This is the file readme of `HiGPUs`. If there are any problems please do not hesitate to contact us :

**Mario Spera**: mario.spera@live.it or mario.spera@uniroma1.it
**Davide Punzo**: Davide.Punzo@roma1.infn.it
**Roberto Capuzzo Dolcetta** : roberto.capuzzodolcetta@uniroma1.it.

## 0.1 Introduction

`HiGPUs` is a parallel direct $N$-body code based on a 6th order Hermite integration scheme. The code uses, at the same time, MPI,OpenMP and CUDA libraries to fully exploit all the capabilities offered by the most modern hybrid supercomputers in the world. Moreover it is implemented using block time steps in order to ensure a fast and precise integration on both small and large space and time scales. If you use HiGPUs for scientific work, we kindly ask you to reference the following paper: "A fully parallel, high precision, N-body code running on hybrid computing platforms" by R. CapuzzoDolcetta, M. Spera, D. Punzo (2012, JCP) , available in arXiv: 2012arXiv1207.2367C. Feel free to suggest modifications, improvements, additions, etc. to the code contacting the authors.

## 0.2 How to use the code

First of all you should check if CUDA or OpenCL (depending on the downloaded version) and MPI libraries are installed on your machine. The code is known to work with CUDA 4.0/5.0 or OpenCL 1.1/1.2 and OpenMPI 1.5.4, under a Linux distribution, using the following kind of GPUs as computing accelerators:

1. Tesla C1060
2. Tesla C2050
3. Tesla M2070
4. Tesla M2090
5. GeForce GTX 480
6. GeForce GTX 580
7. GeForce GTX 670
8. GeForce GTX 680
9. Radeon HD 6970
10. Radeon HD 7850
11. Radeon HD 7870

12. Radeon HD 7970

In any case, your computing accelerator must support double precision (64-bit) arithmetic because `HiGPUs` computes the mutual distances and cumulate accelerations in double precision even if the intermediate operations are done in single (32-bit) precision.

## 0.2.1 Prerequisites

### Drivers

If you have not done it yet, you need to install the specific drivers for your GPU device which are strictly related to your Operating System. Go to

`http://www.nvidia.com/Download/index.aspx?lang=en-us` for nVIDIA GPUs and to

`http://support.amd.com/us/gpudownload/Pages/index.aspx` for AMD GPUs.

**Important note** : `HiGPUs` has been successfully tested with AMD GPUs using the following couples :

1. **Catalyst 12.8 + Ubuntu 10.04.4**
2. **Catalyst 12.11beta + Ubuntu 12.04**

As our knowledge and experiments, other combinations between Linux OS and AMD drivers fail. For nVIDIA drivers we did not have problems.

### Software

If you would like to use a nVIDIA GPU you need to install CUDA (or the nvidia OpenCL implementation which is included in the installation package of CUDA). If you have not done it yet, go to the web site `https://developer.nvidia.com/cuda-downloads` and follow the instructions in the Getting Started Guide.

If you would like to use an AMD GPU, you need to install the AMD OpenCL release. To get the AMD APP SDK with OpenCL Support you can visit the web site `http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/downloads/`.

To get OpenMPI, you can download the package at the web address

`http://www.open-mpi.org/software/ompi/v1.5/` and follow the instructions contained in the readme file.

## 0.2.2 How to compile the code

To use the code, first of all you need to untar the downloaded archive "HiGPUs_CUDA.tar.gz" or "HiGPUs_OpenCL.tar.gz" with the command line

```
tar -zxvf HiGPUs_CUDA.tar.gz
```

This will create the directory `HiGPUs_CUDA` which contains the sources and all other files needed for running the code. The content should look like:

1. docs
2. exec
3. lib
4. Makefile
5. src
6. README

At this point, before compiling the code, you have to tell the Makefile where to find CUDA (or OpenCL) and MPI libraries. To do this, you have to modify the following two lines of the Makefile setting appropriately the variables:

1. `CUDA_INSTALL_PATH := /usr/local/cuda (or OPENCL_INSTALL_PATH := /opt/AMDAPP)`
2. `MPI_INSTALL_PATH := /usr/openmpi`

where we have assumed that the installation paths are : `/usr/local/cuda` for CUDA, `/opt/AMDAPP` for the AMD OpenCL implementation and `/usr/openmpi` for OpenMPI.

Before compiling the code you can add the following flags to the variable MYOPTS:

1. `DCHECK_ERRORS` : it activates some useful functions which may help to find possible errors during the run-time. If this option is enabled, the program terminates its execution telling at what line the error has been detected and giving further information about its type.
2. `DCHECK_TIMES` : it activates some functions which measure the time spent in executing the main sections of `HiGPUs`. The results are written on the hard disk drive every certain interval of time (see below for the details).
3. `DPLUMMER` : it enables an external Plummer-like potential field which adds an analytical contribution to the pure mutual interaction between the stars. To set the related parameters (core radius $b$ and total mass $M_p$) see below (section : **Running the code**). If we call $r$ the distance of a generic particle from the origin of the system of reference, the added potential is of the form

$$\phi(r) = -\frac{M_p}{\sqrt{r^2 + b^2}} \tag{0.1}$$

   having considered $G = 1$.
4. `DGPUCORR` : if activated, the corrector step is performed by the GPU. This can significantly improve performance especially if the system to integrate contains a quite large number of objects (say $N > 10^4$). This improvement is even more evident if the number of computational nodes on which the simulation is running is large.

5. `DGALAXY` : it enables an external, stationary, analytical Milky-Way like background. The potential is that described by Allen, C. and Santillan, A, Revista Mexicana de Astronomia y Astrofisica (ISSN 0185-1101), vol. 22, Oct. 1991, p. 255-263. To set the related parameters (scale radius $R_s$ and scale mass $M_s$) see below (section : **Running the code**). Putting $G = 1$, this potential is composed by three elements :

   a) **Central Mass Distribution** : if we call $r$ the distance of a generic particle from the origin of the system of reference, this element takes the form

   $$\phi_{bulge}(r) = -\frac{M_1}{\sqrt{r^2 + b_1^2}}$$

   ;

   b) **Disk Component** : if we call $R$ the distance of a generic particle from the $z$-axis, the disk component takes the form

   $$\phi_{disk}(R, z) = -\frac{M_2}{\sqrt{R^2 + \left[a_2 + \sqrt{(z^2 + b_2^2)}\right]^2}}$$

   ;

   c) **Halo Component** : if we call $r$ the distance of a generic particle from the origin of the system of reference, this element takes the form

   $$\phi_{halo}(r) = -\left(\frac{M(r)}{r}\right) - \left(\frac{M_3}{1.02 a_3}\right)\left[-\frac{1.02}{1 + \left(\frac{r}{a_3}\right)^{1.02}} \ln\left\{1 + \left(\frac{r}{a_3}\right)^{1.02}\right\}\right]_r^{100\,\mathrm{kpc}}$$

   .

   The parameter introduced have the values $M_1 = 1.40592 \cdot 10^{10} M_\odot$, $b_1 = 387.3$pc, $M_2 = 8.5608 \cdot 10^{10} M_\odot$, $a_2 = 5317.8$pc, $b_2 = 250.0$pc, $M_3 = 1.07068 \cdot 10^{11} M_\odot$, $a_3 = 12000.0$pc. Therefore, it is clear that if you want to add this analytical and stationary contribution to your run you need to specify a scale for your simulation. Specifically you need to specify the scale radius (in parsec) and the mass scale (in solar masses) that you would like to use.

Once you have setup the Makefile you can proceed typing:

1. `make clean` or `make dest`
2. `make`

This creates the executable file `HiGPUs.x` in the folder `exec`. The option `make dest` can be used to a perform a more thorough cleaning of the folder `exec`. It will delete also the output data files produced by any previous run.

### 0.2.3 Parameters and input files

The file containing the parameters needed to start a simulation using `HiGPUs` is called `input_param.txt`. It is included, by default, in the folder `exec`. The file `input_param.txt` has the following aspect

```
262144 // number of particles !
2 // number of gpus to use in a single node (this must be power of two)!
128 // gpus threads per block !
1.0 // integration time !
-3.0 // exponent which defines the maximum time step allowed for particles (2^exponent) !
-25.0 // exponent which defines the minimum time step allowed for particles (2^exponent) !
0.0005 // softening parameter !
0.45 // eta parameter for determining particles time steps (generalized Aarseth criterion)!
0.01 // eta parameter for determining particles time steps (Aarseth criterion) !
0.125 // time step for snapshots !
1000000 // maximum number of snapshots !
0 // rescale to the center of mass 1 = true, 0 = false !
0 // rescale to the center of velocity 1 = true, 0 = false !
init.dat // input file for positions, velocities and masses !
Tesla C2050\# // GPU to use (the name must be terminated with the character #) !
48 // Number of resident warps (or wavefronts) per multiprocessor (compute unit).
```

In general, using this configuration, `HiGPUs` will read the file `init.dat` containing a system composed by `262144` stars. This ensemble will be evolved over `1.0` program time units and one snapshot will be produced every `0.125` program time units. To do this, `HiGPUs` will use `2 Tesla C2050` for each computational node, whose number is specified at launch time, (see later) and the particle time steps will never be greater than $2^{-3}$ and smaller than $2^{-25}$.

Some important notes : at the moment, `HiGPUs` works fine only if the number of stars in the system to simulate is exactly a **power of two**. Therefore it is of crucial importance that, the first parameter of `input_param.txt` is a power of two (in the example above $262144 = 2^{18}$). The number of gpu threads per block can vary. It represents the way to map the work to accomplish on the GPU : particles must be organized in blocks. On average, we obtained a good balance in terms of GPU load and performance putting this value to 128. In any case, `HiGPUs` considers this value as a maximum value and modifies it automatically if and when it is needed.

The time step for a generic particle $\Delta t_i$ is determined performing an average between the standard Aarseth criterion (for the 4th order Hermite) and the generalized one for the Hermite 6th order scheme, that is

$$\Delta t_i = \frac{\eta_{sixth}}{2}\left(\frac{A^{(1)}}{A^{(4)}}\right)^{\frac{1}{3}} + \frac{\sqrt{\eta_{forth}}}{2}\left(\frac{A^{(1)}}{A^{(2)}}\right)$$

where

$$A^{(k)} = \sqrt{\left|a^{(k-1)}\right|\left|a^{(k+1)}\right| + \left|a^{k}\right|^2}$$

and $a^{(k)}$ is the $k$-th derivative of acceleration. This ensures a correct determination of the

time steps even for very peculiar situations. For example, in some situations, the generalized Aarseth criterion can suffer very much of round-off errors and it tends to overestimate the higher order derivatives of the acceleration producing very small time-steps. This behaviour is controlled when this criterion is averaged with the standard Aarseth criterion. This average ensures correct integration even when the options `-DPLUMMER` or `-DGALAXY` are enabled. Typical values for eta parameters are, for `HiGPUs`,
$\eta_{sixth} = 0.45$ and
$\eta_{forth} = 0.01$.

The input data file, whose name is specified in `input_param.txt` (in the example above it is `init.dat`), must contain the $N$-body data for the simulation in the following format :
`position_X position_Y position_Z velocity_X velocity_Y velocity_Z mass`
An example of input data file (called `random.dat`) is included in the folder `exec`.

You need also to specify the name of the GPU that you want to use to simulate your system. If you do not know the exact name of the GPU `HiGPUs` automatically detects all the computing accelerators which are installed on the system and it will list them in the file `HiGPUslog.dat` when you run it. Therefore the exact names will be shown in `HiGPUslog.dat`. If the name specified in `input_param.txt` is not exact, `HiGPUs` will list the available GPUs and than it will stop its execution.

The last parameter in `input_param.txt` is included only in the OpenCL version of HiGPUs and it represents the maximum number of resident warps (or wavefronts) per multiprocessor (compute unit). For nVIDIA GPUs you need to discover the compute capability of your GPU in the document `Compute_Capability.pdf` included in the folder docs of the OpenCL version of the code. Then, look at the Table 10 : "Technical Specifications per Compute Capability" of pag. 149 of the included document `CUDA_C_Programming_Guide.pdf` and search the maximum number of resident warps per multiprocessor corresponding to the compute capability of your GPU. For AMD GPUs use the value of "Max Wavefronts / CU (avg)" for your GPU in the Appendix D : "Device Parameters" of the
`AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf` indluded in the folder docs of `HiGPUs`. This parameter is needed to `HiGPUs` to determine the structure and the computing capability of the GPU device and to optimize the simulation for that accelerator.

### 0.2.4 Running the code

Now you are ready to run the program. If you are using a machine with just one node you can simply type
```
mpirun -np 1 ./HiGPUs.x
```

Since in our program we established a one to one correspondence between MPI processes and computational nodes, it is strongly recommended to run only one MPI process per node. Each MPI process can manage all the GPUs available on each node without problems. At this point, if you want to use more than 1 node, for example `n` nodes you can use the following command to start the simulation:
```
mpirun -np n ./HiGPUs.x
```

If you run the program with the option `-h` you can obtain further useful information:

```
--------------------------------------------------------------------------------
----------------------Help of HiGPUs--------------------------------------------
--------------------------------------------------------------------------------
Usage : mpirun -np [#nodes] ./HiGPUs.x [options]
Options :
-f [file]  (input_param.txt) = it specifies the file containing the simulation
                               parameters
-h = it shows this help screen
-r [file] = restart the simulation from the specified file (HiGPUslog.dat
            is necessary !!)
-v [value] = rescale to a specific "value" of virial ratio
-d [id]:[id]: ... = enable manual selection of the available gpu devices
-p b=[value]:M=[value] = if you compiled with -DPLUMMER, you need to specify
                         the parameters b and M of the Plummer potential
-gal R=[value]:M=[value] = if you compiled with -DGALAXY, you need to specify
                           scale radius R (pc) and scale mass M (M_sun)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

### 0.2.5 Output

The following file will be generated or updated every time a new snapshot is produced. If `MAX` is the maximum number of snapshots specified in the file `input_param.txt` we get :
`(MAX + i).dat` : it contains positions, velocities and masses (in this order) of all the particles in the system as they are at the time of the $i$-th snapshot.

`cpu_memory.dat` : this file reports, every time the function `CPU_memcheck()` is called, the ram memory used by the program at that point.

`Blocks.dat` : it reports the histogram of the block time steps scheme. That is, in the first column the base-2 logarithm of the time step is reported (i.e. the index of the block) while the second column represents the number of particles in that block.

`energy.dat` : it shows the simulation global time versus :

1. the ratio $\frac{E-E_0}{E_0}$ where E is the total energy at the current time and $E_0$ is the total initial energy;
2. total kinetic energy of the system ($T$);
3. total potential energy of the system ($U$);
4. Virial Ratio ($2T/|U|$).

`HD_state.dat` : some info about the space available on your Hard Disk Drive

HiGPUslog.dat : it summarizes the simulation information. It could be something like:

```
================================================
Read parameters file : input_param.txt
N : 262144
Gpus : 2
Threads per block : 128
Time of integration : 1.0
Max time step : 0.125
Min time step : 9.31323e-10
softening : 0.0005
eta 6th order : 0.45
eta 4th order : 0.0001
time for printing : 1.0
Max output files : 1000000
CDM scale : 0
CDV scale : 0
Input file : input_N_262144
Process id : 30298
================================================
Available : Tesla C2050 as device : 0
Available : Quadro as device : 1
Available : Tesla C2050 as device : 2
================================================
Using : Tesla C2050 (device 0)
Using : Tesla C2050 (device 2)
================================================
#Initial Total Energy : #-3.00234523435865e-3#
```

If you have specified the option DCHECK_TIMES some files (called times_1xxxxxx.dat) will be generated. They contain (in order) :

```
M : number of particles updated.
NEXT     : time required to choose the particles that have to be updated.
           These particles are stored in the array named 'next'.
CPY_NEXT : time required to send the array of the indexes of the
           particles to update to the GPU.
PRED     : time required to copy the array 'next' from the Host
           to the GPU and to perform the predictor step.
EVAL     : time required to calculate the accelerations of the particles
           contained in the array 'next'.
REDU     : time required to reduce forces.
REPOS    : time required to collect accelerations.
CPY_ACC  : time required to copy accelerations from the GPU to the Host.
MPI      : time required to perform the MPI_Allreduce() function to
           reduce and collect forces from all the computational nodes.
CORR     : time required to perform the corrector.
CPY_REC  : time to send corrected values of velocities, positions and
```

```
           the reduced accelerations to the GPU. (Only if Corrector is
           performed by the CPU)
RECON    : time required to reorder data on the GPUs after the corrector
           step. (Only if Corrector is performed by the CPU)
THREADS  : threads per block used in the GPU
TOTTHREAD: total number of threads which runs on the GPU
BFACT    : Correspondence between particles to update and GPU threads.
           The correspondence is 1 particles for BFAC threads.
```