flecsph/build/app/id_generators`, or in the install directory, `${CMAKE_PREFIX_PATH}/bin`. The command above will produce a particle file `sod_test1_n10000.h5part`.

3.1.3 Run the test

Use the following command to produce hydrodynamic evolution:

```
mpirun -np 1 ./hydro_1d sod_test1_n10000.par
```

This will produce evolution files: `ev_sod_test1_n10000.h5part` contain particle data, and `scalar_reductions.dat` contain various scalar reductions: total mass, energy, internal energy, total momentum etc.

Plotting the data There are several different ways to plot the results. We encourage user to use whatever user's convenient way but please contact flecsph-support@lanl.gov if you would like to get plotting script from us.

3.2 Stellar Oscillation

3.2.1 Problem description

As a standard test of the numerical method for simulating self-gravitating fluids, we present the evolution of a stable isolated star in equilibrium. This test checks consistency and conservation properties for the coupled system of hydrodynamics and gravity.

3.2.2 Generate initial data

For initial data, we first solve the Lane-Emden equation. This results in a white dwarf with mass 0.2 solar mass and radius 4790 km. We provide radial density profile here. If you are interested in solving yourself, please check `wdID` directoy in `tools/starID_tools` directory.
Using this profile, we set up particle configuration on a perturbed icosahedral lattice. To do that, create `initial_n32.par` using below parameters

```
# Sperical particle distribution: initial data parameter file
#
  initial_data_prefix = "wd_initial_n32"

  # geometry: spherical domain with radius R = 1
  domain_type = 1   # 0:box, 1:sphere
  density_profile = "from file"
  input_density_file = "density_profile_nr8000.dat"
  sphere_radius = 4.79283e+08

  # icosahedra lattice with small perturbations
  lattice_nx =  32  # particle lattice dimension
  lattice_type = 3  # 0:rectangular, 1:hcp, 2:fcc, 3:icosahedral
  lattice_perturbation_amplitude = 0.09 # in units of sm. length

  # equation of state type and parameters
  eos_type = "polytropic"
  #poly_gamma = 1.66667   # polytropic index
  poly_gamma = 0.99

  # density and pressure for relaxation stage
  rho_initial = 5.2e+6
  #pressure_initial = 1.56085e+23
  pressure_initial = 1e+28

  # since we only need spherical distribution of particles,
  # set Sedov energy to zero
  sedov_blast_energy = 0.0
```

```
  # use a good kernel
  sph_kernel = "Wendland C6"
  sph_eta = 1.5
```

You can also directly get this parameter file here. Using this, we can generate initial data via below command

```
mpirun -np 1 ./sedov_3d_generator initial_n32.par
```

Note that you can find `sedov_3d_generator` in your `${PATH_TO_FLECSPH}/flecsph/build/app/id_generators` or `${CMAKE_PRE_PATH}/bin` based on your compilation procedure. After successful generation, you will have `wd_initial_n32.h5part` file.

3.2.3 Particle relaxation and generate self-consistent white dwarf

After generating initial configuration, we relax this particle configuration. First, create `relaxation_n32.par` using below parameters

```
# Sperical particle distribution: relaxation phase
# Use ./hydro_3d <this_file>.par to evolve into relaxed state

# initial data
  initial_data_prefix = "wd_initial_n32"
  initial_iteration = 0

  # geometry
  domain_type = 1   # 0:box, 1:sphere
  external_force_type = "spherical density support"
  density_profile = "from file"
  input_density_file = "density_profile_nr8000.dat"
  sphere_radius = 4.79283e+08

  # density and pressure for relaxation stage
  rho_initial = 5.2e+6
  ##pressure_initial = 1.56085e+23  # actual pressure in WD
  pressure_initial = 1e+28  # artificially increased for faster relaxation

# evolution
  final_iteration = 1000
  relaxation_steps = 1000
  relaxation_beta  = 1e2
  relaxation_gamma = 0.0

  initial_dt = 1.e-9
  timestep_cfl_factor = 0.5

  out_screen_every = 1
  out_scalar_every = 1
  out_diagnostic_every = 10
  out_h5data_every = 100
  output_h5data_prefix = "wd_relaxation_n32"

  thermokinetic_formulation = yes
  adaptive_timestep = yes
  sph_variable_h = yes
```

```
  evolve_internal_energy = no

  # equation of state type and parameters
  eos_type = "polytropic"
  ##poly_gamma = 1.66667   # actual polytropic index for WD
  poly_gamma = 0.99  # artificially lowered for more uniform relaxation

  sph_kernel = "Wendland C6"
  sph_eta = 1.5
```

or get it from here. After that, using below command for relaxation step

```
mpirun -np 1 ./hydro_3d relaxation_n32.par
```

Note that you can find `hydro_3d` in your `${PATH_TO_FLECSPH}/flecsph/build/app/drivers` or `${CMAKE_PRE_PATH}/bin` based on your compilation procedure. After successful generation, you will have `wd_relaxation_n32.h5part` file. After that, we modify the file produced in previous step by overwriting particles pressure and internal energies to correspond self-consistent white dwarf model. To do that, create `modify_n32.h5part` using below parameters

```
#
# Overwrite quantities before relaxation step
# Usage: ./sedov_3d_generator <this_file>.par
# WARNING: overwrites initial_data_prefix !!
#
  modify_initial_data = yes
  initial_data_prefix = "wd_modified_n32"
  initial_iteration = 1000

  # geometry: spherical domain with radius R = 1
  domain_type = 1   # 0:box, 1:sphere
  sphere_radius = 4.79283e+08

  # equation of state type and parameters
  eos_type = "polytropic"
  poly_gamma = 1.66667   # polytropic index

  # reset thermodynamical quantities
  density_profile = "from file"
  input_density_file = "density_profile_nr8000.dat"
  external_force_type = "spherical density support"
  rho_initial = 5.2e+6
  pressure_initial = 1.56085e+23

  sedov_blast_energy = 0.0
  sedov_blast_radius = 0.1 # whatever

  # good kernel
  sph_kernel = "Wendland C6"
  sph_eta = 1.5
```

or you can get it from here. Then, follow the below commands

```
cp wd_relaxation_n32.h5part wd_modified_n32.h5part
mpirun -np 1 ./sedov_3d_generator modify_n32.par
```

This will overwrite `wd_modified_n32.h5part`

3.2.4 Run evolution

Finally, we evolve white dwarf data from previous steps with self-gravity. To do that, create `evolve_n32.par` using below parameters

```
#
# Test of Gravity
#
# initial data
  initial_data_prefix = "wd_modified_n32"
  eos_type = "polytropic"
  poly_gamma = 1.66667   # polytropic index
  rho_initial = 5.2e+6
  pressure_initial = 1.56085e+23
  sphere_radius = 4.79283e+08
  domain_type = 1          # 0:box, 1:sphere

# gravity related parameters:
  enable_fmm = yes
  fmm_macangle = 0.3
  #fmm_max_cell_mass = 0.1

# evolution parameters:
  sph_kernel = "Wendland C6"
  sph_eta = 1.6

  initial_dt = 1.e-12
  timestep_cfl_factor = 0.5

  final_iteration = 10000
  out_screen_every = 1
  out_scalar_every = 1
  out_h5data_every = 100
  out_diagnostic_every = 10
  output_h5data_prefix = "wd_evolution_n32"

  thermokinetic_formulation = yes
  adaptive_timestep = yes
  sph_variable_h = yes
  evolve_internal_energy = yes
  relaxation_repulsion_gamma = 0.0
  gravitational_constant = 6.67383e-8
```

or you can get it from here. Then, follow the below commands

```
mpirun -np 1 ./newtonian_3d evolve_n32.par
```

Note that you can find `newtonian_3d` in your `${PATH_TO_FLECSPH}/flecsph/build/app/drivers` or `${CMAKE_PRE_PATH}/bin` based on your compilation procedure. After successful generation, you will have `wd_evolution_n32.h5part` file.

3.2.5 Analyze the result

The purpose of this test is checking conservation property for coupled system of hydrodynamics and gravity. For gravity computation, we use fast multipole method (FMM). We compare conservation of energy, linear momentum, and angular momentum, computed using the FMM approximation, and using exact N-body computation.

You can obtain both FMM and exact N-body results by changing `fmm_macangle` parameter in `evolve_n32.par`. If you set `fmm_macangle=0.0` this will provide exact N-body result. If you set `fmm_macangle` $>$ `0.0`, you will use FMM approximation to compute gravitational force.
After successful evolution simulation, you will get `scalar_reductions.dat` file that contains information of energy, linear and angular momentum during iterations. You can compare FMM results with exact N-body results by varying `fmm_macangle`.
There are several different ways to plot the results. We encourage user to use whatever user's convenient way but please contact flecsph-support@lanl.gov if you would like to get plotting script and example result data from us.

3.3 Binary White Dwarf Merger

3.3.1 Problem description

The merging of compact objects, such as neutron stars (NSs) and white dwarfs (WDs), is an interesting phenomenon for study in astrophysics. Here, we present a binary and example of a binary white dwarf (BWD) simulation using FleCSPH.

3.3.2 Generate initial data

To set up the system, we begin by initializing two individual stars. For this example, we consider 0.45 and 0.75 solar mass WDs. Provide radial profiles for the WDs with the following normalized columns: r/Rtot, rho*Rtotˆ3/Mtot, m/Mtot, and (drho/dr)*Rtotˆ4/Mtot. Create `input_star1.par` using below parameters:

```
#
# White Dwarf
# Initialization
# Use ./sedov_3d_generator <this_file>.par
#
  initial_data_prefix = "wd_0.45_initial"
  # geometry:
  domain_type        = 1                    # 0:box, 1:sphere
  sphere_radius      = 9.97195636003897e+08
  density_profile    = "from file"
  input_density_file = "WD_M0.45_prof.dat"
  # icosahedra lattice with small perturbations
  lattice_nx = 59                          # particle lattice dimension
  lattice_type = 4                         # 0:rectangular, 1:hcp, 2:fcc, 3:icosahedral
  lattice_perturbation_amplitude = 0.10     # in units of sm. length
  # equation of state type and parameters
  eos_type = "white dwarf"
  # density and pressure for relaxation stage
  rho_initial = 1.55526386469527e+06
  pressure_initial = 5.07690366495309e+22
  # since we only need spherical distribution of particles,
  # set Sedov energy to zero
  sedov_blast_energy = 0.0
  # use a good kernel
  sph_kernel = "Wendland C6"
  sph_eta = 1.6
```

Using this, we can generate initial data via below command

```
mpirun -np 1 ./sedov_3d_generator input_star1.par
```

Repeat for the second star configuration.

3.3.3 Relax with WVT driver

Run the WVT relaxation. Select either Diehl or Arth method in the input file. The number of iterations is currently set to 4000. Create `input_star1.par` using below parameters:

```
#
# White Dwarf Relaxation
# via WVT
# Use ./wvt_3d <this file>
#
# initial data
  initial_data_prefix = "wd_0.45_initial"
  initial_iteration   = 0
  domain_type         = 1                 # 0:box, 1:sphere
  sphere_radius       = 9.97195636003897e+08
  rho_initial         = 1.55526386469527e+06
  density_profile     = "from file"
  input_density_file  = "WD_M0.45_prof.dat"
# evolution
  final_iteration     = 4000
  adaptive_timestep   = false
# output
  out_screen_every = 1
  out_scalar_every = 10
  out_h5data_every = 20
  output_h5data_prefix = "wvt_wd_relaxed_m0.45"
# eos
  eos_type = "no eos"
# sph
  sph_kernel     = "Wendland C6"
  sph_eta        = 1.6
  sph_variable_h = true
# wvt
  wvt_method   = "diehl"                 # "arth" or "diehl"
  wvt_boundary = "reflective"            # "reflective" or "frozen"
  wvt_mu       = 1.e-3                   # small fraction of smoothing length
  wvt_ngb      = 60                      # number of desired neighbors
  wvt_convergence_check = true
  wvt_convergence_point = 0.1
  wvt_h_ngb = true
  wvt_cool_down = 0
  wvt_radius = 9.97195636003897e+08.      # if problems occur at the edge, reduce to less than total star
```

Using this, we can generate initial data with the following command

```
mpirun -np <#> ./wvt_3d relaxation_wd_wvt.par
```

You can check the evolution of the particles with any data analysis and visualization application, e.g., ParaView. However, beware that the density that you will see is not accurate. It is only provided for rough cross-check. For an accurate density.
(OPTIONAL) extract last iteration from wvt_wd_relaxed_m0.45.h5part, with the following requirements: extract_h5part_iteration.py (in flecsph/tools) and wvt_wd_relaxed_m0.45.h5part by running

```
python extract_h5part_iteration.py -f wvt_star_relaxed.h5part -l
```

Finally, check that the star is stable under it's own gravity by running

```
mpirun -np <#> ./newtonian_3d evolution_wd.par
```

with the following parameter file, evolution.par:

```
#
# White Dwarf
# Evolution with FMM
# run by ./newtonian_3d <this_file>
#
# initial data
  initial_data_prefix = "wd_relaxed_m0.45_<########>"
  initial_iteration = 0
# equation of state
  eos_type = "white dwarf"
# sph kernel
  sph_kernel = "Wendland C6"
  sph_eta = 1.2
  sph_variable_h = yes
# evolution parameters
  final_iteration = 200
  initial_dt = 2.e-14
  timestep_cfl_factor = 0.5
  adaptive_timestep = yes
  thermokinetic_formulation = FALSE
# output
  out_screen_every = 10
  out_scalar_every = 10
  out_h5data_every = 20
  output_h5data_prefix = "wd_relaxed_m0.45_ev"
# gravity related parameters:
  enable_fmm = yes
  fmm_macangle = 0.8
  gravitational_constant = 6.67408e-8
```

If stable, you can optionally extract the last

3.3.4 Run with external Roche potential

In order to accurately reflect orbital effects on the star's configuration, run the single star under gravity with the following parameter file, eforce_evolution.par:

```
#
# White Dwarf
# Evolution with FMM
# run by ./newtonian_3d <this_file>
#
# initial data
  initial_data_prefix = "wd_relaxed_m0.45_ev"
  initial_iteration = 0
# equation of state
  eos_type = "white dwarf"
# sph kernel
  sph_kernel = "Wendland C6"
  sph_eta = 1.2
  sph_variable_h = yes
```

```
# evolution parameters
  final_iteration  = 10000
  initial_dt = 2.e-14
  timestep_cfl_factor = 0.75
  adaptive_timestep = yes
  thermokinetic_formulation = FALSE
# output
  out_screen_every = 20
  out_scalar_every = 20
  out_h5data_every = 100
  output_h5data_prefix = "wd_relaxed_m0.45_roche"
# gravity related parameters:
  enable_fmm = yes
  fmm_macangle = 0.8
  gravitational_constant = 6.67408e-8
# eforce parameters:
  external_force_type = "orbit"
  mass_neutron_star   = 1.4913525e33  # companion star
  mass_white_dwarf    = 8.9479575e32  # primary star
  orbital_separation  = 3.064070749e9 # choose orbital separation
```

Run this with the following command:

```
mpirun -np <#> ./newtonian_3d eforce_evolution.par
```

Once you confirm that the star has relaxed into its Roche potential, you can extract the final iteration as before.

3.3.5 Create binary file

Run the following python script to place these stars into a binary. Requirements: make_binary_system.py (in flecsph/tools), star_1.h5part, star_2.h5part.

```
python make_binary_system.py -f <star1.h5part> <star1.h5part> -a <orbital separation in cm>
```

`make_binary_system.py` can be found in `tools` directory. After successful generation, you will have `binary.h5aprt`