GRAPE-6 User's Guide - Additional functions for Multi-host configuration

Junichiro Makino

Department of Astronomy,
School of Sciences,
University of Tokyo,
Hongo 7-3–1, Bunkyo-Ku, Tokyo 113-0033, Japan
Phone : 81–3–5841–4276
Fax : 81–3–5841–7644
E-mail : makino@astron.s..u-tokyo.ac.jp

Version 0.0: Sept. 2, 2001
Version 0.1: Dec. 13, 2001

Abstract

I give the description of the GRAPE-6 interface library functions to support multiple-host, multiple-GRAPE configuration (what is called "monolithic configuration" in the User's Guide).

Contents

1 TODO

- Perhaps better to provide several functions which use MPI functions?

2 Changes

2.1 Version 0.0

- Aug 29, 2001 — created

2.2 Version 0.1

- Dec 13, 2001 Fix the description of `g6_setup_njdata`

3 Overview of multi-host configuration

There are two different ways to use multiple host GRAPE-6 system. In the first way, each GRAPE-6 unit is connected to its host, and communicates only with its host. The host computers (or multiple program threads if we use an SMP host with multiple GRAPE-6 units) communicate with each other through some method, such as message-passing library API (MPI, PVM etc) or through shared memory. In this case, no additional functions are needed to use multiple GRAPE-6 systems. Each host takes care its GRAPE-6, through standard GRAPE-6 API.

This method is simple, but has potential performance problem. Consider the simplest case where we use two hosts and two GRAPE-6 systems. We divide N particles into two subsets each with $N/2$ particles, and let host 0 take care of subset 0 and host 1 subset1. For simplicity, we consider the shared-timestep, direct-summation method. (Note that for the case of the tree method on distributed-memory processors this configuration is well suited.)

At each timestep, we need to calculate the forces on all particles. We can do this by sending all particles to GRAPE-6 through `g6_set_j_particle`. However, since each host knows about only its subset, it can only send half of the particles. In order to send the other half, two host computer need to exchange their data.

This exchange, can of course be done through, for example, MPI call such as `MPI_send_recv`. However, it should be noted that this is time-consuming, in particular we have more than two hosts. if we have p hosts, one host need to receive data from all $p-1$ hosts. Therefore, for N particles, the data communicated is $N(p-1)/p$. In other words, the data to be sent to one host *increases*, albeit slowly, as we increase the number of processors. By increasing the number of processors, we expect to speedup the overall calculation. Both the work of time integration and that of the force calculation are indeed distributed to p processors and can be done in $1/p$ time, compared to the case of 1 processor. However, the communication time remains essentially constant.

This situation is the same in the case of the individual timestep.

For GRAPE-6, we implemented a way to avoid this communication bottleneck, which we describe in the next section.

3.1 The second way: the use of multicast network

Figure 1 show the conceptual hardware setup for two-host, 4-GRAPE system. 4 GRAPE boards are connected to two hosts through two network boards (NB). Two network boards are connected to each other.

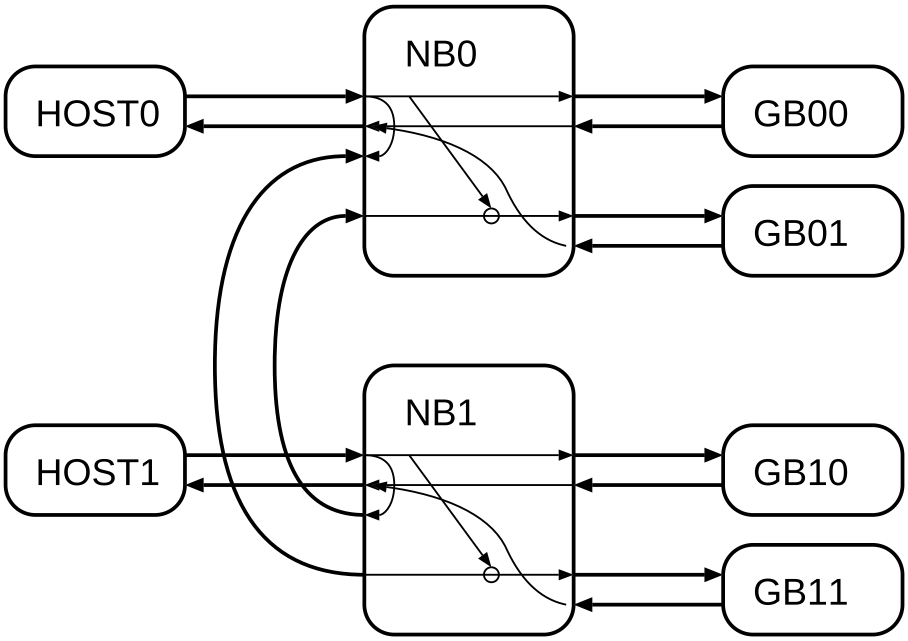

Figure 1: A two-host, 4-GRAPE configulation

The NB, in this case, operates in two different mode. In the first mode, which we call the normal mode, or N-mode, it broadcast the data received from its host to all GRAPE boards. This mode is used to send i particles to GRAPE boards. Host 0 broadcasts the data to GRAPE 00 and 01, and host 1 to GRAPE 10 and 11

In the second, "multicast" mode (M-mode), an NB sends the data received from its host to GRAPE board x0, and the data received from the other NB to GRAPE board x1. Each NB simply send the data received from its host to the other NB. Thus, effectively, in this mode host 0 broadcasts the data to GRAPE 00 and 10, and host 1 GRAPE 01 and 11. In other words, the direction of the broadcast is diagonal to that in N-mode.

We use N-mode to send i-particles and M-mode to send j-particles.

In this way, we can eliminate the communication between the hosts. GRAPEs x0 have subset 0 and x1 subset 1. Thus, GRAPE xy calculates the forces from subset y to subset x.

By adding more NBs, we can scale up this configuration to larger one, such as 4-host, 16-GRAPE system or 8-host, 64-GRAPE system. For practical reasons, what we have currently in Hongo site is multiple 4-host, 16-GRAPE systems.

A 4-host, 16-GRAPE system can be used in 3 modes. In the first mode, we use 4 hosts independently, each with 4 GRAPE boards. In this case NB always works in N-mode. Both j and i particles are broadcasted to all boards. However, different chips take care of different j particles.

In the second mode, we use the whole system as one unit. In this case, we switch to M-mode when sending j-particles, and switch back to N mode to send i-particles. In M-mode, 4 GRAPE boards connected to one NB receive data from different hosts. Which board receive data from which host is rather complex, but the point is that different boards receives data from different data, and therefore 4 GRAPE boards combined receive data from ALL hosts.

In the third mode, we can use two hosts combined, to have 2-host, 8-GRAPE system. In this case, M-mode is a bit modified so that two GRAPE boards are grouped together to receive data from one host.

In the next section, I describe the library functions to switch between two mode, and give simple example.

4 Example

The following is a somewhat simplified version of the code I use in my simple NBODY1 like code with multiprocessor.

```
.....
    g6_setmccount(nprocs);
    g6_open(grape6_id);
.....

void particle_system::update_grape_data(int  n_next)
{
    vector j218 = vector(0.0,0.0,0.0);
    g6_change_cbjpmode(grape6_id, 1);
    MPI_Barrier(MPI_COMM_WORLD);
    g6_reset(grape6_id);
    g6_set_ijp_mode(grape6_id, 1);
    for (int i = 0; i< 200; i++) g6_dummywait();
    g6_reset(grape6_id);
    for (int i = 0; i < n_next; i++) {
        particle *bi = nt[i].pptr;
        vector j6 = ONE_SIXTH*bi->get_jerk();
        vector a2 = 0.5L*bi->get_acc();
        g6_set_j_particle(grape6_id,
                          bi->get_grape_index(),
                          bi->get_index(),
                          bi->get_time(),
                          bi->get_timestep(),
                          bi->get_grape_mass(),
                          (real*)&j218,
                          (real*)&j6,
                          (real*)&a2,
                          (real*)bi->pget_vel(),
                          (real*)bi->pget_pos());
    }

    g6_set_ijp_mode(grape6_id,0);
    MPI_Barrier(MPI_COMM_WORLD);
    g6_change_cbjpmode(grape6_id,0);
    MPI_Barrier(MPI_COMM_WORLD);
    g6_reset(grape6_id);

}
```

The first function, `g6_setmccount` tells the GRAPE-6 library what configuration it should use when asked to send j-particles. We can call it with either `nprocs` equal to 1, 2, or 4. This argument of course must be the same as the number of processors we actually use. In the case of MPI, most likely one would use `MPI_Comm_size` to get the number of processors. This function can be called anytime, even before calling `g6_open`.

The next function, `update_grape_data`, shows the full code to update j-particles after one blockstep in the case of individual timestep (blockstep) integrator. It first switches from N-mode to M-mode, and then actually sends j-particles and finally switches back to N-mode.

This part:

```
g6_change_cbjpmode(grape6_id, 1);
MPI_Barrier(MPI_COMM_WORLD);
g6_reset(grape6_id);
g6_set_ijp_mode(grape6_id, 1);
for (int i = 0; i< 200; i++) g6_dummywait();
g6_reset(grape6_id);
```

is the code to switch to M-mode. The first function, `g6_change_cbjpmode(grape6_id, 1)`, actually changes the mode. If the second argument is one, it lets the NB to M-mode, and if 0, to N-mode.

The remaining functions essentially just reset the hardware after the mode is changed. Note that we are now in message-passing environment. The host computers need to perform barrier synchronization after `g6_change_cbjpmode(grape6_id, 1)` is called, because otherwise other hosts might not have finished the call.

The call to `g6_set_ijp_mode(grape6_id, 1)` change the internal mode of each GRAPE-6 board (now in M-mode, which means one host is now connected to GRAPE-6 boards under different two or four different NBs) to the state to receive j-particle data. I believe this function is not really necessary, since it is called from `g6_set_j_particle` anyway, but I have not yet tested the code without this function called here...

The next line, the call to `g6_reset(grape6_id)`, is to reset the hardware after mode change. This call is necessary since changing the mode can put the system into a strange state.

The call to `g6_set_ijp_mode(grape6_id, 1)` takes effect in something like 1 microsecond after executed. The next dummywait loop is to spin-wait about that amount of time. This is of course a rather dangerous way, since whether this works or not depends on the speed of the host computer. When the host computer is replaced, this code might suddenly stop working. The final call to g6_reset is also probably not necessary.

The second loop simply send j-particles, in the way exactly the same as in the case of single-host code.

The final part switches back to M-mode.

```
g6_set_ijp_mode(grape6_id,0);
MPI_Barrier(MPI_COMM_WORLD);
g6_change_cbjpmode(grape6_id,0);
MPI_Barrier(MPI_COMM_WORLD);
g6_reset(grape6_id);
```

Here, again, the call to `g6_set_ijp_mode(grape6_id,0)` is probably unnecessary. Before we call `g6_change_cbjpmode(grape6_id,0)`, we need to perform barrier synchronization because even when one host has finished sending its j-particle data, others might still be sending data, and if so, it should wait all other hosts to finish sending data before switching back to M-mode. Also, after calling g6_change_cbjpmode, another barrier synchronization is necessary. If we omit this synchronization one host might send some data before some other host changes its NB from M-mode to N-mode. Thus, even if one host has its mode changed, the data it sends might still be received by GRAPE-6 boards under some other hosts, resulting in some unpredictable effects.

5 Reference for functions

5.1 g6_setmccount

```
int g6_setmccount(int mccount)

subroutine g6_setmccount(mccount)
integer mccount
```

This function tells the GRAPE-6 interface library how many hosts are used in parallel. Using this information, the library determines what configuration should be used in M-mode. This function should be called before g6_open.

5.2 g6_set_ijp_mode

```
int g6_set_ijp_mode(int clusterid, int mode)

subroutine g6_set_ijp_mode(clusterid,mode)
integer clusterid, mode
```

When called with mode=0, this funnction puts GRAPE-6 to the state to receive i-particles and other commands. When mode=1, it puts GRAPE-6 to the state to receive j-particles. In usual cases, this function is called internally from other interface functions and it is not necessary for an application program to call this function
explicitly.

5.3 g6_set_cbjpmode

```
int g6_set_cbjpmode(int clusterid, int mode)

subroutine g6_change_cbjpmode(clusterid,mode)
integer clusterid, mode
```

When called with mode=0, this function puts GRAPE-6 to N-mode with mode=1, to M-mode.

5.4 g6_setup_njdata

```
int g6_setup_njdata(int clusterid, int nbody)

subroutine g6_setup_njdata(clusterid,nbody)
integer clusterid, nbody
```

This function must be called before the number of particles you send to the memory (2nd argument of firsthalf/lasthalf) changed. This function must be called only when cbjpmode=1 and ijp_mode=1.

6 Important Notes

There are several quite important guidelines which must be followed to obtain meaningful result in parallel host applications. I'll describe what I'm aware of in the following.

6.1 Number of particles

The number of j particles on all host must exactly be the same. For example, when the total number of particles is 1024 and you use 4 hosts, the number of j-particles to be taken care by each host must be 256. If the number of particles is not an integer multiple of the number of hosts, you have to add dummy particles (with zero mass is fine), so that all hosts have the same number of particles.

This is simply because the argument `nj` for the firsthalf/lasthalf pair is the same for all boards, while they receive the data from different hosts.

6.2 Particle indexing

The GRAPE-6 hardware uses the particle index to avoid self-interaction and also for neighbor calculations. Therefore, the index must be unique over the whole N-body system. One has to assign index so that any of two particles, whether or not they are on the same host or not, must have different indices.

6.3 Error recovery

When a host recognizes a hardware error returned from GRAPE-6, the error may be due to the data sent from some other host. Therefore, the simplest way of the recovery from the error is to let ALL hosts perform the recovery procedure when ANY of the hosts detects error.

6.4 Preventing deadlocks

If more than two multi-host programs are running simultaneously, it is possible that they go into the deadlock state. For example, if program A and B both use g6host0 and g6host1, they need to acquire GRAPE-6 on both machine. Thus, if, for example, on g6host0 A acquires GRAPE-6 and on g6host1 B acquires GRAPE-6, A will be waiting B to release GRAPE-6 on g6host1 and B will be waiting A to release GRAPE-6 on g6host0. Either program cannot go further.

Therefore, some mechanism which prevents such deadlock from taking place is absolutely necessary. Such a mechanism is actually quite simple to implement. If both of programs A and B first try to acquite GRAPE-6 on g6host0, and proceed to g6host1 only if it acquired GRAPE-6 on g6host0, there will be no deadlock. Therefore, a code like the following (Note that this is UNTESTED yet!) should do the work.

```
int i, numprocs, myid;
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Barrier(MPI_COMM_WORLD);
for(i=0;i<numprocs; i++){
    if (myid == i) g6_open(0);
    MPI_Barrier(MPI_COMM_WORLD);
}
```

7 Local info (compile, link and run)

Currently, we have two multi-host systems. One is a 4-host system (g6host0 through g6host3) and the other is a 2-host system (g6host6 and g6host7). The machines g6host4 and g6host5 exist, but without GRAPE-6 boards yet.

7.1 Compile and Link

To compile and link MPI programs, use mpixx (f77, cc, CC) with an enviromental variable

```
setenv MPIROOT /opt/mpich/mpich-p4/mpich-1.2.1
```

This MPICH is based on GCC version egcs-2.91.66.

7.2 Run

To run a 4-host program, use `mpirun -np 4` on g6host0 (when it's ready...).

To run a 2-host program, use `mpirun -np 2` on g6host6.