Subject: Re: Multi-core techniques Posted by wita on Fri, 16 Apr 2010 15:28:15 GMT

View Forum Message <> Reply to Message

Dear Bernat,

There is actually an example included in the header of cgi\_process\_manager.pro, but I will try to explain this in more detail.

To start with, you need to to split your calculations into a set of independent tasks. Usually, this means splitting it by parameter ranges, by date, by region, by tile or whatever. These tasks need to be coded as an array of structures. So if you want to split your tasks over various inputfiles you could do something like this: tasks = Replicate({inputfile:""}, <number\_of\_input\_file>) tasks[0].inputfile="datafile1.dat" tasks[1].inputfile="datafile2.dat" etc.

Then you start "cgi\_process\_manager" and you provide "tasks" as parameter. The process manager will determine your machine setup and start as many bridges as it can find CPUs. Next it will start the task manager which keeps track of which tasks have been processed and which not. The idea behind it, is that you can separate the logic of handling tasks from the execution of tasks. For example, if you have an IDLDataminer license you could easily extend the taskmanager to read the tasks from a database table. I actually have a python implementation of such a task manager that I use a lot.

cgi\_process\_manager will now try to execute the tasks on the IDL\_IDLBridges that it has initialized. The first problem here is that

you can only send simple variables directly over an IDL\_IDLBridge (scalars, arrays), no structures or objects. The workaround I chose is

to first save the data in the task structure to a temporary .SAV file.

The name of this .SAV file is send over to the IDL\_IDLBridge and the procedure "cgi\_process\_client" is executed on the bridge. The cgi\_process\_client procedure restores the contents of the .SAV file and your parameters in the task will be available on the bridge as a variable "task".

The cgi\_process\_manager will continuously monitor the availability of the bridges and start new processes when a bridge comes available.

Finally, there is one pitfall here: how does the cgi\_process\_client

procedure know the name of the procedure/function it has to start? In fact, in the current implementation, it doesn't and you need to provide it yourself by hardcoding it into the cgi\_process\_client procedure. So in this example, line 70 of cgi\_process\_client.pro needs to be changed into something like:

my\_cpu\_intensive\_process, inputfile=task.inputfile

I may change this in the future, by coding the execution string into the task structure itself and use something like: Execute, task.execstring

## A few further remarks

- The process\_manager does not collect output from the client processes.

So your client process must contain logic to store its result.

- The approach with the .SAV seems a bit tricky but I found it to be quite robust. I also looked at the possibility to use shared memory between bridges but that is tricky as well. Moreover, in order to map the shared memory on the bridge you first need to know the layout
- of the structure, which you cannot send over the bridge as well. So you end up with a chicken-and-egg problem.
- Split your tasks in a couple of large chunks instead of using many small tasks. Executing each task involves a small overhead, but this

can become significant if you have to execute many tasks.

Hope this helps.

Allard