Guide for future users of the Milan computational cluster (including some administrator tricks)

Enrico Ragusa

May 2, 2018

This guide is the fruit of the experience I gained during my master thesis and two PhD years at the university of Milan. I will try to include here most of the information needed by any user or not so experienced administrator in order to run numerical simulations, setting the environment and some useful tricks and scripts for submitting jobs using the queue manager HTCondor.

I do not expect that the content of this guide represents the best or at least proper way to do things. Most of this are self-thaught notions that I learned as magic spells on the internet. My phylosophy of life regarding computers is "If it works, do not make further questions...". It might seem to be a bit naive or superficial as an approach, however I realized that any time you invest to fix some issues is worth the effort, the rest is wasted: so my personal opinion is that it is better to understand how to fix one specific problem than learning everything is required to do it in the most efficient way (unless doing it efficiently is the core of the problem). So please don't blame me if the solution to some issues appears as a ridicoulous approach to your eyes or if you could find a much faster and elegant way.

Finally please email me if something is not completely clear or simply it does not work as it should. My home directory reflects quite well my attitude toward chaos in real life, i.e. it's a complete mess, perfectly representing the concept of geological disorder: what is older is also somewhere deeper. It might be that during the reordering I did to write this guide something went missing.

1 Getting started

The first thing you need is an account on the machine of the milan astrophysics group VIRGO. It can be reached using ssh typing on your UNIX terminal:

```
ssh -XY username@virgo.fisica.unimi.it
```

The -XY option allows and forces the forwarding of graphic windows. If you want to use gedit or other programs whose output is prompted on a graphical window (for example if you want to make some plots, or simply if you use a graphic text editor as gedit) it is required.

1.1 What should I do if I am not able to open graphic windows?

There might be two main reasons why you are not able to open graphic windows:

- First you have not set properly the X11 forwarding on your laptop: to do so, create the file /.ssh/config and add the following lines

```
Host *
  ForwardAgent yes
  ForwardX11 yes
```

- Your account or (much worse) the machine has run out of available space on the disc: not being able to open any window despite the -XY options is the classical symptom that for some reason you are not able to write in the Xauth file on your home directory: typically this occurs when you run out of disc space (both if you exceed your quota or the machine disc is completely full: **df -h**, look for the percentage of usage of the home dir).

1.2 User administration

If you still do not have an account, you should ask someone with administrator privileges to type:

```
useradd -G <groupname>,<othergroup>,.....(nospaces) <name>
```

Then he can set the password for the user typing:

```
passwd <name>
```

The group name for the people working with PHANTOM is **ProtoPl**. The group is of paramount importance if you are willing to work with PHANTOM: at login time the machine recognizes your group and initializes all the environment variables needed to use PHANTOM and SPLASH.

If the administrator wants to change the environment settings for the members of a group he should create, edit, and make executable the file /etc/profile.d/custom.sh using:

```
touch /etc/profile.d/custom.sh
vim /etc/profile.d/custom.sh
chmod u+x /etc/profile.d/custom.sh
```

the content of the custom.sh file is currently

```
#!/bin/bash

for group in `id -Gn`; do
    if [ "$group" = "ProtoPl" ]
        then
         export LESS=JSiM
         export SYSTEM=gfortran
         export OMP_NUM_THREADS=16
         export OMP_STACKSIZE=2G
```

```
        export OMP_SCHEDULE="dynamic"
        export GIZA_FONT='Arial'
        export LD_LIBRARY_PATH=/usr/lib64:/usr/local/lib/libgiza
    fi
done
```

the for loop cycles on the names of the groups and set the environment variables for the group ProtoPl. Other useful commands for administrators are: check if a group exist

```
grep <groupname> /etc/group
```

check if user exist

```
grep <name> /etc/passwd
```

add an existing user to a group

```
usermod -G <groupname> <Name> -a
```

NB: the -a option append the group to the existing otherwise you lose all the previous ones. change ownership of a file[1]

```
chown (-R) <user>:<group> file (<or dir>)
```

1.3 Getting PHANTOM

You have two possibilities: downloading the tarball, and downloading the developer version.

For the first, download the tarball from the `https://phantomsph.bitbucket.io/#download`. You can do it directly on VIRGO without copying it from your computer using

```
wget https://phantomsph.bitbucket.io/releases/phantom-v0.9.tar.gz
```

be careful to download the most updated release (change phantom-v0.9.tar.gz with the most recent one).

For the second, if you have an account on bitbucket.org and you want to use the developers version (which is obviously the most updated version) type

```
git clone https://USERNAME@bitbucket.org/danielprice/phantom
```

this will clone the git repository. git is a powerful version control software that tracks the changes you make to the code, so that you can always go back to a previous version of the code. From this folder you will be able to push the changes you make to the code to the online repository (hence pay much attention to what you do).

If you are planning to run a large set of simulations, you need to download also some scripts to make your experience with the queue manager more comfortable. To do that type from your home directory:

```
git clone https://USERNAME@bitbucket.org/EnricoRagusa/submitbin.git
```

[1] Important to notice that users are part of the group named with their name, so that the ownership of a random user's file is ⟨user⟩:⟨user⟩.

Doing this from your home directory allows you not to edit the $PATH environment variable. All users part of the ProtoPl group have the directory $HOME/submitbin already set in their $PATH variable.

This repository is public, do whatever you want with the scripts but please if you make some modifications be sure they are not breaking anything before pushing them.

Keep in mind that the submitbin folder contains the most updated version of the source files producing this pdf. You might wish to compile the pdf going in the folder HowtoTeXGuide and typing:

```
pdflatex tutorialSim.tex
bibtex tuttorialSim.aux
pdflatex tutorialSim.tex
pdflatex tutorialSim.tex
```

the repeated commands are necessary in order to have all the internal label-references and bibliography set properly.

2 Setting up a PHANTOM simulation

You can read this first section to understand a couple of important issues regarding the compilation of the executables, if you are not interested and you want directly to run a simulation, you can directly jump to Section 2.2.

2.1 Environment variables

PHANTOM is a modular code, which means that some of the choices regarding the simulation you are going to run need to be done already when you are compiling the code (e.g. the code is not compiling the sections with selfgravity or dust evolution if they are not added on purpose). This allows to reduce the possible choices when the code is executing with benefits for the code computational efficiency. Beside the main executable PHANTOM there are also two other main programs: one to create the first dump (initial conditions) PHANTOMSETUP and the other to post process the results PHANTOMANALYSIS.

Depending on the user needs, some environment variables are set by the value of some environment variableless in the phantom/build/Makefile. The setting of these variables is intrinsically done when you type make with the "local" Makefile (I will explain it in a minute), all these variables are set and the compiler builds the executables following the indications prescribed by these variables. To simplify this process, the variables are grouped under few general variables such as SYSTEM and SETUP.

Let me deepen briefly this approach looking at the meaning of a couple of sections of the Makefile. The information regarding the compiler are stored in the environment variable SYSTEM, which in our case is automatically set to SYSTEM=gfortran when you login on VIRGO

```
ifeq ($(SYSTEM), gfortran)
    FC= gfortran
    FFLAGS+= -O3 -Wall -Wno-unused-dummy-argument -frecord-marker=4 -gdwarf-2 \
            -mcmodel=medium -finline-functions-called-once -finline-limit=1500\
```

```
            -funroll-loops -ftree-vectorize \
           -std=f2008 -fall-intrinsics
    DBLFLAG= -fdefault-real-8 -fdefault-double-8
    DEBUGFLAG= -g -fcheck=all -ffpe-trap=invalid -finit-real=nan\
              -finit-integer=nan -fbacktrace
    KNOWN_SYSTEM=yes
    ENDIANFLAGBIG= -fconvert=big-endian
    ENDIANFLAGLITTLE= -fconvert=little-endian
    OMPFLAGS = -fopenmp #-fmax-stack-var-size=10000000
    CC = gcc
    CCFLAGS = -O3
    LIBCXX = -lstdc++
endif
```

this set gfortran as the compiler for most of the source files with some flags (FFLAGS). We will see that this is important for our purposes when using the queue manager HTCondor.

The other important variable is the SETUP variable which encapsulate the setting of some source files used when compiling. This is the list of the variables set when you choose SETUP=dustydisc

```
ifeq ($(SETUP), dustydisc)
#   locally isothermal discs
    FPPFLAGS= -DDISC_VISCOSITY
    SETUPFILE= setup_dustydisc.f90
    MODFILE= moddump_dustadd.f90
    MAXP=200000
    ISOTHERMAL=yes
    DUST=yes
    DUSTFRAC=yes
    KNOWN_SETUP=yes
    IND_TIMESTEPS=yes
    ANALYSIS=analysis_dustydisc.f90
    KERNEL=quintic
endif
```

When the code will be compiled, the precompiler will look for #ifdef statements in the source code (files with .F90 instead of .f90) and will include or discard those sections. E.g. -DDISC_VISCOSITY (equivalent to have DISC_VISCOSITY=yes) turn on the modifications required in order to use the artificial viscosity as a physical Shakura & Sunjaev viscosity as explained (Price et al., 2017).

2.2 How to create a new simulation

Create a new folder to store the results of your new simulation. Enter this folder and call script writemake.sh ⟨setup-name⟩ contained in the phantom/scripts/ folder. If the phantom folder is in your home ∼/ directory, typing

```
$HOME/phantom/scripts/writemake.sh dustydisc > Makefile
```

will create a "local" Makefile where the SETUP variable, previously described, is set to "dustydisc". Depending on what is written in the environment variables of this SETUP, you will have a simulation with (with reference to the

second verbatim section in the previous page, from top to bottom): artificial viscosity as disc viscosity; using the file setup_dustydisc.f90 as a plugin module of PHANTOMSETUP executable; using the file moddump_dustadd.f90 as a plugin module of PHANTOMMODDUMP executable; setting the length of the array as 2×10^6 (set the memory footprint, if you use less particles you are simply not using the entire array); the simulation uses a locally isothermal equation of state; it evolves dust (DUST), using the one fluid algorithm (DUSTFRAC=no will set the two fluid algorithm); it uses individual timestepping; is uses the file analysis_dustydisc.f90 as a plugin module for PHANTOMANALYSIS; it uses a quintic kernel to smooth particle quantities.

Now that we have the Makefile we type

```
make
make setup
make analysis
```

to have the three main executables (many other executables can be compiled for different purposes). The local Makefile copies the executables in the folder where you type make. It might happen that if you are trying to compile the executables with a large number of particles ($N_{\rm part} \gtrsim 2.5 \times 10^6$), you might need to set

```
ulimit -s unlimited
```

otherwise when executing it will try to use an amount of stack memory larger than the default allowed to the user. Furthermore, if you are trying to compile with a very very large number of particles ($N_{\rm part} \gtrsim 10^7$) you might need to edit the $-mcmodel$ flag in the FFLAGS variable under the gfortran $SYSTEM setting of the phantom/build/Makefile as follows

```
ifeq ($(SYSTEM), gfortran)
    include Makefile_defaults_gfortran
ifneq ($(UNAME), Darwin)
   FFLAGS+= -mcmodel=large
endif
endif
```

We now create the first dump of the simulation using

```
./phantomsetup <nameofsim>
```

this will prompt an interactive menu. The output of this first call of ./phantomsetup is the production of a nameofsim.setup file. Typing again

```
./phantomsetup <nameofsim>
```

in presence of a nameofsim.setup file in the folder will produce the first dump, named nameofsim_00000.tmp, it is a binary file (to read it use ssplash nameofsim_00000.tmp, see Section 4). As a general rule, if you are planning to make several simulations with very similar parameters, I suggest to go through the interactive setup once and then to copy the .setup file manually or using a script (in Section 5.1 an example of a script that I use to make faster the setting up of multiple simulations).

Beside the creation of the first dump. A file nameofsim.in is created after you run for the second time PHANTOMSETUP (or first time if the .setup file is already present in the directory). I will discuss this file in the next section.

3 Running the simulation

The file nameofsim.in mentioned in the previous section contains some technical specifications of the simulation, the most important are: the number of the equation of state (ieos), options controlling viscosity (I will discuss this briefly in a dedicated Section 3.3.1), options controlling the dust, duration of the simulation and other output options. Most of these quantities are set automatically by the setup, which already contains some of these information, in other situations you need to change it manually if you need them to be different. If you need to set up a large number of simulations you might wish to edit the setup source in order to have the .in written with your preferences (this will avoid the manual editing of all the .in files). Another trick, that however works only for the output options, is to put a .in file in the folder with the output options already set, they will not change when running PHANTOMSETUP.

It's time to run your simulation! There are three possibilities:

1. You run your simulation on VIRGO: 16 core machine, one simulation at a time, with two you halve the computational power per simulation. this is a good choice if you need to make some tests.

2. Submitting a job with the queue manager HTCondor from VIRGO in one of the clusters of the ATLAS collaboration. You will have 240 core on 10 different machines. Each simulation will be faster than on VIRGO, and you can launch multiple simulations at a time. This represents a good choice to start dealing with the queue manager. However since we run multicore simulations, when the queue is long, the chances to have an entire machine empty to run your simulation are very low, and your simulations do not run.

3. Submitting a job with the queue manager HTCondor from GASPARE in a very broad machine pool which include almost all the machines of the department, which become available when they are not used. This is the best and most efficient choice. Among the machines at your disposal, you can count on the machines BALDASSARRE and MELCHIORRE which are two armored tanks with 64 cores and 128 Gb RAM!

IMPORTANT if you are willing to run the simulation using point 2. or 3. you need to recompile PHANTOM using the -static flag: since it will run on a different machine, you might get some error concerning dynamic libraries (.so files[2], whose path is contained in the LD_LIBRARY_PATH variable). To do it you need to change the FFLAGS in the file build/Makefile_defaults_gfortran adding -static as follows:

```
FC= gfortran
FFLAGS+= -O3 -Wall -Wno-unused-dummy-argument -frecord-marker=4 -gdwarf-2 \
        -mcmodel=medium -finline-functions-called-once -finline-limit=1500\
         -funroll-loops -ftree-vectorize \
        -std=f2008 -fall-intrinsics -static <<<<<<<<########## LOOK THIS LINE!!!
DBLFLAG= -fdefault-real-8 -fdefault-double-8
```

[2]The static linking uses instead the .a files whose path is in the LIBRARY_PATH variable instead

```
DEBUGFLAG= -g -fcheck=all -ffpe-trap=invalid -finit-real=nan\
           -finit-integer=nan -fbacktrace
KNOWN_SYSTEM=yes
ENDIANFLAGBIG= -fconvert=big-endian
ENDIANFLAGLITTLE= -fconvert=little-endian
OMPFLAGS = -fopenmp #-fmax-stack-var-size=10000000
CC = gcc
CCFLAGS = -O3
LIBCXX = -lstdc++
```

It must be said that this procedure can be avoided setting the variable "LD_LIBRARY_PATH=/usr/lib64" in the submission file, instead "LD_LIBRARY_PATH=." as presented in Sec. 3.2.

3.1 Running on VIRGO

This is the simplest case. From the directory of the simulation you type:

```
nohup ./phantom nameofsim.in &
```

nohup is a command required if you want that your simulation keeps running when you logout from the machine. The standard output of PHANTOM that would be prompted on the screen will be automatically redirected to a nohup.out named file in the directory (unless you redirect it to somewhere else using the simbol > otherfile.out before the ampersand symbol &). The ampersand symbol is needed since you want that you job run in the background. If for some reason you want the simulation to stop you can use the command kill

```
kill <job PID>
```

The PID can be obtained typing

```
top
```

and looking for the number in the first column associated to your running job.

3.2 HTCondor from VIRGO

HTCondor (Condor for friends) is a powerful queue manager currently used in our department. It manages the jobs submitted in a global queue in a set of clusters called pool. To have things working properly, be sure you have cloned also the repository SubmitBin as explained at the end of Section 1.3.

The first thing you have to do is to create a submission file. You can do it using one of the scripts contained in the repository SubmitBin as follows:

```
generatesubfile.sh nameofsim
```

This will create a file named "submission_file" starting from the following template:

```
##############################
#                            #
# Submission_file for phantom #
#                            #
```

```
##############################

Universe     = vanilla
Executable   = phantom
Args         = PREFIX.in
Requirements = (TARGET.OpSysMajorVer == 7) && \
               (TARGET.Machine =!= "proof-05.mi.infn.it")
Environment  = OMP_NUM_THREADS=22;OMP_STACKSIZE=512M;\
               OMP_SCHEDULE=dynamic;LD_LIBRARY_PATH=.

transfer_input_files    = PREFIX.discparams,\
                          PREFIX.in,\
                          PREFIX_00000.tmp,\

when_to_transfer_output = ON_EXIT_OR_EVICT

request_cpus = 22

Error   = err.$(Cluster).$(Process)
Output  = out.$(Cluster).$(Process)
Log     = phantom.log.$(Cluster).$(Process)

Queue
```

IMPORTANT: There have been few modifications to the name patterns of the files (e.g., the discparams file is now produced with a different names depending on the options that you choose when producing your first dump). If your submission fail, it could be that you are not transferring the correct files. If this happens, you need to edit the template in order to include in the file transfer_input_files the correct files.

The universe is a setting required by Condor for the type submission, the vanilla one is the less specific (as far as I could understand). The requirements require an OS equal or more recent that CentOS 7, and not to run on the machine 5 of the proof cluster (which gave me some problems in the past). OMP_NUM_THREADS is the number of OMP threads on which your simulation is parallelized (it should be equal to the number of CPUs you are requiring). Transfer_input_files are the files that has to be transfered on the host machine in addition to the executable (the .in file, the .discparams and the starting dump). **Be careful:** if your simulation is starting from a dump different from the 00000.tmp one it needs to be edited manually (unless you are using the submit.sh script I will describe below).

Now you submit your job using:

```
condor_submit submission_file
```

Condor will attribute a number, the "ClusterID", to your job and create some log files in your directory: err.ClusterID (contains std error), phantom.log.ClusterID (contains technical information about the machine that is running the job) and out.ClusterID (contains the std output). You can check the status of the job in the local (all the jobs submitted from VIRGO) queue using

```
condor_q
```

and in the global queue using the wrapper in the SubmitBin repository

```
condorpool
```

To remove the job from the queue type:

```
condor_rm ClusterID
```

This basic approach to the job submission has the following problems: first, you have no access to the intermediate dumps until the simulation end. Second, if your simulation takes 2 weeks to run, the chances that your job is evicted by someone with higher priority are extremely high; every time you get evicted your simulation is restarted from the beginning. Even though sometimes the partial output is copied in the /var/lib/condor/spool folder, this requires a manual restart: copying back in the proper folder all the output and editing by hand all the files for the next resubmission. You can also leave your simulation running again from the beginning implying obviously that your simulation might incur again in an eviction....As a result, your simulation will last forever.

The best workaround to this is to break the simulation in smaller and shorter jobs. Doing this manually is impossible, it would require a huge waste of time from you to constantly check the status of your simulations. Luckily, VIRGO is always awake and aware of what's going on, so we can use a bash script to do this for us as I am going to describe in the following dedicated Section.

3.2.1 Automatic job re-submission

You can use the script submit.sh in the submitbin folder you downloaded in Section 1.3 to make your simulation an array of shorter jobs in order to avoid the situation we just described above. After the submission of the job, the script checks every few minutes if the file out.ClusterID exist and if something is written inside it; if yes, it means that your simulation has finished and is ready for resubmission; the script checks for the number of the last dump and edit the submission_file for the new submission.

How do we force the simulation to become a set of shorter jobs? Instead of running the simulation until "tmax" is reached, PHANTOM allows you to stop your simulation after a given amount of wall time, after a given number of dumps is produced or after a given number of steps is computed. You can achieve this setting your nameofsim.in file as follows:

```
# options controlling run time and input/output
            tmax =   4.570E+04    ! end time
           dtmax =        457.    ! time between dumps
            nmax =          -1    ! maximum number of timesteps \
                                    (0=just get derivs and stop)
            nout =          -1    ! number of steps between dumps \
                                    (-ve=ignore)
       nmaxdumps =          10    ! stop after n full dumps \
                                    (-ve=ignore)
        twallmax =      012:00    ! maximum wall time \
                                    (hhh:mm, 000:00=ignore)
       dtwallmax =      000:00    ! maximum wall time between full dumps \
                                    (hhh:mm, 000:00=ignore)
```

```
      nfulldump =           1    ! full dump every n dumps
       iverbose =           0    ! verboseness of log \
                                   (-1=quiet 0=default 1=allsteps 2=debug 5=max)
```

twallmax=012:00 means that if phantom understand that with the next dump the job will exceed 12 hours of wall time the simulation stop, and Condor transfer back the output to your machine; nmaxdump=10 means that no matter if you reach 12 hours, if you do 10 dumps in less time, your simulation stops and you have back your dump. **NB:** if after the submission your simulation takes more than 12 hours two things might have happened: first, you have been evicted during the first 12 hours and your simulation was restarted; in this case you can simply leave it like it is, in the worst case scenario you have lost the first 11 h 59' 59", the simulation will restart automatically, it will simply take a bit more than expected. Second, the first dump took longer than 12 hours implying that as soon as PHANTOM will produce the first dump, Condor will deliver back the output; in this case you should change the "dtmax" variable to have that less simulation time passes between two dumps, or setting an "nmax" to limit the number of timesteps between two dumps (if you make too many timesteps per dump you are having anyway some problems with the dynamic range, so if you realize that this is the problem check carefully the parameters you chose).

Now that you modified your .in file, after you generated the submission file (as explained above) you submit your job typing

```
nohup submit.sh <N> &
```

where $< N >$ is the number of resubmission you want before the script dies and stop resubmitting. In case the script resubmit a simulation with the same initial dump for two times, the script dies: interpreting it as a problem or as if the simulation has actually ended before reaching the N resubmission. In the ideal world if you have say nmaxdumps=10 and you tmax is set to have 100 dumps, $< N >= 10$ re-submissions should be fine. However, this is almost never the case since it might happen that you are not able to produce 10 dumps in 12 hours. Since the script stops anyway when the simulation ends, my choice is to use a generously large N, say:

```
nohup submit.sh 50 &
```

How to stop the simulation? As before, you first type

```
condor_rm ClusterID
```

but this is not sufficient since the submit.sh will keep running in the background forever. You have 2 possibilities: in the folder of the simulation you want to stop, type

```
lsof nohup.out
```

this will list the IDs of the processes writing in the nohup.out file (even though submit.sh do not produce standard output); you find the PID of the process writing there and you kill it.

The second possibility is to use a wrapper of nohup I wrote (it will remain beta tested forever). You initialize the script typing (do it just the first time ever you are using it):

```
procIDdirInit.sh
```

this creates in your home directory the ProcID directory (you can change procIDdirInit.sh to have ProcID in whatever folder you prefer). Then, instead of using nohup you use wrapnohup. The submission in this case will be:

```
wrapnohup.sh submit.sh 50 &
```

this will store some information of the nohup job in the file $HOME/ProcID/activenohupID.txt. Typing:

```
activeID.sh
```

will produce an output which should look like this:

```
ven 29 dic 2017, 15.30.32, CET: 2096908 \
/home/unimi/enrico.ragusa/simulazioni/2ndCQTau/009q20au1mm2 submit.sh 50
ven 29 dic 2017, 15.31.00, CET: 2097054 \
/home/unimi/enrico.ragusa/simulazioni/2ndCQTau/006q20au1mm2 submit.sh 50
ven 29 dic 2017, 15.31.20, CET: 2097082 \
/home/unimi/enrico.ragusa/simulazioni/2ndCQTau/003q20au1mm2 submit.sh 50
ven 29 dic 2017, 15.58.18, CET: 2101077 \
/home/unimi/enrico.ragusa/simulazioni/2ndCQTau/012q20au1mm2 submit.sh 50
sab 30 dic 2017, 17.38.53, CET: 2306229 \
/home/unimi/enrico.ragusa/simulazioni/3rdCQTauHoR/002q20au1mm2 submit.sh 50
sab 30 dic 2017, 17.39.08, CET: 2306251 \
/home/unimi/enrico.ragusa/simulazioni/3rdCQTauHoR/002q20au1mum submit.sh 50
sab 30 dic 2017, 17.39.27, CET: 2306282 \
/home/unimi/enrico.ragusa/simulazioni/3rdCQTauHoR/004q20au1mm2 submit.sh 50
sab 30 dic 2017, 17.39.44, CET: 2306300 \
/home/unimi/enrico.ragusa/simulazioni/3rdCQTauHoR/004q20au1mum submit.sh 50
```

where time, location and PID of the nohup call has took place. This is the most efficient way I had in my mind to track long nohup jobs. **N.B.** for some reason escaping my understanding sometimes, when a bad closure of the remote connection occurs, you lose track of some jobs with the activeID.sh approach. In that case, you can apply the first lsof method I described to the .wrapout file contained in the folder.

```
lsof submit.sh\ 50.wrapout
```

Finally to check that no submit.sh processes are still running type:

```
ps -ef|grep submit.sh
```

this will list all the processes containing "submit.sh" active on the machine (including obviously the grep call).

3.3 HTCondor from GASPARE

UPDATE: This is not anymore useful for new users since VIRGO has been added to the superpool of which GASPARE is part. Now all the jobs submitted using condor from VIRGO run on a very large pool of machines.

To obtain an account on GASPARE you need to email Franco Leveraro, a guy at the computational centre. I have not administration privileges on this machine so the environment settings that are granted on VIRGO for all the user need to be done by each user on their /home directory. First thing is to change your .bashrc file to include the following environment variables that are automatically set on VIRGO.

```
export SYSTEM=gfortran
export OMP_NUM_THREADS=32
export OMP_SCHEDULE="dynamic"
export OMP_STACKSIZE=1024M
export GIZA_FONT='Helvetica'
export PATH=$HOME/submitbin/:$PATH
```

The submission consists in the same procedure as on VIRGO (described in Sec. 3.2) including the -static thing. Nevertheless, on GASPARE some libraries are missing. In particular, two missing libraries are libgfortran.a and libquadmath.a. These two libraries can be found in the SubmitBin folder you should have already downloaded from bitbucket.org. The $HOME/submitbin/lib folder needs to be added to the LIBRARY_PATH variable. You can do this updating your .bashrc adding this following line

```
export LIBRARY_PATH=$LIBRARY_PATH:$HOME/submitbin/lib
```

For completeness, in case in the future you will compile jobs with dynamic linking on GASPARE for other purposes, add the following paths to the LD_LIBRARY_PATH variable, to do that copy the following line in your .bashrc

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64
```

Remember that in general, whenever you have problems with libraries, you can create your home directory $HOME/lib folder where collect the missing libraries. You can obtain them as copies from other machines with the same OS and architecture, or that are in folders not part of the library paths. In practice, you create the lib folder

```
mkdir $HOME/lib
```

then update your .bashrc adding

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/lib
export LIBRARY_PATH=$LIBRARY_PATH:$HOME/submitbin/lib
```

How many cores should I require here? If you go to your submitbin directory, you will find the template for the submission_file, named with great fantasy submission_file_template. There I suggest you to change the variables **OMP_NUM_THREADS = 16** and **requested_cpus = 16**. In this way you will match the requirements of a very large number of machines. If you require more cores, only few machines will be able to satisfy your requirement with the result that the speed up you would gain with the higher number of cores is completely vanished by the time you wait in the queue for your job to start.

Finally it is important to say that despite two of the machines MELCHIORRE BALDASSARRE are equipped with 64 cores, the tests I ran on these machines showed that the parallelization efficiency decreases rapidly above 32 cores, probably due to the fact that they are 32 physical cores hyperthreaded to 64.

3.3.1 How to install matplotlib on GASPARE

Create in the already existent $HOME/lib directory (if not existent create it) the folder pythonlib

```
mkdir $HOME/lib/pythonlib
```

Now install the libraries using easy-install

```
easy_install --prefix=$HOME/lib/pythonlib matplotlib
```

Now add to your .bashrc file the following line

```
export PYTHONPATH=$HOME/lib/pythonlib/lib/python2.7/site-packages
```

4 SPLASH and PHANTOMANALYSIS

SPLASH is the interactive plotter that is able to read and visualize the binary dump files produced by PHANTOM. VIRGO is periodically updated to the latest version of SPLASH, however on GASPARE you will need to install it on your home directory: follow the instructions at `http://users.monash.edu.au/~dprice/splash/download.html`. If you don't manage to do that, ask me. You can copy my executables and libraries directly in your $HOME/bin folder.

To make some practice with reading and elaborating the output using SPLASH and PHANTOMANALYSIS, in the VIRGO folder /usr/local/phantom/simExamp you will find 10 dumps of a numerical simulation.

With reference to that folder, files testPhantBin_000* are the dumps, they are binary files and can be read by ssplash. Dumps contain information about position, velocity, density and other quantities of each particle. Typing

```
ssplash testPhantBin_00010
```

this will prompt an interactive menu: inserting 2 <enter>, 1 <enter>, 6 <enter>, <enter>, <enter> visualizes the density distribution of the 10th dump as a colour plot.

The PHANTOMANALYSIS allows the user to extract some elaboration of the data contained in the dumps (e.g. the angular momentum of a given disc annulus, the accretion rate at a given radius).

```
./phantomanalysis testPhantBin_00010
```

will produce the ASCII file angm00010, containing the analysis (if you want to change the columns you need to edit the ANALYSIS variable in the SETUP, in Sec. 2.1 with another existing/edited/new-from-scratch module analisys_[....].f90 source file).

```
asplash angm00010
```

will prompt an interactive menu (the a in asplash refer to ASCII file reader) allowing you to make column vs column plots.

4.1 Using python for analysis

You can create the .so dynamic library to import particles positions and velocities in a python script.

```
make pyanalysis
```

This will compile the class PhantomAnalysis. **NB:** to create a dynamic library the flag -static must be disabled! If you get some error, it is likely that this is the reason. I recall that you have probably added the flag -static, since not on all the machines of the farm the required libraries are present. Check that the flag -static in the build/Makefile_defaults_gfortran is not there. Now you can include in your python script the class using:

```
from libanalysis import PhantomAnalysis as pa
```

Then, you can load the dump you want using:

```
dump = pa('nameofyourfile_00000')
```

In my specific case, I experienced also some problems since the script tries to load also the variable for the internal energy, but since I was running a locally isothermal simulation it was not able to find it. Not sure I did something wrong, but I solved it commenting the following line in the file phantom/scripts/pyphantom/phantomanalysis.py:

```
#self.utherm = self.get_part_u(npart)/(udist**2/utime**2)
```

You can access to the values of the variables using a syntax of the type:

```
dump.xyzh
```

If you are using ipython, you can see a list of the variables typing "dump." and then hitting the TAB key to complete the command. A list of the variables will appear.

5 scripts

5.1 Spanning the parameter space

This script simplifies the initialization of a set of a simulation where all the parameters are fixed and you change only the planet mass, the distance of the planet and the grain size. For this to work you need to set up a template.setup file. Below you will find the example also of one of the template.setup file. This script is contained in the submitbin/setmod directory you have downloaded from bitbucket. It allows you to create a ready to launch simulation typing:

```
scriptname.sh <massratio> <planet-separation> <dust-size> <mm/micron>
```

WARNING: you cannot use this script without providing the templates and additional scripts in the launching folder. This is meant to be only an example about how you can use sed to change a template file containing the setup.

```
#!/bin/bash

MASSR=$1;
SEMAX=$2;
SIZE=$3;
UNIT=$4;

######## CATCH ERRORS #########
if [ "$#" -lt 4 ]; then
echo "Too few arguments.";
echo "Usage: setmod.sh q a grainsize unit";
exit 1;
fi

if [ $UNIT != "mm" ] && [ $UNIT != "mum" ] && [ $UNIT != "mm2" ]; then
echo "Unit must be mm, mm2 or mum";
exit 1;
fi

echo "massratio: 0.$MASSR; semimaj-axis: $SEMAX; grainsize: $SIZE $UNIT";

#Setting up the simulation folder
if [ ! -d ""$MASSR"q"$SEMAX"au"$SIZE""$UNIT"" ] || [ -e ./forceit ]; then
  mkdir ""$MASSR"q"$SEMAX"au"$SIZE""$UNIT"";
else
  echo "This simulation already exist force, it touching a file named \
  \"forceit\" in this folder";
  exit 1
fi

cd ""$MASSR"q"$SEMAX"au"$SIZE""$UNIT"";

#copying or creating executables
if [ ! -s ../phantom ] || [ ! -s ../phantomsetup ] || [ ! -s ../phantomsetup ]
then
  ~/phantom_PUBLIC_myfun/scripts/writemake.sh dustydisc> Makefile;
  make;
  make setup;
  make analysis;
else
  cp ../phantom ../phantomsetup ../phantomanalysis .
fi

cp ../templ.setup .
cp ../templ.in ./""$MASSR"q"$SEMAX"au"$SIZE""$UNIT".in"

#Personalzing the setup
if [ $UNIT = "mm" ]; then
  sed 's/MASSRQ/0.'$MASSR'/' < ../templLarge.setup| \
  sed 's/SEMAX/'$SEMAX'/'| \
```

```
  sed 's/SIZE/0.'$SIZE'/' > ./""$MASSR"q"$SEMAX"au"$SIZE""$UNIT".setup";
elif [ $UNIT = "mum" ]; then
  sed 's/MASSRQ/0.'$MASSR'/' < ../templSmall.setup| \
  sed 's/SEMAX/'$SEMAX'/'| \
  sed 's/SIZE/0.000'$SIZE'/' > ./""$MASSR"q"$SEMAX"au"$SIZE""$UNIT".setup";
elif [ $UNIT = "mm2" ]; then
  sed 's/MASSRQ/0.'$MASSR'/' < ../templLargeSameSmall.setup| \
  sed 's/SEMAX/'$SEMAX'/'| \
  sed 's/SIZE/0.'$SIZE'/' > ./""$MASSR"q"$SEMAX"au"$SIZE""$UNIT".setup";
else
 echo "Something wrong with the units chosen, it must be mm or mum"
 echo "This is actually second filter, \
 if this message appears something went really wrong";
 exit 1;
fi

#Creating the setup
./phantomsetup ""$MASSR"q"$SEMAX"au"$SIZE""$UNIT""

#Generating submission file
generatesubfile.sh ""$MASSR"q"$SEMAX"au"$SIZE""$UNIT""
```

Copied here you will find the templSmall.setup, which is one of the template .setup file called in the script. You should notice that the massratio, separation and grain size have been changed to the flags MASSRQ, SEMAX and SIZE, so that calling sed with the substitution option allows you to substitute the flag with the value you desire.

```
# input file for dustydisc setup routine

# resolution
                  np =      750000    ! number of gas particles

# units
               udist =   1.496E+13    ! distance unit in cm
               umass =   1.989E+33    ! mass unit in g

# options for central star
              star_m =       1.500    ! star mass
           accradius =       1.000    ! star accretion radius

# options for gas accretion disc
                R_in =       1.000    ! inner radius
               R_out =        150.    ! outer radius in gas
         iprofilegas =           1    ! set gas surface density \
                                        profile (0=power-law, 1=tapered power-law)
            mass_set =           1    ! how to set gas disc mass (0=disc mass,\
                                        1=surface density, 2=Toomre Q \
                                        (not yet implemented))
              pindex =       0.300    ! p index
```

```
        sigma_naught =   2.818E-07    ! Sigma0 of the gas profile \
                                        Sigma = Sigma0*(R/Rc)^-p*Exp(-(R/Rc)^(2-p))
                 R_c =         56.    ! characteristic radius of the\
                                        exponential taper
              qindex =       0.450    ! q index
            alphaSS =       0.010    ! desired alphaSS
               xinc =       0.000    ! inclination angle
                H_R =        0.10    ! H/R at R=Rin
         binary_set =           1    ! set the binary? (0=no,1=yes)
         setplanets =           0    ! add planets? (0=no,1=yes)

# options for the binary
          massratio =      MASSRQ    ! mass ratio of binary
       semimajoraxis =       SEMAX    ! initial separation of binary (code units)
        eccentricity =       0.000    ! eccentricity of the binary
   accretion_radius2 =       0.600    ! accretion radius for secondary
         inclination =       0.000    ! disc inclination [deg] with \
                                        respect to xy plane

# options for dust accretion disc
   dust_to_gas_ratio =       0.015    ! dust to gas ratio
    profile_set_dust =           1    ! how to set dust density profile \
                                        (0=equal to gas, 1=custom)
        iprofiledust =           1    ! set dust surface density profile \
                                        (0=power-law, 1=tapered power-law)
         pindex_dust =       0.300    ! p index
            R_c_dust =         56.    ! characteristic radius of the \
                                        exponential taper
           R_indust =       1.000    ! inner radius in dust
          R_outdust =        150.    ! outer radius in dust
        grainsizeinp =        SIZE    ! grain size (in cm)
        graindensinp =       3.000    ! intrinsic grain density (in g/cm^3)
```

References

Price D. J., et al., 2017, preprint, (arXiv:1702.03930)