G5nbPIPE User's Guide

for GRAPE-7 software package version 2.1

K & F Computing Research Co.
E-mail: support@kfcr.jp

Abstract

In this document, we give a description of G5nbPIPE, a backend logic for GRAPE-7 add-in card. G5nbPIPE is an extention of G5PIPE. It calculates gravitational forces among particles as well as G5PIPE does. In addition, it creates lists of neghbor particles. This document focuses on functions newly added to G5nbPIPE. For usage of functions inherited from G5PIPE, see *G5PIPE User's Guide.*

Contents

1   Difference from G5PIPE

G5nbPIPE is an extention of G5PIPE. It calculates gravitational forces among particles as well as G5PIPE does. In addition, it creates lists of neghbor particles and stores them into its internal memory, (shown in figure 1 as *NB Memory*). The peak performance of G5nbPIPE is lower than that of G5PIPE. This is because hardware resources required to implement one pipeline logic increased due to the additional function, resulting decrease of the number of pipelines, N_{pipe}, integrated into the FPGA.
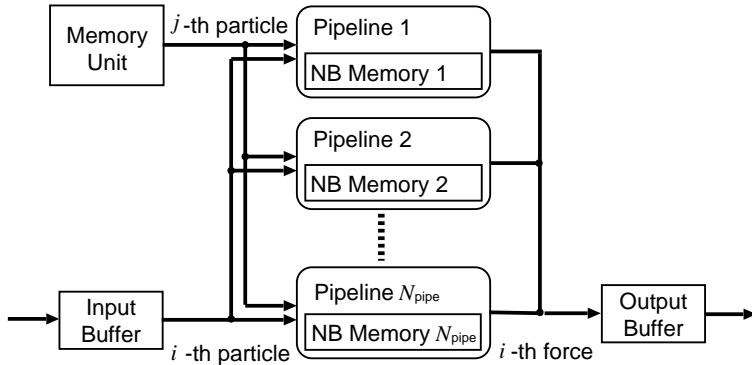

Figure 1: A schematic of G5nbPIPE.

Table 1 summaryises specification of G5nbPIPE. Figures per add-in card are shown in the table. The peak performance is, for example, that of one FPGA chip multiplied by the number of chips on the card. The number of pipelines is that of one FPGA multiplied by the number of chips. The figures shown are subject to change, and it is not recommended to hardcode them into your application program.

Table 1: G5nbPIPE Specifications

	Peak performance (Gflops)	Pipeline number	clock cyvle(MHz)	Memory unit size (# of particles)	NB memory size (# of particles)
Model 100	71	14	133	4095	24
Model 300	182	48	100	12285	72
Model 600	365	96	100	24570	144
Model 800	731	26	185	32764	96

In the rest of this document, we focuses on functions newly added to G5nbPIPE. For usage of functions inherited from G5PIPE, see *G5PIPE User's Guide*.

2   Usage of G5nbPIPE

2.1   Compilation and Linkage

G5nbPIPE can be conroled via G5PIPE library functions. This library includes functions to handle lists of neighbor particles, as well as all functions included in G5PIPE

library. In order to use the library in your own application program, you need to include g5nbutil.h in your code. You also need to link G5nbPIPE library (libg75nb.a), HIB library (libhib.a), and C-language standard math library (libm.a). Following line shows an example of compilation command:

```
cc -o foo foo.c -L/usr/g7pkg/lib -I/usr/g7pkg/include -lg75nb -lhib -lm
```

2.2   A Functionality Test Program

You can use a command ./scripts/checknb.csh in order to check functionallity of G5nbPIPE. Usage of checknb.csh is the same as check.csh, which is described in *GRAPE-7 Installation Guide* section 3.2.

The command checknb.csh runs many-body simulations in the same way as check.csh does. The final distribution of all particles for each run is saved to a temporary file. Then the file is compared with its corresponding in /usr/g7pkg/direct/snapshots. Furthermore, at the final step of the run, all neighbor particles of all particles are also saved to a temporary file, and compared with its correnspondig.

The test would take several tens of minutes. If you see any error message during the test, please make a contact with us at support@kfcr.jp.

2.3   Environment Variables

Environment variables supported by G5nbPIPE library is the same as that supported by G5PIPE library.

2.4   Running Sample Programs

The GRAPE-7 software package includes following sample codes:

```
/usr/g7pkg/direct/direct
/usr/g7pkg/direct/directa
/usr/g7pkg/direct/directmc
/usr/g7pkg/direct/directtest
/usr/g7pkg/vtc/vtc
/usr/g7pkg/directf77/direct
/usr/g7pkg/direct/directnb
/usr/g7pkg/direct/directnba
```

Here we assume the package is installed in /usr/g7pkg. In the directory /usr/g7pkg/direct, you can find direct, directa, directmc, directtest, directnb and directnba.

Only two codes, directnb and directnba perform creation of neighbor-particle lists. See *G5PIPE User's Guide* for description of other codes.

The code directnb calculates gravitational force among paritcles in the same way as directtest does. At the same moment, it creates lists of particles which reside nearby i-particles. directnba is the same as directnb except that it shows a tiny animation.

3

Particles listed up as neighbor of an *i*-particle which has index 0, are shown in red. Other particles are shown in yellow.

In `directnb.c`, you can find a function `calc_gravity5()`. In this function, creation of neighbor-particle lists and calculation of gravitational forces are performed simultaneously. Sample codes in `G5PIPE User's Guide` section 5.3 and a function `calc_gravity4()` in `directtest.c` may help your understanding of this function.

2.5 Difference from GRAPE-5/GRAPE-6A

Most functions in G5nbPIPE library provided in order to handle neighbor-particle lists are the same as those of legacy GRAPE-5 library (`libg5a.a`) and GRAPE-6A library (`libg65.a`). Exceptions are listed below:

- The neighbor-particle memory of G5nbPIPE is smaller than that of GRAPE-5. Its size, i.e., the maximum number of neighbor particles which can be stored for each *i*-particle, is obtained by `g5_get_nbmemsize()`. The number returned by this function is summarised in table 1. However, the number should not be hardcoded into application programs, since it is subject to change.

- The returning value of `g5_get_neighbor_list()` is changed. In the legacy GRAPE-5 library, the function always returns length of the neighbor-particle list. In G5nbPIPE library, the returned length is multiplied by -1, if the list is overflown. The behavior of the function in old and new libraries are the same, if the list is not overflown.

- A new function `g5_set_nbmemsize()` is added. sets the maximum length of a neighbor-particle list. After calling this function, only a limited part of the neighbor-particle memory is transfered to the host computer. This function can be used to reduce unnecessary data transfer from GRAPE-7 to the host computer, if the number of neighbor particles you need to obtain is known to be smaller than the size of neighbor-particle memory.

3   Reference for G5nbPIPE Library Functions (only for functions added to G5PIPE library)

G5nbPIPE library functions are programming interface to manipulate G5nbPIPE. The library includes functions to handle lists of neighbor particles, as well as all functions included in G5PIPE library. In the following, only functions which are used to handle lists of neighbor particles are described. For description of functions included in G5PIPE library, see *G5PIPE User's Guide* (`/usr/g7pkg/doc/g5user.pdf`).

3.1   Synopsis

3.1.1   Standard Functions in C

The following functions provide high-level programming interface to manipulate G5nbPIPE. The functions described here hide the number of the cards to the user. This approach simplifies the user program. As long as user controls G5nbPIPE via these functions, the user can handle multiple cards as a single object. The user does not need to care about how many cards are attached to the host computer.

```
void g5_set_h_to_all(double h);
void g5_set_h(int ni, double *h);
int  g5_read_neighbor_list(void);
int  g5_get_neighbor_list(int ip, int *list);
int  g5_get_nbmemsize(void);
int  g5_set_nbmemsize(int size);
```

3.1.2   Primitive Functions in C

The following functions provide low-level programming interface to manipulate individual G5nbPIPE configured in each GRAPE-7 card. For each invocation of these functions, user need to specifies the device ID (`devid`) explicitly. The device ID of each GRAPE-7 card is obtained using `/usr/g7pkg/hibutil/lsgrape` utility. See *GRAPE-7 Installation Guide* for its usage. Meaning of the arguments other than `devid` are the same as those of corresponding standard function described in the previous section.

```
void g5_set_h_to_allMC(int devid, double h);
void g5_set_hMC(int devid, int ni, double *h);
int  g5_read_neighbor_listMC(int devid);
int  g5_get_neighbor_listMC(int devid, int ip, int *list);
int  g5_set_nbmemsizeMC(int devid, int size);
int  g5_get_nbmemsizeMC(int devid);
```

3.1.3 Standard Functions in Fortran

The following subroutines and functions provide programming interface in Fortran. They provide the same functionality as their counterparts in C.

```
subroutine g5_set_h_to_all(h)
real*8 h

subroutine g5_set_h(ni, *h)
integer ni
real*8 h(*)

function g5_read_neighbor_list()
integer g5_read_neighbor_list

function g5_get_neighbor_list(ip, *list)
integer ip
integer list(*)
integer function g5_get_neighbor_list

function g5_get_nbmemsize()
integer g5_get_nbmemsize

function g5_set_nbmemsize(size)
integer size
integer g5_set_nbmemsize
```

3.1.4 Primitive Functions in Fortran

The following subroutines and functions provide programming interface in Fortran. They provide the same functionality as their counterparts in C.

```
subroutine g5_set_h_to_allMC(devid, h)
integer devid
real*8 h

subroutine g5_set_hMC(devid, ni, *h)
integer devid
integer ni
real*8 h(*)

function g5_read_neighbor_listMC(devid)
integer devid
integer g5_read_neighbor_listMC

function g5_get_neighbor_listMC(devid, ip, *list)
integer devid
integer ip
integer list(*)
integer g5_get_neighbor_listMC

function g5_get_nbmemsizeMC(devid)
integer devid
integer g5_get_nbmemsizeMC

function g5_set_nbmemsizeMC(devid, size)
integer devid
integer size
integer g5_set_nbmemsizeMC
```

3.2 Description

3.2.1 Standard Functions in C

void g5_set_h_to_all(double h)   sets a neighbor radius h. When you perform a force calculation after setting the radius, a list of neighbor particles is created for each i-particles. A neighbor of an i-particle is defined as a particle that resides within the distance h from the i-particle. You can obtain the list using `g5_read_neighbor_list()` and `g5_get_neighbor_list()`.

void g5_set_h(int ni, double *h)   sets neighbor radii $h[0]...h[ni-1]$ for ni i-particles. The number ni must not exceed the value returned by `g5_get_number_of_pipelines()`.

int g5_read_neighbor_list(void)   read neighbor-particle lists into a library-internal work space. This function returns 0 or 1. The value 0 indicates that all neighbor-particles of all ni i-particles set by `g5_set_xi`(ni, xi) are successfully obtained. The returning value 1 indicates that some of the neighbor particles are lost since the neighbor-particle memory has overflown. Use `g5_get_neighbor_list()` to identify the i-particle whose neighbor-particle list has overflown.
   This function call must precede `g5_get_neighbor_list()`.

int g5_get_neighbor_list(int ip, int *list)   returns length of the neighbor-particle list, n_{ip}, for the ip-th i-particle set by `g5_set_xi`(ni, xi). The value is multiplied by -1 and $-n_{\mathrm{ip}}$ is returned, if the list has overflown. That is, the sign of the returned value tells if the list has overflown or not, and the absolute value tells the number of neighbor particles obtained.
   Indices of the neighbor particles are returned to $list[0]...list[n_{\mathrm{ip}}-1]$. These indices are valid even if the list has overflown. Note that the indices returned to `list` indicate positions in the array mj and xj passed to `g5_set_jp(adr, nj, mj, xj)`. On the other hand, the value ip indicates a position in the array xi passed to `g5_set_xi(ni, xi)`.
   `G5_read_neighbor_list()` must be called before calling this function.

int g5_get_nbmemsize(void)   returns the maximum length of a neighbor-particle list. Table 1 summarises the length for each model.
   In the case of Model 300 and Model 600, multiple neighbor-particle memories in multiple pFPGAs are used to store neighbor particles for a single i-particle. Since each pFPGA can store up to 24 neighbors for one i-particle, Model 300 (which has 3 pFPGAs) and Model 600 (which has 6 pFPGAs) can store up to 72 and 144 neighbors, respectively. Note that a neighbor-particle list may overflow even if the number of neighbors is not exceeding these numbers, since the list may overflow if one of the multiple memories is overflown.
   In a simulation using Model 600, for example, only 25 neighbors may cause an overflow, if all of them resides in a same memory unit. In order to reduce the occurrence of such a situation, `g5_set_jp(adr, nj, mj, xj)` sends a j-particle which has index $6n + k$ to the memory unit of the k-th pFPGA. Here, n is a non-negative interger and k is one of

0, 1, 2, 3, 4 and 5. This shuffles the distinations of j-particles neighboring each other in the source array xj and mj.

int g5_set_nbmemsize(int size) sets the maximum length of a neighbor-particle list. After calling this function, only the first *size* neighbor particles are transfered from the neighbor-particle memory to the host computer (more precisely, the number of transfered particles is set to a multiple of the number of FPGAs not exceeding *size*). This function can be used to reduce unnecessary data transfer from GRAPE-7 to the host computer, if the number of neighbor particles you need to obtain is known to be smaller than the size of neighbor-particle memory.

When a *size* exceeding the size of neighbor-particle memory is given, the maximum length of the list is set to the size of the memory itself. When a negative *size* is given, 0 is set as the maximum length. This function returns the maximum length actually set.

3.2.2   Primitive Functions in C, Functions in Fortran

The behavior of all primitive functions is mostly the same as that of standard ones. The only difference is that the former can individually handle each card. The behavior of all functions in Fortran is the same as that of C's.

4 Examples

4.1 An example in C

The following code shows how to perform creation of neighbor-particle lists for ni i-particles. The neighbor particle is defined as a particle that resides within a "neighbor radius", h, from the ii-th particle. For the ii-th particle, one list $nblist[ii]$ is created, which contains indices of the neighbor particles.

```c
#include <stdio.h>
#include "g5util.h"
#define NJMAX (10000)    // For simplicity, these numbers are hardcoded
#define NPIPES (300)     // in this sample. In practical code, however,
#define NBMEMSIZE (200)  // these should be obtained by G5nbPIPE library
                         // functions.

void main(int argc, char **argv)
{
    int i, ii, nj, step, final_step = 100;
    double h, eps, size, dt;
    static double mj[NJMAX], xj[NJMAX][3], vj[NJMAX][3];
    static double a[NJMAX][3], p[NJMAX];
    int npipes, nbof;
    static int nnb[NPIPES], nblist[NPIPES][NBMEMSIZE];

    readnbody(&nj, mj, xj, vj, argv[1]);
    g5_open();
    size = 10.0;
    g5_set_range(-size/2.0, size/2.0, mj[0]);
    npipes = g5_get_number_of_pipelines();
    h = size*0.01;
    for (step = 0; step < final_step; step++) {
        g5_set_jp(0, nj, mj, xj);
        g5_set_eps2_to_all(eps*eps);
        g5_set_h_to_all(h);                    // set neighbor radius.
        g5_set_n(nj);
        for (i = 0; i < nj; i += npipes) { // inside this loop,
            int ni = npipes;                   // create neighbor-particle
            if (i+ni > nj) {                   // lists for ni i-particles.
                ni = nj-i;
            }
            g5_set_xi(ni, (double (*)[3])xj[i]);
            g5_run();
            g5_get_force(ni, (double (*)[3])a[i], p+i);
            // read the lists into a library-internal buffer.
```

```
            nbof = g5_read_neighbor_list();
            if (nbof == 1) {
                fprintf(stderr, "some NB lists overflown\n");
            }
            for (ii = 0; ii < ni; ii++) {
                // obtain the neighbor-particle list of the ii-th i-particle.
                nnb[ii] = g5_get_neighbor_list(ii, nblist[ii]);
                if (nnb[ii] < 0) {
                    fprintf(stderr, "NB list of particle %d overflown\n", ii);
                    nnb[ii] *= -1;
                }
                fprintf(stderr, "NB list length of particle %d is %d\n", ii, nnb[ii]);
            }
            /* --- here, do whatever you want to do using the lists. --- */
        }
        integrate(xj, vj, a, dt, nj);
    }
    g5_close();
    writenbody(nj, mj, xj, vj, argv[2]);
}
```

References

[1] Kawai A., Fukushige T., Makino J., and Taiji M.,
    GRAPE-5: A Special-Purpose Computer for N-Body Simulations,
    Publ. Astron. Soc. Japan (2000), Vol. 52, p. 659,
    `http://xxx.lanl.gov/abs/astro-ph/9909116`.

Acknowledgment

Modification History

version	date	description	author(s)
2.1	11-Feb-2008	Created.	AK