Subject: Multi-core techniques
Posted by Tim B on Fri, 16 Apr 2010 01:15:50 GMT
View Forum Message <> Reply to Message

I am working with various satellite datasets (e.g. Pathfinder SST)
where
most of the prior work has been producing 50km resolution analysis.
Even
an intensive piece of code over a full 30-odd year data set could
still be
finished overnight. The algorithms that I've seen don't excessively
use
array processing (no FFT's or such) and there is a lot of data
independence i.e.
values over the course of a few days are used rather than values over
a complete year.

With a move to higher resolution, i.e. 4km, there is significantly
more data (approx 150 4km
pixels in a 50km pixel). Given the data independence, my usual
approach would be to
create a thread pool around the number of available cores and
calculate each data
'piece' independently. However IDL doesn't expose threads to the
programmer. So I'd
value any thoughts about how to take advantage of multicore CPU's
(heck, even my laptop
is a dual core machine). My thoughts are:

- use C(?) to manage forking separate processes that start IDL and
pass parameters to the
  appropriate procedure to run in IDL

- run a number of IDL programs in parallel, the same code but
processing different
  temporal regions of the dataset. I can start up IDL in different
windows on an 8-core
  machine and each seems to be a separate process.

- use a different language/architecture completely :-)

I'd be interested to hear from anyone else trying to take advantage of
multicore CPU's..

Tim Burgess

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

Subject: Re: Multi-core techniques
Posted by natha on Fri, 16 Apr 2010 17:03:31 GMT
View Forum Message <> Reply to Message

Hi Allard,

Thank you for your explanation. It helped me to understand what are you doing in your code...
I started doing some tests and now I have a problem.

On my first execution I forget to destroy the idl_idlbridges and now if I try to create another bridge I get a "bus error" and I can't continue.

Your code is not bad and definitely it would be better if the cgi_process_client could know the procedure/function to start. Creating .SAV files to pass the parameters is a little bit tricky but

I can't imagine another way to do pass objects, structures, etc.. to the bridge.
Perhaps another feature to improve could be the possibility to collect output data. It's possible to use the GetVar from the IDL_IDLbridge and, actually, I don't how I would add this but it would be cool !

Thanks again Allard. Have fun !

nata
