Calculating in Parallel - a simple tool

Arjen Markus

May 26, 2014

1 Introduction

The `CIP` tool (Calculating in Parallel) is meant to solve a common enough problem: use the computer's resources as efficiently as possible without too much work. More specifically, it is meant for this situation:

- You need to run a bunch of calculations with the same program, say various water quality scenarios.
- Each calculation takes a fairly long while, a couple of hours perhaps.
- The computational program does not use the entire computer, that is, it does not exploit its multiple processors to speed up the calculation.
- You do not want to sit around and wait for the calculations to be run one by one.

You could write a small batch file or script that simply starts the various calculations and let it run on. But that might clog up the computer. And if you use the cluster, you see yourself forced to use one node per calculation.

Well, enter `CIP`. It helps you manage the calculations by starting them in the background, but not more than there are processors around. When a calculation is finished, the next is started until there are no more calculations left. Management of these calculations is automatic, you just need to follow a few simple conventions.

This documentation describes what these conventions are and how `CIP` does its work. Hopefully it is a useful tool.

2 Setting up the calculations

`CIP` assumes that the files required as input for your calculations as well as the output files are generally stored in a single directory. So what it does is:

- Scan the subdirectories of the directory in which it was started

- If there is a file called `nocomp.cip` or `started.cip` in that subdirectory, it is skipped. Otherwise a calculation is started in that subdirectory. The first file indicates that this subdirectory is not meant for a calculation, the second file means a calculation has already been started (it may actually have finished).
- As soon as a calculation is to be started, a file `started.cip` is created to indicate this is the case. Then the calculation is started (see below), the output from the program or programs is stored in a file `output.cip` and when the calculation is finished, a file `done.cip` is created.

Since this procedure is done in parallel "N" times, your program is started "N" times, each in its own subdirectory and each assigned to its own processor (`CIP` does assume that the operating system is smart enough to do this without any outside help).

The various files are meant to help avoid racing conditions – you do not want to start the same calculation twice, certainly not at the same time – and to monitor the status of the calculation.

Here is an example of what a directory with several subdirectories might look like. The calculation is done via a batch file (or shell script) `runprogram.bat`, which calls `program1` and `program2`. The input consists of a file `database` in a subdirectory on its own and a specific file `myinput.inp` in each of the calculation subdirectories. The directory and its subdirectories would look something like this:

```
scenarios/
    fixed-data/
        database               (Database required by the programs)
        nocomp.cip             (Prevent computation here)
    scen1/
        myinput.inp            (Specific input for scenario "scen1")
    scen2/
        myinput.inp            (Specific input for scenario "scen2")
    scen3/
        myinput.inp            (Specific input for scenario "scen2")

    runprogram.bat             (Batch file to run the programs)
    program1.exe               (Computational program)
    program2.exe               (Postprocessing, creates reports)
```

The `runprogram.bat` batch file looks like this:

```
program1 %1
if exists output goto postprocessing
goto end
:postprocessing
program2 %1
:end
```

The batch file takes as argument the name of the input file – `myinput.inp` in this case. This name however is fixed, as `CIP` has no way to specify such arguments per calculation. If you need that, you can store a copy of the batch file in the calculation subdirectories.

Note: Since the calculations are done in various subdirectories, some care must be taken that the relevant programs, batch files and scripts can all be found. `CIP` tries to take care of all this by extending the `PATH` environment variable, but it does not know if arguments like `myinput.inp` refer to a file or not. You may need to help it by adding ".." or even the complete directory to such arguments.

3 Command-line arguments

`CIP` recognises the following arguments:

`-help`

Present an overview of the commands.

`-status`

Show which calculatons have already been started or done.

`-clear`

Clear up the working directories so that you can restart all the calculations. This command removes the files `started.cip`, `output.cip` and `done.cip`, but not the file `nocomp.cip` in each of the directories.

`-procs <N>`

Specify the number of simultaneous calculations (one per processor). If not, `CIP` will determine the likely number of useable processors with a simple heuristic method. This option allows you to control that number.

`-path <dir>`

Specify a directory to be added to the PATH environment variable. You can specify one directory per `-path` argument. Note: The working directory is prepended after processing all the arguments.

`-run`

This argument is used internally – see section 4. You should not use it yourself.

`--`

Indicates the end of the options. Anything after that is considered part of the command that is to be run. This is especially useful if your program requires arguments that might be mistaken for `CIP` options.

anything else

All other arguments are assumed to be part of the command that is to be run.

In the above example, running `CIP` can be done by:

```
cip -clean runprogram.bat myinput.inp
```

or if you want to redo the calculations (after a change in the program perhaps or a new database):

```
cip -clear runprogram.bat myinput.inp
```

With this command an individual calculation would be more or less equivalent to the following series of commands:

```
cd scen1
echo Started >started.cip
runprogram.bat myinput.inp >output.cip
echo Done >done.cip
```

4 Some technical details

To better understand what `CIP` is doing, here are a few details:

- The number of processors is determined on Windows via the `NUMBER_OF_PROCESSORS` environment variable. This is a simple, but not entirely reliable method. The alternatives I have been able to find give unexpected and completely useless results.

  On Linux it counts the number of lines that contain "processor :" in the pseudo-file `/proc/cpuinfo`.

  Whatever the method, hyperthreading will report a single hardware processor as two or more processors, whereas the effective number of processors might be less than that, as hyperthreading only makes a single processor appear to be several processors. There is no way of examining this that I know of. You might want to specify a lower number of processors to use to prevent slowdown of your computer.

- `CIP` clones itself "N" times. These clones use the special argument, `-run`, as an indication that they should actually run the program. This way the operating system does the actual parallel processing and each clone can simply wait for the computational program to finish before starting the next.

- The files `started.cip` are created as "atomically" as possible, so that race conditions are avoided as much as possible. It is thus quite unlikely that two clones are able to start a calculation in the same directory.

- If two or more computers have access to the same working directory, for instance because it resides on a network disk, then you can actually start `CIP` on each of these computers and gain even more efficiency.

- `CIP` comes in a Windows version and a Linux version.