Royal Observatory
of Belgium

Running Cloudy in parallel

Peter van Hoof
Royal Observatory of Belgium

Cloudy Workshop
Chiang Mai, May 2018

Running Cloudy in parallel (1)

Royal Observatory of Belgium

There are two commands in Cloudy that are parallelized:

- The optimizer
- The grid command

Both commands generate lots of individual models and are therefore very easy to parallelize.

There are two methods in use for parallelization:

- Based on the fork() system call
- Based on the Message Passing Interface (MPI)

Both methods have advantages and disadvantages which will be discussed on the next slide. It is also possible to run both commands sequentially on a single CPU, but this can take a long time!

All these methods should give essentially identical results. If they don't, you can report that as a bug.

Royal Observatory of Belgium

Running Cloudy in parallel (2)

How do you choose which method of parallelization to use?

- The fork() method works out of the box. It will be automatically compiled into the code on all systems that support it in a default compilation. This includes Linux and other UNIX systems, Mac OS X, and Cygwin. So this is a hassle-free solution.
- The downside is that you can only fork new processes on a single machine, i.e. you are limited to a single computer or node.

So using the default fork()-based method of parallelization is ideal for running small/medium-sized grids (or optimizations with few free parameters) on your laptop or desktop or even a single compute server.

- The big advantage of MPI is that it allows you to use cores on multiple nodes of an HPC machine. This allows you to use a virtually unlimited number of cores (well, as many as the admins allow you to use...)
- The downside of MPI is that it needs external support scripts and libraries. The compilation and startup of MPI codes is not standardized and can differ from one system to another. You may need to load specific modules, or set search paths to find the executables and libraries.

Running Cloudy in parallel (3)

Royal Observatory of Belgium

So for very CPU-intensive applications (large grids or optimizations with lots of free parameters) it can be advantageous to use MPI on a suitable HPC cluster.

In a default compilation, Cloudy will run the optimizer and grid commands in parallel on systems that support the fork() system call and where the number of cores can be determined. By default it will use all threads (except on Mac OS X, where it will ignore hyperthreads). The number of threads to use can be changed on the **optimize phymir** and **grid** commands.

How do you choose the number of cores?

- Optimizer runs can use no more than 2xN cores simultaneously, where N is the number of parameters that are varied by the optimizer. So ideally you would use 2xN cores, but you could also use N cores, or ceil((2xN)/3), or some other small fraction of 2xN. This choice assures that the threads are well-balanced.
- Grid runs will compute N_1xN_2x... independent models, where N_1 is the number of grid points in the first variable, N_2 is the same for the second variable, etc. For good load balancing you can choose an integer divisor of this total number of models, but this is not strictly necessary.

Running Cloudy in parallel (4)

Royal Observatory of Belgium

To compile Cloudy for an MPI run requires several steps.

- MPI typically uses a wrapper around the compiler, called something like mpiCC, but alternative names are mpicxx or mpic++. You may need to load a module to find this.
- You need to figure out which compiler the script wraps around. You can do this by typing “mpiCC --version”. Typically this will show that it is either g++ or icc (the Intel compiler).
- You need to compile Cloudy in one of the sys_xxx subdirectories. If mpiCC wraps around g++, use the sys_mpi_gcc directory. If mpiCC wraps around icc, use the sys_mpi_icc directory. In these directories you can compile as you normally would.
- To run the MPI executable, you typically need to use the mpirun executable, i.e. “mpirun /path/to/cloudy.exe -r script”. A batch system will communicate to mpirun how many ranks to start on which nodes. Outside of a batch system, other methods are needed, like using the “-np” flag (for running on the local machine) or using a hostfile (for running on remote machines). See the man page for more info.