User Guide of Phantom-GRAPE

Kohji Yoshikawa
Center for Computational Sciences
University of Tsukuba

Ataru Tanikawa
Department of Earth Science and Astronomy
University of Tokyo

`https://bitbucket.org/kohji/phantom-grape`

1 What is Phantom-GRAPE?

"Phantom-GRAPE" is a numerical software library to efficiently compute the Newtonian forces and potentials between particles in N-body simulations with the aid of SIMD instruction sets, Advanced Vector eXtenstion (AVX) and AVX-512, available on recent x86 processors. It includes two versions, one of which computes Newtonian accelerations and potentials in a single precision for collisionless N-body simulations, and the other computes the Newtonian acceleration and its time derivative 'jerk' in a double precision for collisional self-gravitating systems. The detailed structure of the library is described in Tanikawa et al. (2012, 2013).

Since 1990s, a number of GRAPE (GRAvity PipE) systems, dedicated hardware accelerators for gravitational N-body simulations, have been developed and utilized in astrophysical self-gravitating systems (Sugimoto et al., 1990; Makino et al., 2003; Fukushige et al., 2005). Recently, GPUs are utilized to accelerate N-body simulations (Hamada and Iitaka, 2007; Gaburov et al., 2009; Bédorf et al., 2012). Our approach is to accelerate N-body simulations with SIMD instructions and has an advantage over these hardware-aid acceleration in several points: it is free from data trasnfer overhead between a processor and an accelerator, and does not require any additional hardware. This approach is first applied to N-body simulations by Nitadori et al. (2006). Phantom-GRAPE is named after GRAPE systems since it is designed to be a software replacement of GRAPE hardwares.

Note that Phantom-GRAPE provides only the capability to compute the particle-particle interaction quickly, not the entire capability to perform N-body simulations. So, users can utilize Phantom-GRAPE in combination with existing N-body codes based on Tree, TreePM, PPPM methods by replacing the code for particle-particle interaction with the application programming interfaces (APIs) of Phantom-GRAPE.

In this document, we describe how to use Phantom-GRAPE effectively in users' N-body simulations in detail, including the compilation of libraries and its APIs.

2 Requirements

Phantom-GRAPE is implemented with the AVX/AVX2 instruction set and optionally the FMA3 instruction set, and thus requires processors that support these instruction sets. Since 2018, it also supports the AVX-512 instruction set. As of January 2018, most of commodity-based and server processors by Intel and AMD support the AVX2 and the FMA3 instruction sets. Phantom-GRAPE is written in C, C++, and inline assembly languages and requires GCC 4.X.X or higher to compile and build the library. For the implementation with the AVX-512 instruction set, GCC 6.X.X or higher is required.

3 License

Phantom-GRAPE is distributed under the MIT license. You may freely distribute and copy the software, and also modify it as you wish, and distribute these modified versions as long as you leave the copyright notices, and the no-warranty notice intact.

4 What Phantom-GRAPE computes

Phantom-GRAPE consists of two implementations, one of which is for collisionless self-gravitating systems, and the other is for collisional systems. In this document, the former is referred to as "PG5", while the latter is called "PG6", since their APIs are compatible with those of GRAPE-5 and GRAPE-6, respectively.

4.1 Phantom-GRAPE/PG5

PG5 computes Newtonian accelerations among a set of particles as

$$\boldsymbol{a}_i = \sum_{j=1}^{N_j} \frac{m_j(\boldsymbol{x}_j - \boldsymbol{x}_i)}{(|\boldsymbol{x}_j - \boldsymbol{x}_i|^2 + \epsilon_i^2)^{3/2}}, \tag{1}$$

where $\boldsymbol{x}_i$, ϵ_i, and m_i are the position, softening length and mass of i-th particle. Here, particles with subscript "j" exert forces on those with subscript "i". In the rest of this document, the former are referred to as "j-particles", and the latter as "i-particles". We also have an option to compute naturally symmetrized softening (Saitoh and Makino, 2012) given by

$$\boldsymbol{a}_i = \sum_{j=1}^{N_j} \frac{m_j(\boldsymbol{x}_j - \boldsymbol{x}_i)}{(|\boldsymbol{x}_j - \boldsymbol{x}_i|^2 + \epsilon_i^2 + \epsilon_j^2)^{3/2}}. \tag{2}$$

PG5 also has a capability to compute the inter-particle force with an arbitrary shape with a given cut off radius r_{cut} as

$$\boldsymbol{a}_i = \sum_{j=1}^{N_j} f(|\boldsymbol{x}_j - \boldsymbol{x}_i|) \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{|\boldsymbol{x}_j - \boldsymbol{x}_i|}, \tag{3}$$

where $f(|\boldsymbol{x}_j - \boldsymbol{x}_i|)$ is a user-specified force shape as a function of inter-particle separation, and satisfies $f(r) = 0$ if $r > r_{\text{cut}}$. This capability is useful for the TreePM method usually adopted in cosmological N-body simulations, in which the long-range force is computed with the particle-mesh (PM) scheme, and the short-range force is computed in the form of (3). PG5 computes accelerations with a user-defined force shape by interpolating a pre-calculated look-up table.

PG5 accelerates these calculations by computing (1), (2) and (3) for multiple i-particles in parallel. Spatial coordinates of particle positions are input in double-precision, but they are converted to single-precision inside PG5. The number of interactions computed in parallel is eight and 16 with PG5 library implemented with the AVX and AVX-512 instruction sets, respectively.

4.2 Phantom-GRAPE/PG6

PG6 is a library for collisional self-gravitating systems, and computes accelerations and jerks of particles to integrate the particles' orbit with the fouth-order Hermite scheme (Makino and Aarseth, 1992). The acceleration and jerk of the ith particle is given by

$$\boldsymbol{a}_i = \sum_{j=1}^{N_j} \frac{m_j \boldsymbol{r}_{ji}}{(|\boldsymbol{r}_{ji}|^2 + \epsilon^2)^{3/2}}, \tag{4}$$

and

$$\boldsymbol{j}_i = \sum_{j=1}^{N_j} \frac{m_j}{(|\boldsymbol{r}_{ji}|^2 + \epsilon^2)^{3/2}} \left[\boldsymbol{v}_{ji} - \frac{3\boldsymbol{r}_{ji} \cdot \boldsymbol{v}_{ji}}{|\boldsymbol{r}_{ji}|^2 + \epsilon^2} \boldsymbol{r}_{ji} \right], \tag{5}$$

where $\boldsymbol{r}_{ji} = \boldsymbol{x}_j - \boldsymbol{x}_i$ and $\boldsymbol{v}_{ji} = \boldsymbol{v}_j - \boldsymbol{v}_i$. In PG6, coordinates and velocities of particles are input in double and single precision, respectively. In calculations of accelerations and jerks, only the relative coordinate vector $\boldsymbol{r}_{ji}$ is computed in double precision and other operations are done in single precision. Thus, PG6 implemented with the AVX (AVX-512) instruction sets can compute 8 (16) interactions in parallel.

5 Compiling the Libraries

Following is the directory tree of the Phantom-GRAPE package.

```
phantom_grape ........................................ root directory
 |__PG5 .............................................. PG5 package
 |   |__newton .............................. PG5 for the Newtonian force shape
 |   |   |__setting.mk
 |   |   |__libpg5 ................ PG5 library with the AVX/AVX2 instruction sets
 |   |   |__libpg5_avx512 ............ PG5 library with the AVX-512 instruction set
 |   |   |__direct ......................................... sample code
 |   |__table ............................... PG5 for an arbitrary force shape
 |       |__setting.mk
 |       |__libpg5................PG5 library with the AVX/AVX2 instructioin sets
 |       |__libpg5_avx512............PG5 library with the AVX-512 instructioin set
 |       |__force_prof ..................................... sample code
 |__PG6 .............................................. PG6 package
     |__setting.mk
     |__libpg6 .................... PG6 library with the AVX/AVX2 instruction sets
     |__libpg6_avx512 ................ PG6 library with the AVX-512 instruction set
     |__banana ............................................. sample code
```

The directories PG5 and PG6 contain source code for PG5 and PG6 packages, respectively. Inside the PG5 directory, two directories newton and table contain the source codes of PG5 for the Newtonian force and the one for an arbitrary force shape, respectively. In each of PG5/newton, PG5/table, and PG6 directories, we have directories named libpgX and libpgX_avx512. The former contains the code implemented with the AVX and AVX2 instruction sets using inline assembly, while the latter contains the ones which supports all of the AVX, AVX2 and AVX-512 instruction sets but is implemented using Intel intrinsic functions rather than inline assembly. Therefore, users should use the source codes in libpg5_avx512 and libpg6_avx512 if users' processors does support the AVX-512 instruction set, and use the ones in libpg5 and libpg6 otherwise. Although the source codes in libpg5_avx512 and libpg6_avx512 can be properly compiled and work on processors with the AVX/AVX2 instruction sets but without the AVX-512 instruction set, the ones in libpg5 and libpg6 yield better performance because of its low-level implementation with inline assembly.

5.1 PG5 for the Newtonian force

First, edit the setting.mk file in the directory newton to configure the setting of PG5.

```
enable_openmp=yes
enable_symmetric=no

#instruction_set=avx
instruction_set=avx2
#instruction_set=avx512
```

Listing 1: PG5/newton/setting.mk

The switch enable_openmp should be set to yes to perform force calculations in a multi-thread manner using the OpenMP programming interface. Otherwise, as is in the case that users' program is multi-threaded and the Phantom-GRAPE APIs are called in OpenMP parallel regions, it should be turned off by setting as follows.

```
enable_openmp=no
```

Listing 2: Disabling the OpenMP multi-threading

If the switch `enable_symmetric` is turned on, PG5 computes the inter-particle forces with naturally symmetrized softening length given by (2). The `instructin_set` switch specifies the SIMD instruction set to be adopted in computing inter-particle forces. There are three optioins `avx`, `avx2` and `avx512`, each of which indicates the adoption of the AVX, AVX-2, and AVX-512 instruction sets, respectively. Then, issuing a `make` command in `newton/libpg5` or `newton/libpg5_avx512` directory will create the Phantom-GRAPE library `libpg5.a` in the same directory.

```
% cd newton/libpg5
% make libpg5.a
```

Listing 3: Compiling the PG5 library in the `libpg5` directory

To verify the performance of Phantom-GRAPE library compiled in the above procedure, we have a package of a simple N-body code in the `direct` directory, in which the force calculation is performed with the brute force direct scheme. With the following procedure, users can build the executable binary.

```
% cd ../direct
% make direct
```

Listing 4: Compiling a sample N-body code

The performance of Phantom-GRAPE library is measured by running the `direct` program with the following command.

```
% ./direct init/pl4k output.dat
```

Listing 5: Running the sample N-body code `direct`.

This command performs an N-body simulation starting from an initial condition of the Plummer model represented with 4K = 4096 particles, and gives the following output.

```
gravity 5.775973 cycle per loop
gravity 5.580705 cycle per loop
gravity 7.073731 cycle per loop
gravity 5.813126 cycle per loop
step: 600 time: 6.000000e+00
e: -2.694354e-01 de: 6.089192e-06
ke: 2.711584e-01 pe: -5.405939e-01
ke/pe: -5.015936e-01

2.060791e+09 interaction per sec, 78.310042 Gflops
```

Listing 6: A partial output of the `direct code`

This indicates that a single loop of the force calculation takes 5 to 7 CPU cycles and Phantom-GRAPE computes 2×10^9 inter-particle interactions per second or equivalently 78.3 Gflops assuming that a single inter-particle interaction consumes 38 floating-point operations. The `init` directory contains several files of initial conditions with various numbers of particles, and users can check the performance for them.

5.2 PG5 for an arbitrary force shape

The directory `table` contains the source code of the PG5 library for an arbitrary force shape, which computes inter-particle force given by (3). Before compiling the PG5 library, users should edit a `setting.mk` in the `table` directory, in which two optional switches are found as follows.

```
enable_openmp=yes

#instruction_set = avx
instruction_set = avx2
#instruction_set = avx512
```

Listing 7: `PG5/table/setting.mk`

Their meanings are the same as the PG5 for the Newtonian force shape. The shape of inter-particle force can be specified through a function `pg5_gen_force_table` defined in `pg5_table.c`, which creates the look-up table for the force calculation. By default, the PG5 library returns the force shape given by

$$f(r) = R(r, \epsilon_{\rm PP}) - R(r, \epsilon_{\rm PM}), \tag{6}$$

where $R(r, \epsilon)$ is the S2 softening function used in TreePM and PPPM methods, and $\epsilon_{\rm PP}$ and $\epsilon_{\rm PM}$ are the softening length for particle-particle interaction and the one for the particle-mesh scheme, respectively. The values of $\epsilon_{\rm PP}$ and $\epsilon_{\rm PM}$ are defined in the `pppm.h` file. After editting the `Makefile` and configuring the force shape, issuing a `make` command in the `libpg5` or `libpg5_avx512` directory creates the PG5 library `libpg5_table.a` in the same directory.

```
% cd libpg5
% make libpg5_table.a
```

Listing 8: Building the PG5 library for an arbitrary force shape in the `libpg5` directory

For the verification of computed force shape and computational performance, a sample code is prepared in the `force_prof` directory.

```
% cd ../force_prof
% make force_prof
```

Listing 9: Compiling and building a sample code `force_prof`.

```
% ./force_prof 32768 0.1 > force_shape.dat
```

Listing 10: Running `force_prof`.

Running the `force_prof` as shown in Listing 10 performs force calculation for 32768 particles randomly distributed with a maximum inter-particle separation of 0.1 in the simulation unit. Following is an example of the output which indicates the number of computed interactions per second is 8.5×10^8 and is equivalent to computational performance of 32.3 Gflops when we count 38 floating point operations per interaction.

```
 8.508592e+08 interactions / sec
 3.233265e+01 Gflops
```

Listing 11: Output of `force_prof`.

The computed force shape is output to `force_shape.dat` as a function of inter-particle separation.

5.3 PG6

The directory `PG6/libpg6` and `PG6/libpg6_avx512`contains the source code for PG6 library. Configuration of the library can be set by editting the `setting.mk` file.

```
enable_openmp=yes

#instruction_set=avx
instruction_set=avx2
#instruction_set=avx512
```

Listing 12: `setting.mk` in PG6 directory

The meanings of these switches are identical to those in PG5 library. Then, issuing the `make` command in the `libpg6` and `libpg6_avx512` directories creates the PG6 library.

```
% cd libpg6
% make libpg6.a
```

Listing 13: Building the PG6 library

Users can verify the performance of the PG6 library using the sample code `banana`. The `banana` code can be compiled in the `banana` directory as following

```
% cd ../banana
% make banana01
```

Listing 14: Building the sample program of the PG6 library.

This code simulates the time evolution of a self-gravitating system with the four-th order Hermite scheme (Makino and Aarseth, 1992) using the PG6 library. The setup of the simulation is configured by `input.list` file, A sample of which is shown below.

```
0 1.0 100.0
0.0
0.2 0.01 8.0
init/pl016k.init
tmp
```

Listing 15: `input.list`

The second line is an inverse of a softening length. If it is zero as shown in this sample, it means that the softening length itself is set to zero. The fourth line indicates a file for an initial condition, a plummer model with 16K particles in this sample. Running `banana01` produces following output for each timestep.

```
Time:     0.125
Time:     0.125
---Energy-------------------------------------------------------------------
tot: -0.25000002 w: -0.49956180 k: 0.24956179 de: -2.646e-08 de/|e|: -1.059e-07
---Time---------------------------------------------------------------------
tcalc: 2.38e+01 1.84e+01 Gflops niave: 2.32e+02 4.470170e+05
```

Listing 16: Partial output of `banana01`

This output indicates that Phantom-GRAPE/G6 library performs force calculation with the performance of 18.4 Gflops.

6 Application Programming Interface

Here, all the APIs defined in the PG5 and PG6 libraries are presented. Although most of them are compatible with those defined in the GRAPE-5 and GRAPE-6 runtime libraries, there are several additional APIs for new functionalities, and some of the APIs in the original G5 and G6 runtime libraries are deleted since they are specific to hardware configurations of the G5 and G6 boards.

6.1 PG5 for the Newtonian force

First, APIs for single-thread codes are presented. Users can compute accelerations in the form of (1) using the following APIs

- `void g5_open();`

  This function performs an initialization of the PG5 library, and should be called before other APIs are called in users' code.

- `void g5_close();`

  In the PG5 library for the Newtonian force, this function does nothing. It is defined just for the compatibility with the PG5 for an arbitrary force shape.

- `int g5_get_number_of_pipelines();`

  This function returns the number of particles whose accelerations are computed simultaneously. In the case of the AVX and AVX2 instruction sets, it returns four.

- `int g5_get_jmemsize();`

  This function returns the number of j-particles that can be stored in the buffer of the PG5 library.

- `void g5_set_xmj(int adr, int nj, double (*xj)[3], double *mj);`

  This function sets the data of j-particles to a buffer in the PG5 library. `xj` and `mj` are pointers to arrays of positions and masses of j-particles, and `nj` and `adr` are the number of j-particles to be set and the starting address of the buffer region where the particles' data are stored, respectively. This function can be called successively to incrementally set the j-particles.

- `void g5_set_n(int nj);`

  This function sets the number of j-particles to be involved in the calculation of inter-particle force. The first `n` particles in the buffer of j-particles are taken into account.

- `void g5_set_xi(int ni, double (*xi)[3]);`

  This function set the position data of i-particles to the buffer of the PG5 library. `ni` is the number of i-particles to be set. Note that `ni` cannot be larger than the number of pipelines.

- `void g5_set_xei(int ni, double (*xi)[3], double *eps2);`

  This function set the position and square of the softening legth of i-particles to the buffer of the PG5 library. `ni` is the number of i-particles to be set. Note that `ni` cannot be larger than the number of pipelines.

- `void g5_set_eps(int ni, double *epsi);`

  This function set the softening length of i-particles to the buffer of the PG5 library. `ni` is the number of i-particles to be set. Note that `ni` cannot be larger than the number of pipelines.

- `void g5_set_eps_to_all(double eps);`

  This function set softening length of all the i-particles to `eps`.

- `void g5_run();`

  This function perform the calculation of inter-particle force.

- `void g5_get_force(int ni, double (*a)[3], double *p);`

  This function retrieves the computed accelerations and potentials of i-particles. `ni` is the number of i-particles in consideration. Note that `ni` cannot be larger than the number of pipelines.

- `void g5_calculate_force_on_x(double (*x)[3], double (*a)[3], double *p, int ni);`

  This function is equivalent with the sequential combination of `g5_set_xi(ni, x)`, `g5_run()` and `g5_get_force(ni, a, p)`. Unlike `g5_set_xi()` and `g5_get_force()`, the number of i-particles `ni` can be larger than the number of pipelines and be folded into loops internally.

- `void g5_calculate_force_on_xe(double (*x)[3], double *eps2, double (*a)[3], double *p, int ni);`

  This function is equivalent with the sequential combination of `g5_set_xei(ni, xi, eps2)`, `g5_run()` and `g5_get_force(ni, a, p)`. Unlike `g5_set_xei()` and `g5_get_force()`, the number of i-particles `ni` can be larger than the number of pipelines and be folded into loops internally.

If users want to perform the inter-particle force calculation in a multi-thread manner, the following APIs with a suffix "MC" are useful, in which `devid` is the indices of threads. With these APIs, the PG5 perform force calculations in a thread-safe manner.

- `void g5_set_xmjMC(int devid, int adr, int nj, double (*xj)[3], double *mj);`
- `void g5_set_nMC(int devid, int nj);`
- `void g5_set_xiMC(int devid, int ni, double (*xi)[3]);`
- `void g5_set_xeiMC(int devid, int ni, double (*xi)[3], double *eps2);`
- `void g5_runMC(int devid);`
- `void g5_get_forceMC(int devid, int ni, double (*a)[3], double *p);`

In computing accelerations with the symmetrized softening in the form of (2), the following APIs are modified so that they accept input of j-particles' softening length.

- `void g5_set_xmj(int adr, int nj, double (*xj)[3], double *mj, double *epsj);`
- `void g5_set_xmjMC(int devid, int adr, int nj, double (*xj)[3], double *mj, double *epsj);`

6.2 PG5 for an arbitrary force shape

The APIs of PG5 for an arbitrary force shape are almost the same with those for the Newtonian force shape, and contain several additional APIs for setting up the force shape to be computed. Note that PG5 for an arbitrary force shape does not support the symmetrized softening and that the softening length is supposed to be the same among all the particles. By default, the force shape is set to

$$f(r) = R(r, \epsilon_{\mathrm{PP}}) - R(r, \epsilon_{\mathrm{PM}}), \tag{7}$$

where $R(r, \epsilon)$ is the $S2$-profile

$$R(r,\epsilon) = \begin{cases} (224R - 224R^3 + 70R^4 + 48R^5 - 21R^6)/140 & 0 \le R \le 1 \\ (12/R^2 - 224 + 869R - 840R^2 + 224R^3 + 70R^4 - 48R^5 + 7R^6)/140 & 1 \le R \le 2 \\ 1/R^2 & 2 \le R \end{cases}, \quad (8)$$

where $R = r/\epsilon$, and is frequently adopted in the PPPM and TreePM method in gravitational N-body simulations (Hockney and Eastwood, 1988).

- `void g5_open();`

  This function performs an initialization of the PG5 library, and should be called before other APIs are called in users' code. By default, this API sets the look-up table for the force shape given by equation (7). The parameters $\epsilon_{\rm PP}$ and $\epsilon_{\rm PM}$ are set with the definitions of preprocessor macros, `SFT_FOR_PP` and `SFT_FOR_PM` in the `pppm.h` file, respectively.

- `void g5_close();`

  In the PG5 library for the Newtonian force, this function does nothing. It is defined just for the compatibility with the PG5 for an arbitrary force shape.

- `int g5_get_number_of_pipelines();`

  This function returns the number of particles whose accelerations are computed simultaneously. In the case of the AVX and AVX2 instruction sets, it returns four.

- `int g5_get_jmemsize();`

  This function returns the number of j-particles that can be stored in the buffer of the PG5 library.

- `void g5_set_xmj(int adr, int nj, double (*xj)[3], double *mj);`

  This function sets the data of j-particles to a buffer in the PG5 library. `xj` and `mj` are pointers to arrays of positions and masses of j-particles, and `nj` and `adr` are the number of j-particles to be set and the starting address of the buffer region where the particles' data are stored, respectively. This function can be called successively to incrementally set the j-particles.

- `void g5_set_n(int nj);`

  This function sets the number of j-particles to be involved in the calculation of inter-particle force. The first `n` particles in the buffer of j-particles are taken into account.

- `void g5_set_xi(int ni, double (*xi)[3]);`

  This function set the position data of i-particles to the buffer of the PG5 library. `ni` is the number of i-particles to be set. Note that `ni` cannot be larger than the number of pipelines.

- `void g5_run();`

  This function perform the calculation of inter-particle force.

- `void g5_get_force(int ni, double (*a)[3], double *p);`

  This function retrieves the computed accelerations and potentials of i-particles. `ni` is the number of i-particles in consideration. Note that `ni` cannot be larger than the number of pipelines.

- `void g5_calculate_force_on_x(double (*x)[3], double (*a)[3], double *p, int ni);`

  This function is equivalent with the sequential combination of `g5_set_xi(ni, x)`, `g5_run()` and `g5_get_force(ni, a, p)`. Unlike `g5_set_xi()` and `g5_get_force()`, the number of i-particles `ni` can be larger than the number of pipelines and be folded into loops internally.

Users can use the following thread-safe APIs with a suffix "MC" in a multi-threaded program, where `devid` is the index of threads given by `omp_get_thread_num()` of the `OpenMP` API.

- `void g5_set_xmjMC(int devid, int adr, int nj, double (*xj)[3], double *mj);`
- `void g5_set_nMC(int devid, int nj);`
- `void g5_set_xiMC(int devid, int ni, double (*xi)[3]);`
- `void g5_set_xeiMC(int devid, int ni, double (*xi)[3], double *eps2);`
- `void g5_runMC(int devid);`
- `void g5_get_forceMC(int devid, int ni, double (*a)[3], double *p);`

6.3 PG6

To be written.

- `void g6_open(int gpid);`

  Initializes the Phantom-GRAPE/PG6 library. The argument `gpid` is a dummy and just for backward compatibility.

- `void g6_close(int gpid);`

  Terminates the capability of the Phantom-GRAPE/PG6 library. The argument `gpid` is a dummy and just for backward compatibility.

- `int g6_npipes();`

  This function returns the number of particles whose accelerations are computed simultaneously. It is 48 in the current version.

- `void g6_set_j_particle(int gpid, int padr, int pidx, double t, double dt, double mass, double *snp, double *jrk, double *acc, double *vel, double *pos);`

  This function sends j-particle data to the Phantom-GRAPE/PG6 library. `pos`, `vel`, `acc`, `jrk` and `snp` are the position, velocity, jerk and snap of the particle and must be arrays with the size of three.

- `void g6_set_ti(int gpid, double time);`

  This function specifies the current time `time` for the predictor of the Hermite time integration scheme.

- `void g6calc_firsthalf(int gpid, int nj, int ni, int *pidx, double (*pos)[3], double (*vel)[3], double (*acc)[3], double (*jrk)[3], double *pot, double eps2, double *h2);`

This function starts to compute the gravitational forces and jerks of the i-particles. The arguments other than `nj`, the number of j-particles, are just for the backward compatibility with the GRAPE-6 runtime library and redundant.

- `int g6calc_lasthalf(int gpid, int nj, int ni, int *pidx, double (*pos)[3], double (*vel)[3], double eps2, double *h2, double (*acc)[3], double (*jrk)[3], double *pot)`

  This function returns zero after finishing the calculation of gravitational accelerations, jerks and potentials. The arguments `pos`, `vel`, `h2`, `acc`, `jrk`, `pot` are the positions, velocities, the radii of the neighbor sphere, accelerations and gravitational potentials of the i-particles, respectively. `eps2` is the squared softening length. The computed results are stored in `acc`, `jrk` and `pot`.

- `int g6calc_lasthalf2(int gpid, int nj, int ni, int *pidx, double (*pos)[3], double (*vel)[3], double eps2, double *h2, double (*acc)[3], double (*jrk)[3], double *pot, int *nnb)`

  This function is identical to `g6calc_lasthalf` except that it returns `nnb`, the indices for the nearest neighbors.

- `int g6calc_lasthalf2p(int gpid, int nj, int ni, int *pidx, double (*pos)[3], double (*vel)[3], double eps2, double *h2, double (*acc)[3], double (*jrk)[3], double *pot, int *nnb, float *rnnb);`

  This function is identical to `g6calc_lasthalf` except that it returns `nnb`, the indices for the nearest neighbors.

- `int g6_read_neighbour_list(int gpid);`

  This function is prepared only for backward compatibility.

- `int g6_get_neighbour_list(int gpid, int ipipe, int maxlen, int *nblen, int *nbl);`

  This function returns the neighbour list of a particle calculated for particle `ipipe`. The argument `maxlen` gives the size of the array `nbl`. `nblen` is the length of the neighbour list and `nbl` is the actual list sorted by the indices.

References

Bédorf, J., Gaburov, E., and Portegies Zwart, S.: 2012, *Journal of Computational Physics,* **231**, 2825

Fukushige, T., Makino, J., and Kawai, A.: 2005, *PASJ,* **57**, 1009

Gaburov, E., Harfst, S., and Portegies Zwart, S.: 2009, *New Astronomy,* **14**, 630

Hamada, T. and Iitaka, T.: 2007, *ArXiv Astrophysics e-prints,*

Hockney, R. W. and Eastwood, J. W.: 1988, *Computer simulation using particles*, Bristol: Hilger, 1988

Makino, J. and Aarseth, S. J.: 1992, *PASJ,* **44**, 141

Makino, J., Fukushige, T., Koga, M., and Namura, K.: 2003, *PASJ,* **55**, 1163

Nitadori, K., Makino, J., and Hut, P.: 2006, *New Astronomy,* **12**, 169

Saitoh, T. R. and Makino, J.: 2012, *New Astronomy,* **17**, 76

Sugimoto, D., Chikada, Y., Makino, J., Ito, T., Ebisuzaki, T., and Umemura, M.: 1990, *Nature,* **345**, 33

Tanikawa, A., Yoshikawa, K., Nitadori, K., and Okamoto, T.: 2013, *New Astronomy,* **19**, 74

Tanikawa, A., Yoshikawa, K., Okamoto, T., and Nitadori, K.: 2012, *New Astronomy,* **17**, 82