alphacrucis

connect

As with any other Linux machine, you can connect with `alphacrucis` with either one of the two commands, from inside IAG:

```
$ ssh username@alphacrucis
$ ssh username@10.180.0.63
```

For security reasons, the machine does not allow you to connect from outside IAG. Therefore, if you are e.g. at home, you have to connect to your PC at IAG first and then execute one of the previous commands.

Before you start working in `alphacrucis`, you should better check your `~/.bashrc` file, to set the path of the desired fortran compiler. I suspect that, if at the time of applying for an account you checked on that you will be suing fortran, your `~/.bashrc` muist have been filled with all the available fortran compiler and all you have to do is to comment off the lines corresponding to the desired compiler, and comment on the lines corresponding to the rest of the compilers. If you do not expect to use fortran, then just ignore this paragraph.

job queue

In general in big clusters like `alphacrucis`, you should better not simply run/execute programs. All time-consuming commands should be "submitted" to the cluster system, ask for permission to run, and wait for the cluster resources to be freed in order to start your job. This thing is called queuing, i.e. you submit a request to use some resources, and your request is run when the system is ready.

You may perform some tests before continuing. To this purpose, create a file `machines`, in order to list the nodes used by MPI. For example, the following `machines` files defines the execution of a program in 4 nodes and 96 cores.

```
r1i0n0
r1i0n1
r1i0n2
r1i0n3
```

For a run on more nodes, you simply add more lines/nodes in the above file

```
r1i0n0
r1i0n1
r1i0n2
 ...
r1i0n15
```

You normally run the parallel version as:

```
$ mpirun -np 8 ./hdustparv2.02.bc input = despo.inp
```

But it is recommended that even for a test, you should better queue the job. In particular, you should use the queuing system Torque/Maui to submit your jobs. In that case, you do not need to use the file `machines` mentioned above, and instead define the number of nodes and processors you need directly in the submission script. A sample for submission script is `runs/hdust/sample.job` (see below). You submit the job with the command:

```
$ qsub sample.job
$ qstat # see status of your jobs
$ showq # see all running jobs in the cluster
$ psall # see the CPUs running for you right now (IAG alias)
```

With `qstat` you can see the job ID in the first column, which will be something like `[6-digit number].alphacrucis`. You can kill this job with the command:

```
$ qdel [job ID]
```

You can see the situation in ganglia. You can get help from the following Wikis: Gina, emu2009 (invalid link), LAi (Laboratório de Astroinformática).

submission script

A request is submitted via a submission script, as the following:

```
#PBS -S /bin/bash
```

```
#PBS -V
#PBS -N hdust                          # name fo the program
#PBS -l nodes=128,walltime=36:00:00    # number of processors, max time
#PBS -o output_${PBS_JOBID}            # log file
#PBS -e error_${PBS_JOBID}             # error file
#PBS -m e
#PBS -M desprh@gmail.com               # your email

# the file that is produced and contains all the input data
MASTERFILE=despo.inp
# the executable
HDUST=./hdustparv2.02.bc

#NO NEED DO MODIFY BEYOND THIS POINT
NSLOTS=`cat $PBS_NODEFILE | wc -l`

echo "---------------------------------"
echo "Running MPI HDUST on" $NSLOTS "cores"
echo "Executable: " $HDUST
echo "Master input file: " $MASTERFILE
echo "---------------------------------"

cd ${PBS_O_WORKDIR}

START=$(date +%s.%N)
mpirun -n $NSLOTS --mca btl_tcp_if_exclude ib1 -machinefile \
    $PBS_NODEFILE       $HDUST file=$MASTERFILE
END=$(date +%s.%N)
DIFF=$(echo "$END - $START" | bc)

MIN=`echo "$DIFF*0.0166667" | bc`

echo "--------------------"
echo "Finished MPI HDUST  "
echo Exectution time: $MIN minutes "($DIFF seconds)"
echo "--------------------"
```

The lines starting with `#PBS` are directives to the cluster (see relevant wiki).

The most important is the directive for the cluster resources requested.

```
#PBS -l nodes=128,walltime=36:00:00
```

With the above, we request the use of 128 nodes for 36 hours. This is important, because your submission script will be killed after 36 hours, so if your program keeps running, it will be interrupted. In general, the more the nodes the less the time, while you should better enter a longer time than you expect (otherwise, if the execution needs longer time than you had thought, you will need to resubmit with the time revised - to longer). But again, if you request a lot of nodes for a long time, your submission script might wait for longer until it finally starts, as the resources requested are high, and this means you will have those resources tied in time that other people might need them too.

To get an idea, a run of `hdust` with the use of 48 processors takes ~1-2 hours.

You might also run a serial program in the cluster. In this case you should set the number nodes to 1, while the time should not be shorter than needed, because the execution would be interrupted if the program still runs when the time requested has passed.

A rough estimation of the time to be requested, could be achived as follows. Let's say we request N_C processors. Check how long the program runs in your PC, say t_P. Assuming that the processors of the cluster and the processor in your PC are of similar capabilities, and also that the case that you want to run in the cluster is similar to the case that you run in your PC (having N_P threads) is needed in your personal computer. Then, the time to be requested should be of the order $t_C = t_P N_C / N_P$, provided that the main part of the code can be executed in parallel (e.g. as is the case when you calculate the trajectories of photons: the calculation of each trajectory can be executed in parallel with the calculation of every other trajectory - if each trajectory does not affect the other trajectories)

```
#PBS -l nodes=128           # number of nodes
#PBS -l ppn=2               # number of CPUs per node (total CPUs=nodes*ppn)
#PBS -l walltime=36:00:00   # max time
```

dummy