# GRAPE-3A User's Guide

Junichiro Makino

Department of Information Science and Graphics,
College of Arts and Sciences,
University of Tokyo,
Komaba 3–8–1, Meguro-ku, Tokyo 153, Japan
Phone : 81–3–3465–3925
Fax : 81–3–3465–3925 (same as phone)
E-mail : makino@kyohou.c.u-tokyo.ac.jp

**Abstract**

In this document, I'll describe how to run the simple $N$-body programs you have installed. Also, I give the full description of the GRAPE-3A interface software package.

# Contents


**1 Notes** **3**

**2 How to run simple test programs** **3**
2.1 grape3a . . . 3
2.1.1 Limitations . . . 4
2.1.2 Timing . . . 5
2.2 dcode . . . 5
2.2.1 Limitations . . . 6

**3 Reference Manual for subroutines** **7**
3.1 DESCRIPTION . . . 9
3.2 EXAMPLES . . . 11
3.2.1 Example in C . . . 11
3.2.2 Example in Fortran . . . 11
3.3 NEIGHBOR LIST . . . 13
3.4 NEIGHBOR LIST ON MULTI-BOARD SYSTEMS . . . 13
3.5 TIMESHARING . . . 15
3.6 SELF-INTERACTION . . . 15
3.7 RANGES AND ACCURACY . . . 15
3.8 LINKING . . . 16
3.9 BUGS . . . 17

**4 Questions and Answers** **17**
4.1 More than 32768 particles. . . . 17
4.2 Saving time by not sending mass at each timestep. . . . 18
4.3 Is `g3init` a subroutine or a function? . . . 19
4.4 `g3nchips` gives 0. . . . 19
4.5 Numbering of particles (in fortran). . . . 19
4.6 How to assign different neighbor spheres to different particles. . . . 20
4.7 How to assign different softenings to different particles. . . . 20
4.8 Unit for softening and neighbor radius. . . . 21
4.9 Value of gravitational constant. . . . 21
4.10 Neighbor list seems incorrect. . . . 21
4.11 System crash on SPARCstation 10/LX/5/20. . . . 21

**5 Update History** **21**

# 1 Notes

In the following sections, I assume the directory you installed the grape3a software is /usr/grape3a. If it is different, replace /usr/grape3a below with the actual directory name where files are.

# 2 How to run simple test programs

In GRAPE-3A software package, two $N$-body time integration programs are included. In this section I describe how to use them.

## 2.1 grape3a

The program **grape3a** integrates an $N$-body systetm using time-centered leapfrog integrator with constant timestep. The executable is in the directory `/usr/grape3a/grape`. To run **grape3a**, type

`/usr/grape3a/grape/grape3a` *parameter_file* > *log_file*

The format of the parameter file is the following

`0` *dt tstop tout tsnapout eps* `1`
*output_snapshot_file_name*
*input_snapshot_file_name*

The first parameter in the first line must be zero. The second parameter gives the timestep for integration. The next one gives the time to terminate the calculation. The 4th parameter is the time interval to output log information. The 5th parameter is the time interval to produce the output snapshot. The 6th parameter is the softening parameter. Here, standard Plummer softening is used. The last parameter in the first line is currently unused. I recommend to supply the value 1 here.

The second line gives the name of the output snapshot file. To suppress the creation of the output snapshot, give "∼" as the name. If you compiled the **grape3a** program under NEMO environment, the output snapshot file is in the standard NEMO snapshot format and can be processed with any of NEMO programs. If you are not using NEMO, the output is done in binary, but in the same order as used in the input snapshot file which will be described below. One difference is that the output also includes the potential in the last part of each snapshot. All floating point values are in `float`, not in `double`. All output is done using `fwrite(3)` If you are familiar with C, it should be straightforward to write a function to read in this snapshot file. If not, I recommend to install NEMO and recompile `grape3a` with NEMO.

The third line gives the name of the input snapshot file. The format for the input snapshot file is the NEMO "stoa" format, which is the output format of NEMO "stoa" program used to convert NEMO snapshot file to ascii format. The format is

```
NOBJ                    number of particles [integer]
NDIM                    number of dimensions [integer] (MUST BE 3)
TIME                    time of snapshot [real]
MASS(i)                 particle masses, i = 1...NOBJ [real]
. . . . .
X(i) Y(i) Z(i)          particle positions, i = 1...NOBJ [real]
. . . . .
U(i) V(i) W(i)          particle velocities, i = 1...NOBJ [real]
. . . . .
```

The following is an example of the input parameter file:

```
0 0.03125 20 1 1  0.03125 8
~
pltest.ascii
```

The input snapshot file `pltest.ascii` is included in the driver software. The sample parameter file `demopl` is also included. To run the program, type

```
grape3a demopl > testlog
```

The first part of the output file `testlog` would look like

```
argc = 2, argv[1]=demopl
n=   0 dt= 0.0312 tstop=  20.000 dtout=  1.000 dtsnapout=  1.000
eps= 0.03125 pesample=8
output snap = ~
input snap = pltest.ascii
Enter diag:CPU sec.=        1.77
T=       0.000  E =    -0.260484159
DE= 0.00000000e+00  V.R. =   0.482206
CM :-1.34760e-09  5.09488e-10  6.69161e-10
CMV:-1.93097e-10 -1.21958e-10  5.62252e-11
AM :-2.11387e-03  1.27270e-02  1.54577e-02
Exit  diag:CPU sec.=        1.78
Enter diag:CPU sec.=        3.43
T=       1.000  E =    -0.260437131
DE=-1.80573581e-04  V.R. =   0.488161
CM :-2.14914e-09  4.97340e-09 -1.36060e-09
CMV:-3.85955e-09  3.22359e-09 -2.63549e-09
AM :-2.11731e-03  1.27269e-02  1.54570e-02
Exit  diag:CPU sec.=        3.45
Enter diag:CPU sec.=        5.10
```

The first line might not appear if you are using NEMO. I hope the output is self-descriptive. This is the result of run on SS-10 and SolFlower SFVME-116 adapter. The CPU time might be a bit longer if you are using a slower machine.

### 2.1.1 Limitations

In the default setting, **grape3a** can handle up to 65536 particles. If you want to use more particles, edit the file `grape/grape.c` and change all lines with "#define NMAX nnnnn", so that nnnnn will be the number you want to use. Then recompile the program by "make grape3a". This program can correctly handle the number of particles larger than the hardware of GRAPE-3A can handle (which is, for wrapping boards and PCB boards, 32768 and 131072).

The program **grape3a** uses a number of undocumented features of GRAPE-3A software library in order to obtain the maximum possible speed. The program **grape3a** will exist in the future version of the software. However, the routines used in the program may change/disappear.

### 2.1.2 Timing

The CPU time per timestep is given roughly by

$$T_{step} = aN + bN^2, \tag{1}$$

where $N$ is the number of particles, $a$ is the time constant which depends on the host computer, and $b$ is the time constant which depends on the speed of the GRAPE-3A hardware. In the case of SPARCstation-10, $a$ is roughly $3.8 \times 10^{-5}$ sec. For GRAPE-3A with 4 chips running on 20 MHz clock, $b = 1.25 \times 10^{-8}$sec. With these numbers, the above equation becomes

$$T_{step} = 0.38 \left(\frac{N}{10^4}\right) + 1.25 \left(\frac{N}{10^4}\right)^2. \tag{2}$$

On host machines slower than SPARCstation-10, the constant $a$ would be 2-3 times larger.

## 2.2 dcode

Program **dcode** is quite similar to **grape3a**. The major difference is that it is written in Fortran using only the documented features of GRAPE-3A software driver, while **grape3a** is written in C using undocumented features. Another important difference is that it can currently handle only the number of particles up to the hardware limit returned by the function `max_particle_number()` (`g3maxptcl`).

Program **dcode** is in the directory dcode. To run **dcode**, type

`/usr/grape3a/dcode/dcode` *parameter_file* > *log_file*

The format of the parameter file is the following

*Headline*
*input_snapshot_file_name*
*output_snapshot_file_name*
*tstop dtsnapout eta dt eps ianaly*
`.false.`

The first line is the headline printed out to the log file.

The second line gives the name of the input snapshot file. The format for the input snapshot file is the NEMO "stoa" format, which is the output format of NEMO "stoa" program used to convert NEMO snapshot file to ascii format. This is the same as that used for **grape3a**.

The first parameter in the 4th gives the time to terminate the calculation. The second parameter is the time interval to produce the output snapshot. The third parameter is currently unused. You have to give some number, however. The 4th parameter is the timestep. The 5th parameter is the softening parameter. Here, again, standard Plummer softening is used. The last parameter is the interval to output log information in terms of the number of timesteps.

The third line gives the name of the output snapshot file. To suppress the creation of the output snapshot, give one white space before the name of the snapshot file. The output is done in ascii, in the same format as that of the input file.

The following is an example of the input parameter file:

```
test
inbody
 outbody
15 0.25  0.1 0.015625 0.1 8 6
.false.
```

The input snapshot file `inbody` is included in the driver software. The sample parameter file `inparms` is also included. To run the program, type

```
dcode <inparms >outlog
```

The first part of the output file `outlog` would look like

```
 test
Snapshot file = NONE
 n=   30 tstop=  15.000 dtout= 0.250 eta= 0.1000 eps= 0.100
 dt(first)=0.01562 ianaly=    8

 T:     0.00 E: -0.1723971 DE: 0.000E+00 PE:  -0.3786671 Q: 0.54
 steps:  0
 CM :   -0.3972517E-15    0.2701832E-15   -0.2775558E-16
 CMV:   -0.3469447E-17    0.1014813E-15   -0.3642919E-16
 CPU min =     0.2666667E-02

 T:     0.00 E: -0.1723971 DE: 0.000E+00 PE:  -0.3786671 Q: 0.54
 steps:  0
 CM :   -0.3972517E-15    0.2701832E-15   -0.2775558E-16
 CMV:   -0.3469447E-17    0.1014813E-15   -0.3642919E-16
 CPU min =     0.2666667E-02

 T:     0.12 E: -0.1726256 DE: 0.132E-02 PE:  -0.3743062 Q: 0.54
 steps:  8
 CM :   -0.3937822E-15    0.2658464E-15   -0.4163336E-16
 CMV:    0.1734723E-17    0.8782038E-16   -0.2255141E-16
 CPU min =     0.4333333E-02

 T:     0.25 E: -0.1722749 DE:-0.709E-03 PE:  -0.3642766 Q: 0.53
 steps:  16
 CM :   -0.3694961E-15    0.3603888E-15   -0.6938894E-17
 CMV:    0.1734723E-17    0.8998878E-16   -0.1908196E-16
 CPU min =     0.5500000E-02

 T:     0.25 E: -0.1722749 DE:-0.709E-03 PE:  -0.3642766 Q: 0.53
 steps:  16
 CM :   -0.3694961E-15    0.3603888E-15   -0.6938894E-17
```

I presume the output is self-descriptive. This is the result of run on SS-10 and SolFlower SFVME-116 adapter. The CPU time might be a bit longer if you are using a slower machine.

### 2.2.1 Limitations

`Dcode` cannot handle the number of particles larger than the hardware limit of GRAPE-3A returned by the function `max_particle_number()` (`g3maxptcl`).

# 3 Reference Manual for subroutines

## SYNOPSIS — C interface

```
 int initialize_grape3()

 int free_grape3()

 void set_scales(minx,minm)
 REAL minx;
 REAL minm;

 void set_mass_correction_factor(c)
 REAL c;

 int number_of_available_chips()

 int number_of_chips_per_board()

int number_of_boards()

 void set_n(n)
 int n;

 void set_xj(j, x)
 int j;
 REAL x[3];

 void set_mj(j, m)
 int j;
 REAL m;

 void set_eps2(eps2)
 REAL eps2;

 void set_h2(h2,ichip)
 REAL h2;
 int ichip;

 void calculate_force_on_x(x,a,p,n)
    REAL x[][3];
    REAL a[][3];
    REAL  p[];
    int n;

 int max_particle_number();

 int max_neighbor_number();
```

```
int read_neighbor_list(iboard)
int iboard;

int get_neighbor_list(ichip,list)
int ichip;
int list[];
```

### SYNOPSIS — Fortran interface

```
function g3init
integer g3init

function g3free
integer g3free

subroutine g3setscales(minx,minm)
REAL minx
REAL minm

subroutine g3setmasscor(c)
REAL c

function g3nchips
integer g3nchips

function g3nchipsperboard
integer g3nchipsperboard

function g3nboards
integer g3nboards

subroutine g3setn(n)
integer n

subroutine g3setxj(j, x)
integer j
REAL x(3)

subroutine g3setmj(j, m)
integer j
REAL m

subroutine g3seteps2(eps2)
REAL eps2

subroutine g3seth2(h2,ichip)
```

```
REAL h2
integer ichip

subroutine g3frc(x,a,p,n)
   REAL x(3,*)
   REAL a(3,*)
   REAL  p(*)
   integer n

function g3maxptcl
integer g3maxptcl

function g3maxnnb
integer g3maxnnb

function g3readnbl(iboard)
integer g3readnbl,iboard

function g3getnbl(ichip,list)
integer g3getnbl
integer ichip
integer list
```

### 3.1 DESCRIPTION

`Grape3` is a set of simple interface routines to GRAPE-3/3A special-purpose N-body engine. The `grape3` routines provide all necessary functions to control the GRAPE-3/3A hardware.

`initialize_grape3` (g3init) initializes the GRAPE-3/3A hardware and interface package. On success, it returns 0. Any value other than 0 means error.

`free_grape3` (g3free) releases GRAPE3 and allows other users to acquire GRAPE-3/3A.

`set_scales(minx,minm)` (g3setscales) specifies the system of units used within GRAPE-3/3A system. GRAPE-3/3A uses fixed-point format for position and force. Therefore, the user must specify the unit length and unit mass, which determine the resolution. The bit length of position data is 19. Thus, if you specify `minx=1/1024`, the resolution of the position data within GRAPE-3/3A is 1/1024. The range of the position is $(-256, 256)$. If you specify smaller `minx`, you get better resolution but the range becomes smaller, too. For example, if you specify `minx=1/4096`, the range becomes $(-64, 64)$. In general, the maximum value for position is given by $\texttt{minx} \times 2^{18}$. The value of `minm` must not exceed the mass of any particle in the system (except for mass-less test particles). The maximum mass of one particle allowed is given by $\texttt{minm} * 64 * \min[(\texttt{rmin}/\texttt{minx}), (\texttt{rmin}/\texttt{minx})^2]$, where `rmin` is the distance to its nearest neighbor, including the softening parameter. This limit guarantees that both the force and the potential from this particle to any other particle does not overflow. If you are to calculate the force and potential on the particle itself, you might want to avoid the overflow in the self potential. In that case, the maximum allowed mass is $\texttt{minm} * 64 * (\texttt{eps}/\texttt{minx})$. (Take the smaller one of the above two values).

`set_mass_correction_factor` (g3setmascorr) auguments the function `set_scales`. By default, the

system of units in GRAPE-3/3A is chosen so that the force from the particle with mass `minm` would not underflow, even if it is far away (for example, distance is `minx`$*2^{19}$. For a direct summation calculation, this is fine. However, if one want to use TREECODE, it can cause trouble since (a) the force or potential from one node can be very large, in the unit of force/potential from one particle far away. (b) The total force and/or potential can be even larger. Thus, this function allows the underflow of the force from light particles, in order to prevent the overflow of total force/potential. If the total number of particles is larger than $10^5$, it is recommended to call `set_scales` with argument $c = 10^5/N$. Default value of this factor is unity.

`number_of_available_chips` (g3nchips) returns the total number of chips in the present system. `number_of_chips_per_board` (g3nchipsperboard) returns the number of chips per board. `number_of_boards` (g3nboards) returns the number of boards in the present system. Naturally, it is guaranteed that g3nchips=g3nchipsperboard*g3nboards.

`set_n` (g3setn) sets the number of particles `n` to GRAPE-3/3A. The force from `n` particles is accumulated on GRAPE-3/3A. `n` should be smaller than the value returned by `max_particle_number()`.

`set_xj` (g3setxj) writes the position data to the `j`-th location of particle memory of GRAPE-3. Index starts from 0.

`set_m` (g3setmj) writes the particle mass to the `j`-th location of particle memory of GRAPE-3.

`set_eps2` (g3seteps2) writes the squared softening parameter to GRAPE-3. Note that with this routine `eps2` is common on all chips. To set `eps2` on one chip, use `set_eps2_to_chip(eps2,ichip)`. In fortran, use `g3seteps2tochip(eps2,ichip)`.

`set_h2` (g3seth2) writes the squared neighbor sphere radius `h2` to chip `ichip`.

`calculate_force_on_x` (g3frc) passes the positions to calculates the gravity to GRAPE-3, starts the calculation, wait until GRAPE-3 finishes, and receives the calculated forces and potentials from GRAPE-3. It calculates the force on `n` particles in parallel, and `n` should not exceed `number_of_available_chips()` (g3nchips in fortran). The calculated potential includes self-interaction, which is not corrected within this routine. Note that this is different from GRAPE-1A, in which the potential is automatically corrected. The correction of the self-interaction will be described in more detail below.

`max_particle_number` (g3maxptcl) returns the maximum number of particles that can fit into the memory of GRAPE-3A (this function is available, at present, on GRAPE-3A only. The hardware limit of GRAPE-3 is 32748). Particle number passed to `set_n` must not exceed the value returned by this function. Particle index passed to finctions `set_xj` and `set_mj` must be smaller than this value.

`max_neighbor_number` (g3maxnnb) returns the maximum number of neighbors supported by the hardware. On present hardware (GRAPE-3/3A wrapping board), it is 256. Upon request, it is possible to increase the number of neighbors to up to 2048.

`read_neighbor_list` (g3readnbl) reads in the neighbor list of one board to the internal work space, and returns the total number of neighbors on that board. See section 3.3 for more details.

`get_neighbor_list` (g3getnbl) returns the number of neighbors and the neighbor list for chip `ichip` of the board specified in `read_neighbor_list`.

## 3.2 EXAMPLES

### 3.2.1 Example in C

The following code evaluates accelerations on N particles by direct summation.

```
main()
{
    float x[N][3];    /* position of particles */
    float acc[N][3];  /* calculated acceleration */
    float m[N];       /* mass of particles */
    float pot[N];       /* calculated potential */
    float eps,epsinv;        /* softening parameter */
    int i,j,nchips,nn;
    ......
    if(initialize_grape3()){
        fprintf(stderr,"GRAPE-3 initialize error\\n");
        exit(-1);
    }
    set_scales(1.0/1024,m[0]);
    nchips=number_of_available_chips();
    ......
    if(N > max_particle_number()){
       fprintf(stderr,"Number of particles %d too large\n", N);
       exit(-1);
    }
    set_n(N);
    for(j=0; j<N; j++){
       set_xj(j,x[j]);
       set_mj(j,m[j]);
    }
    set_eps2(eps*eps);
    for(i=0;i<N,i+=nchips){
        nn = nchips;
        if(i+nn > N) nn = N-i;
        calculate_force_on_x(x[i],acc[i],&pot[i],nn);
    }
    epsinv = 1.0/eps;
    for(i=0;i<N,i++){
        pot[i] += m[i]*epsinv;
    }
    ......
}
```

### 3.2.2 Example in Fortran

The following is the routine used in program `dcode`, in a slightly modified form. Here, positions and accelerations are declared as `pos(maxbod,ndim)`. Since in fortran the arrays are stored in the column-first order, the element that follows `pos(i,1)` is `pos(i+1,1)` and not `pos(i,2)`. With

this type of array, you cannot pass the arrays directly to GRAPE-3A interface routines, since it assumes the three elements of one vector is stored consecutively. If the arrays were declared as `pos(ndim,maxbod)`, the program would become somewhat simpler.

```
      subroutine accel_grape3a
      integer nbody, ndim, maxbod
      parameter(ndim=3,maxbod=32768)
      REAL pos(maxbod,ndim),acc(maxbod,ndim),
     $     mass(maxbod), pot(maxbod)
      REAL eps2
      common /arrays/pos,acc, mass,pot
      common /sysprr/eps
      common /syspri/nbody
      integer i, j, k,ii
      REAL xwork(3,4), awork(3,4),pwork(4)
      REAL xscale,epsinv
      epsinv = 1.0d0/sqrt(eps2)
      call g3init()
      xscale = 1.0d0/1024
      call g3setscales(xscale, mass(1)/32)
      call g3seteps2(eps2)
      call g3setn(nbody)
        do 20 i=1,nbody
           xwork(1,1)=pos(i,1)
           xwork(2,1)=pos(i,2)
           xwork(3,1)=pos(i,3)
           call g3setxj(i-1,xwork)
           call g3setmj(i-1,mass(i))
20         continue
      do 30 i=1,nbody,4
         ii = 4
         if(i+ii .gt. nbody+1) ii = nbody - i + 1
         do 40 j = 1,ii
            do 50 k = 1,3
               xwork(k,j)=pos(i+j-1,k)
50          continue
40       continue
         call g3frc(xwork,awork,pwork,ii)
         do 60 j = 1,ii
            do 70 k = 1,3
               acc(i+j-1,k)=awork(k,j)
70          continue
            pot(i+j-1) = pwork(j) + mass(i+j-1)*epsinv
60       continue
30    continue
      call g3free
      end
```

### 3.3 NEIGHBOR LIST

On GRAPE-3A board, the data stored are (a) the union of neighbor list of all particles for which forces are calculated in parallel and (b) for each particle in that list, list of particles for which it is a neighbor. From the hardware's point of view, it is explained as follows. Each of the 4 GRAPE chips of a GRAPE-3A board sets a flag when the distance between the particle in its register and the particle supplied from the memory is less than `h`. The circuit for the neighbor list stores the current value of the particle counter when any of these 4 flags is set. At the same time, it also stores the values of the flags themselves, so that the host computer can tell which particles are actually the neighbor of a particle.

The function `read_neighbor_list` simply reads in these information to the internal work space. It returns the length of this unified list. If the return value is equal to `max_neighbor_number()`, it means that the neighbor list has overflowen. Otherwise all neighbors are correctly recorded.

Note that `read_neighbor_list` performs an "destructive" read operation of the neighbor list hardware of one board. Thus, `read_neighbor_list` can be performed only once after one force calculation. Second call to `read_neighbor_list` would destroy the content of the internal work space.

The function `get_neighbor_list` selects the neighbors of the specified particle from the unified list.

Note that you have to set `h2` of unused chips to be 0. For example, if you want to construct the neighbor list of one particle, the sequence would look like:

```
set_h2(h2,0);
for(i=1;i<number_of_available_chips(); i++)set_h2(0.0,i);
calculate_force_on_x(x,a,p,1);
if(read_neighbor_list() == max_neighbor_number()){
    fprintf(stderr,"Too many neighbors ...\n");
    /* procedure to correct the list */
}
get_neighbor_list(0,nblist);
```

### 3.4 NEIGHBOR LIST ON MULTI-BOARD SYSTEMS

With the multi-board configulation, the function `read_neighbor_list` reads in the hardware neighbor list of one board specified in the argument. The procedure to obtain the neighbor list for `N` particles would look like the following **(WARNING: THE FOLLOWING CODE FRAGMENT IS UNTESTED!!!)**.

```
main()
{
    float x[N][3];    /* position of particles */
    float acc[N][3];  /* calculated acceleration */
    float m[N];       /* mass of particles */
    float pot[N];       /* calculated potential */
    float h2[N];        /* the squared radius of the neighbor sphere */
    float eps,epsinv;        /* softening parameter */
    int i,j,nchips,nn;
    int nblist[N][1023]; /* array to store the neighbor lists */
    int nnb[N];          /* neughbor number for each particle */
```

```
    ......
    if(initialize_grape3()){
        fprintf(stderr,"GRAPE-3 initialize error\\n");
        exit(-1);
    }
    set_scales(1.0/1024,m[0]);
    nchips=number_of_available_chips();
    ......
    if(N > max_particle_number()){
       fprintf(stderr,"Number of particles %d too large\n", N);
       exit(-1);
    }
    set_n(N);
    for(j=0; j<N; j++){
       set_xj(j,x[j]);
       set_mj(j,m[j]);
    }
    set_eps2(eps*eps);
    for(i=0;i<N,i+=nchips){
        int nb = number_of_available_chips()/number_of_chips_per_board();
        int nbc = number_of_chips_per_board();
        int ib;
        nn = nchips;
        if(i+nn > N) nn = N-i;
        for(k=0;k<nn;k++)set_h2(h2[i+k],k);
        for(k=nn;k<nchips;k++)set_h2(0.0,k);
        calculate_force_on_x(x[i],acc[i],&pot[i],nn);
        for(ib=0;ib<nb;ib++){
            if(read_neighbor_list(ib) >= max_neighbor_number()){
                fprintf(stderr,"Too many neighbors ...\n");
                /* procedure to correct the list */
            }
            for(k=0;k<nbc;k++){
                intger ip = i + ib*nbc +k; /* particle index */
                if(ip < N)get_neighbor_list(k,nblist[i]);
            }
        }
    }
    epsinv = 1.0/eps;
    for(i=0;i<N,i++){
        pot[i] += m[i]*epsinv;
    }

    ......
}
```

## 3.5 TIMESHARING

The Grape3 interface offers a rather primitive method for sharing the GRAPE-3A hardware by several programs running simultaneously. A program acquires GRAPE-3A by calling `initialize_grape3`. If someone is already using GRAPE-3, `initialize_grape3` prints the message "Someone is using GRAPE-3, waiting" and polls GRAPE-3A once per every second. Upon success, `initialize_grape3` returns. A program can release GRAPE-3A by calling `free_grape3`. Thus, a program occupies GRAPE-3A hardware between the calls to `initialize_grape3` and `free_grape3`.

In order to attain the sharing of GRAPE-3A between multiple users, therefore, programmers must write their program so that it releases GRAPE-3A as long as possible and as many times as possible.

## 3.6 SELF-INTERACTION

The potential energy returned includes the self-interaction. As shown in the code example above, you have to correct the returned potential on the host computer. Note that the self-potential calculated on GRAPE-3A can contain error of 1 percent on average. Therefore, if you use a very small softening compared to the size of the system, the error in the potential energy can be large and systematic. Note that the calculated force has no such problem since the force due to the self-interaction is exactly zero. In addition, this systematic error remains constant throughout the simulation, since this error is not affected by the positions of particles.

There are four ways to deal with the error in the potential energy. The first one is just to ignore it. As stated above, this error remains constant throughout the calculation. So, if you are interested in the change of the potential energy, it is calculated free from this systematic error. The second way is to use the value of softening which gives small round-off error, such as an integer power of 2 (0.5, 0.25, 0.125 and so on).

The third way is to explicitly avoid the self-intercation by setting the mass of particles to calculate the force to be zero. The author, however, does not recommend this approach since this approach gives you no improvement over the accuracy of the actual result. It gives you better energy conservation. And yet the orbits of particles are integrated in exactly the same way.

The fourth way is to correct the self-interaction more accurately, by actually calculating the self-interaction on GRAPE-3A hardware. You can do this by calculating the force and potential from the particle itself on GRAPE-3A.

There is one exception: If you assign zero to softening parameter, self-potential, which would be an infinity, will not included. So in that case you do not need the correction.

## 3.7 RANGES AND ACCURACY

The number format used in GRAPE-3A is complicated and difficult to understand. Those who do not want to get into the details of hardware are recommended not to use GRAPE-3A and wait for a machine with higher accuracy.

The position is expressed in 19-bit fixed point number in GRAPE-3A. In GRAPE-3A interface software, the input numbers are divided by `minx`, specified by `set_scales` and then rounded to nearest integer. If the absolute value of result is larger than 262143 ($= 2^{18} - 1$), its absolute value is forced to be 262143. Therefore, you have to make some compromise between the resolution and the range. The minimum distance `minx` must be sufficiently smaller than the softening parameter, since otherwise the force between nearby particles would be calculated inaccurately. In most cases, `minx` $= 0.1 *$ `eps` would be a reasonable choice.

This 19-bit fixed point format is used only for the calculation of the relative position between two particles. After the relative distance is calculated, it is converted to 12-bit logarithmic format. Its relative accuracy is about 1 percent. Therefore, regardless of the choice of `minx` and `minm`, the force between two particles always contains the relative error of 1-2 percent on average.

The calculation of the force from one particle is done using this logarithmic format. Here we do not have to worry about the overflow. However, before summing up the forces from different particles, it is converted to the fixed point format again. Thus, the range of mass needs to be scaled so that it does not cause overflow/underflow.

In the GRAPE-3A interface software, the unit of mass is chosen so that the force from the particle with `minm` will not become zero when the distance between two particles is the maximum value possible with the fixed-point format used to express position. So, if you use zero softening and the relative position is the minimum possible, the absolute value of the force is roughly $2^{40}$ for this minimum mass particle. If you use non-zero softening, the maximum value of the force is limited by the softening parameter. The force is converted to 48-bit integer before summation. So the force from any one particle should not exceed the maximum value expressible with this fixed point expression. This is where the limit for the maximum mass $\texttt{minm} * 64 * (\texttt{rmin}/\texttt{minx})^2$ comes from. One should also make sure the potential does not overflow. So, the mass should not exceed $\texttt{minm} * 64 * \texttt{rmin}/\texttt{minx}$.

To avoid the overflow in the self potential, the mass of particles must be smaller than $\texttt{minm} * 64 * (\texttt{eps}/\texttt{minx})$

If one is not using the treecode, the above criterions for the maximum possible mass of a particle rarely cause any trouble. However, if you use a treecode, you end up with a "particle" with the mass comparable to the total mass of the system. Thus, the above limitation becomes difficult to follow. Consider the case you have $10^6$ equal-mass particles. If you use the Heggie units, the total mass is unity and the typical radius of the system is also unity. `minm` would be $10^{-6}$, and `minx` would be $10^{-3}$. With treecode, it is very likely to have one particle to see the node with 1/8 of the total mass, at the distance of the typical system size. So, $m = 1/8$ and $r = 1$. This mass actually exceeds the limit of $\texttt{minm} * 64 * \texttt{rmin}/\texttt{minx}$ by a factor of two.

In order to fix this problem, Version 1.5 (dated July 31, 1994 or later) introduced a new function, `set_mass_correction_factor(c)`. This function changes the internal system of units so that the actual value of mass in GRAPE-3A hardware is $c$ times the value set by `set_mj`. The calculated force and potential are scaled approproately. Thus, if one calls `set_mass_correction_factor(c)`, the limit of maximum mass now becomes $1/c$ times larger than the value without correction. This function must be called before you call `set_mj`. Note that this function is *unsafe*, since it allows the forces from light particles to *underflow*. However, as far as you are using the treecode, it is very unlikly that the underflow actually occurs.

## 3.8 LINKING

Two libraries (libgrape3a.a and libdgrape3a.a) are provided. The first one is the library for single precision calculation and the second one is for double. Contact your system manager to know where they are. If you have installed the software following the instructions in *GRAPE-3A Installation Guide*, the libraries are in the directory `/usr/grape3/gp3abd`, where `/usr/grape3a` must be replaced by the directory you have actually installed the software.

Note that the difference between the single- and double-prec libraries are not in their accuracy, since both library use the same hardware with the accuracy still lower than the single precision. The difference is simply that the single-prec library takes single-prec arguments and the double-prec library takes double-prec arguments. If your program is written in double-prec and it is difficult to convert it to single prec, use double precision. In most workstations the single-prec calculation

offers sufficient accuracy, and you do not really need double precisition.

The debugging-mode libraries (libgrape3a_g.a and libdgrape3a_g.a) are also available. If you use them, each subroutine prints out a lot of messages about the arguments it received and the numbers it sends/receives from GRAPE and other things. They might be useful for some cases.

### 3.9 BUGS

Some of the library functions do not exist/are not well tested in the GRAPE-3 library.

Library with the date earlier than April 8, 1993 returns 254, not 255, when the neighbor list was overflown.

Library with the date earlier than Apr 10, 1993 has the following problem: When you call `set_mj` with mass in the range of `[minm, 16384*minm)]`, the mass is first converted to integer. This conversion causes huge error if, for example, `mass = 1.9 minm`, since `mass` is converted to `minm`.

Library with the date earlier than July 31, 1994 does not have function `set_mass_correction_factor`.

### AUTHOR

Junichiro Makino

### UPDATE HISTORY FOR REFERENCE MANUAL

```
20-Sept-91    Created.                          JM
26-Nov-91     Document for neighbor list.       JM
29-Nov-91     Fortran interface.                JM
5-Feb-93      Document for library.             JM
8-Mar-93      Self-interaction, accuracy etc.   JM
19-Mar-93     Changed to LATEX form.            JM
28-Mar-93     A number of typographical errors.
8-Apr-93      BUG in g3getnbl fixed             JM
              - now returns 255
              - old version returns 254
10-Apr-93     Maximum mass clarified            JM
              Bug fix in set_mj()
31-Jul-94     set_mass_correction_factor        JM
              implemented
```

## 4 Questions and Answers

### 4.1 More than 32768 particles.

Q: I want to use more than 32768 (or 131072 on PCB boards) particles in my code. Is it possible and how?

A:The algorithm is simply to divide N particles to small subsets, each containing less than 32k particles. Then, for each subsets, you first send positions (g3setxj) to GRAPE-3A, and then calculate the forces on ALL particles from that subset, and accumulate the partial accelerations. The program `grape3a` does this. The code would look like (note: the following code is UNTESTED.)

```
      real x(3,nbody), acc(3,nbody), pot(nbody)
      real minx,minm,eps
      real awork(3,4),pwork(4)
      integer njmax,g3maxptcl
      call g3init()
      njmax = g3maxptcl()
      call g3setscales(minx,minm)
      call g3seteps2(eps2)
c
c     clear acc and pot here
c
      do 10 jstart = 1, nbody,njmax
         jend = jstart + njmax - 1
         if(jend .gt. nbody) jend = nbody
         nj = jend - jstart + 1
         call g3setn(nj)
         do 20 j=jstart,jend
            j_board = j - jstart
            call g3setxj(x(1,j),j_board)
            call g3setmj(m(j),j_board)
20       continue
         do 30 i=1,nbody,4
            ii = 4
            if(i+ii .gt. nbody+1) ii = nbody - i + 1
            call g3frc(x(1,i),awork,pwork,ii)
            do 60 j = 1,ii
               do 70 k = 1,3
                  acc(k,i+j-1)=acc(k,i+j-1) + awork(k,j)
 70            continue
               pot(i+j-1) = pot(i+j-1)+pwork(j)
 60         continue
 30      continue
10    continue
      call g3free()
c
c     Do the correction for the self-potential here
c
```

### 4.2 Saving time by not sending mass at each timestep.

Q: In my program, the mass of particles is constant. Can I save time by setting it only once at the beginning of the simulations?

A: Yes. However, in that case you have to occupy the GRAPE-3A hardware throughout one simulation, since if you release the hardware, someone might use it and set the different values of mass to the GRAPE-3A hardware. Therefore, you call `g3init` only at the beginning of the program and `g3free` only at the end of the program. I do not recommend to do this because of the following

reasons. (a) The gain in speed is quite small. Even if you have rather small number of particles, the gain you get is several percents. If you have large N ( $> 10^4$), the gain would be less than one percent. (b) If you call `g3free` only at the end of the program, no one can use GRAPE-3A while your program is running. This is not very convenient if several people are using GRAPE-3A. Even if you are the only user, it is very useful to be able to run small test runs without stopping the production runs.

### 4.3 Is `g3init` a subroutine or a function?

Q: In the reference manual, `g3init` and `g3free` are functions. However, in sample programs they are used as subroutines. Which is correct?

A: Both. You can use it both way, since you can ignore the return value in the present implementation. It may be better to use it as a function, since in future implementation the return value might have meaning.

### 4.4 `g3nchips` gives 0.

Q: `g3nchips` returned 0. Why?

A: There are several possible reasons. The first one is that you did not declare it as integer. You need have the line

```
integer g3nchips,g3maxnnb,g3readnbl,g3getnbl
```

so that these functions are recognized as integer function. The other point is that you have to refer them as, *e.g.*, `g3nchips()`. You must not omit the last parenthesis. Thus

```
integer g3nchips
nchips = g3nchips
```

is INCORRECT and you should write

```
integer g3nchips
nchips = g3nchips()
```

### 4.5 Numbering of particles (in fortran).

Q: How the particles and chips are numbered?

A: Numbering starts with 0 for BOTH of C and fortran interfaces. This applies to all interface functions. For example, the neighbor list returned from GRAPE-3A also assumes the starting index is 0. If you use standard fortran convention with which the starting index is 1, you have to add 1 to each element of the returned neighbor list.

### 4.6 How to assign different neighbor spheres to different particles.

Q: How can I calculate the neighbor list for all particles, each with its own neighbor sphere radius?

A: The following code fragment would work for single-board configulation. For multi-board configulation, see section 3.4. The following code is for 4-chip wire-wrapped board. (note: THIS IS UNTESTED)

```
      do 30 i=1,nbody,4
         ii = 4
         if(i+ii .gt. nbody+1) ii = nbody - i + 1
c
c        set h2
c
         do 40 j = 1,ii
            call g3seth2(h2(i+j-1),j-1)
 40      continue
c
c        set h2 of unused chips to 0
c
         do 45 j = ii + 1, 4
            call g3seth2(0.0,j-1)
 45      continue
         call g3frc(x(1,i),awork,pwork,ii)
         if(g3readnbl() .ge. g3maxnnb()) then
c
c           the number of neighbors is too large.
c           here, you might want to
c           a) reduce h and try again
c           b) calculate neighbor list on host
c           c) or do something else.
c
         endif
         do 50 j = 1,ii
            nbcount(i+j-1) =  g3getnbl(j-1,nblist(1,i+j-1))
 50      continue
         do 60 j = 1,ii
            do 70 k = 1,3
               acc(k,i+j-1)= awork(k,j)
 70         continue
            pot(i+j-1) = pwork(j)
 60      continue
 30   continue
```

### 4.7 How to assign different softenings to different particles.

Q: Can I use different softenings on different particles?

A: Yes. You can do that using the function `g3seteps2tochips`, in the similar way to give different `h2` using `g3seth2`. However, note that the resulting softening is not symmetric, since when calculating the force on a particle, the same softening length is applied to all forces from all particles. In other word, when you calculate the force on particle $i$ from particle $j$, the softening length specified for particle $i$ is used.

### 4.8 Unit for softening and neighbor radius.

Q: What is the unit for the softening length and neighbor sphere radius?

A: The same unit as used for position. Since `eps2` and `h2` are supplied in squared value, the allowed range for them is ($\texttt{minx}^2$, $\texttt{minx}^2 \times 2^{38}$).

### 4.9 Value of gravitational constant.

Q: What is the value of G (gravitational constant)?

A: it is one. If you want to use $G$ different from 1, you multiply the force and the potential you got from GRAPE-3A by the value of $G$ you want to use.

### 4.10 Neighbor list seems incorrect.

Q: In some cases, the neighbor list returned from GRAPE-3A seems incorrect. Some particles with distance slightly larger than the specified neighbor sphere radius appear in the list and some particles with the distance slightly smaller than the specified neighbor sphere radius do not. Why?

A: This is due to round-off. The distance calculated by GRAPE-3A contains the round-off error of about 1% on average. So, particles with the distance a few percent larger than the neighbor sphere radius can be included in the neighbor list and vice versa. If it is very important to have the accurate neighbor list, give GRAPE-3A the radius, say, 10% larger than the true value and select on the host computer the real neighbors.

### 4.11 System crash on SPARCstation 10/LX/5/20.

Q: When I tried to run two programs on SS-10, system sometimes crashed. Why? What can I do?

A: This problem is due to some problem in SUNOS 4.1.3. Both SS-10 and SS-LX (they are both "sun4m" machines, according to `arch -k` command) running 4.1.3 suffer this problem. Thus, you can do one of the following three things: (a) Upgrade to Solaris 2.x. This solves the problem (This is first observed by Rainer Spurzem). (b) Switch to non-sun4m machines, such as SS-2. (c) If you have no other choice than using 4.1.3 *and* SS-10 (or any sun4m machines), be careful not to run two programs.

## 5 Update History

Version 1.0: March 18, 1993

— Created
Version 1.1: March 21, 1993
— Reference manual for subroutine libraries, Q&A, etc.
Version 1.2: March 24, 1993
— A number of typographical corrections (By Dr. Hozumi).
Version 1.3: March 28, 1993
— More typographical corrections. (By Philippe Brieu)
Version 1.3.1: Apr. 1, 1993
— Comment on the behavior of self-potential for `eps2=0`.
– Comment on single/double precision library.
Version 1.4: Sep. 17, 1993
— Support for the new PCB board and multi-board systems.
Version 1.4A: Jul. 18, 1994
— Additional descriptions of the usage of the neighbor list functions.
Version 1.5: Jul. 31, 1994
— new function g3setmascorr (set_mass_correction_factor).