Subject: Re: Speeding up multiple file reading
Posted by David Fanning on Thu, 02 Feb 2006 15:44:51 GMT
View Forum Message <> Reply to Message

clivecook59@gmail.com writes:

> I have a program where i need to read in multiple files. Currently i
> read in 6000 binary files using a function i have written. This reads
> three columns of data out of each file. To do this i use a loop that
> calls the function to read a file who's data is then added to an array.
> At the moment it takes around 90 seconds to go through the 6000 files.
> Is there any way that i can read this data not using a loop? Or at
> least are there any tips for speeding this up?

What does your code look like for reading the three columns
of data? This is likely to be the weakest link in your process.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Speeding up multiple file reading
Posted by Maarten[1] on Thu, 02 Feb 2006 15:48:00 GMT
View Forum Message <> Reply to Message

clivecook59@gmail.com wrote:

> I have a program where i need to read in multiple files. Currently i
> read in 6000 binary files using a function i have written. This reads
> three columns of data out of each file. To do this i use a loop that
> calls the function to read a file who's data is then added to an array.
> At the moment it takes around 90 seconds to go through the 6000 files.
> Is there any way that i can read this data not using a loop? Or at
> least are there any tips for speeding this up?

Is this data static (a look-up table, for instance?) In that case you
might want to read the data once, and write it back out to a format
that can more easily hold multiple arrays (say, hdf or netcdf). I think
you'll find that hdf is much faster, especially since you don't have to
append to an array, but create it once at the right size.

That said, if you can avoid resizing an array, and just create one

array of the final size and use that instead, I think you'll see some improvement.

How large are these files? 90 seconds for 6000 files doesn't sound too shabby, unless each file holds only 3 points. Part of the problem may be due to overhead of the filesystem itself, which goes avay by using a single file instead.

Maarten

Subject: Re: Speeding up multiple file reading
Posted by clivecook59 on Thu, 02 Feb 2006 16:08:28 GMT
View Forum Message <> Reply to Message

Hi Thanks,

I guess my question was was there a way to avoid loops. Previoulsy i have over come this problem by processing the data and saving it in the IDL save format. I then open the data into a program which produces a contour plot for analysing the data. This is considerably quicker however it requires running additional programs. I was hoping to combine it all into one program.

thanks

Clive

Subject: Re: Speeding up multiple file reading
Posted by David Fanning on Thu, 02 Feb 2006 16:15:02 GMT
View Forum Message <> Reply to Message

clivecook59@gmail.com writes:

> I guess my question was was there a way to avoid loops.

No, if you have to open 6000 files, you have to open 6000 files. No non-loopy way there. But there are fast loops, and there are slow loops. Since you seem reluctant to expose your code, I would guess SLOW loops. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Speeding up multiple file reading
Posted by clivecook59 on Thu, 02 Feb 2006 16:42:14 GMT
View Forum Message <> Reply to Message

The code for reading the binary files was not written by me and reads a proprietary data format. I don't think that there is much i can do with that code. However i do perform some operations on the data within the loop and have been trying to relocate them outside of the loop. An example of this is,

```
interp_height = 0 + INDGEN(int)*(16 - 0)/FLOAT(int - 1)

for i =0,count1 -1 do begin

data = read_binary_function(binary_file(i))

ch1x = data.ch1x ;1-D
ch2x = data.ch1x ;1-D
ch3x = data.ch1x ;1-D

correction = .....

sigheight(i,*) = height * correction

ch1(i,*)  = interpol(ch1x,sig_height(i,*),interp_height)
ch2(i,*)  = interpol(ch2x,sig_height(i,*),interp_height)
ch3(i,*)  = interpol(ch3x,sig_height(i,*),interp_height)

endfor
```

So (not sure if this is explained very well) i am using the interpol function to grid the data to a regaular grid governed by the interp_height. I have tried to remove these interpol steps from the loop but with no luck,

```
ch1  = interpol(ch1x,sig_height,interp_height)
```

In this case ch1x,sig_height and interp_height have the same dimensions but does not produce the same results as in the loop. I use rebin to produce a new 2-D array with the same dimension for interp_height as ch1x and sig_height, (transpose(rebin(interp_height,400,100))).

I hope this is clear.

thanks

Clive

Subject: Re: Speeding up multiple file reading
Posted by David Fanning on Thu, 02 Feb 2006 16:50:53 GMT
View Forum Message <> Reply to Message

clivecook59@gmail.com writes:

> I hope this is clear.

Well, it is clear a WHOLE lot more than just reading the
data is going on! I'd say 90 seconds is probably a reasonable
amount of time, under the (previously unannounced) circumstances.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Speeding up multiple file reading
Posted by JD Smith on Thu, 02 Feb 2006 16:56:52 GMT
View Forum Message <> Reply to Message

On Thu, 02 Feb 2006 08:44:51 -0700, David Fanning wrote:

> clivecook59@gmail.com writes:
>
>> I have a program where i need to read in multiple files. Currently i
>> read in 6000 binary files using a function i have written. This reads
>> three columns of data out of each file. To do this i use a loop that
>> calls the function to read a file who's data is then added to an array.
>> At the moment it takes around 90 seconds to go through the 6000 files.
>> Is there any way that i can read this data not using a loop? Or at
>> least are there any tips for speeding this up?
>
> What does your code look like for reading the three columns
> of data? This is likely to be the weakest link in your process.

When it comes to understanding why a given bit of IDL code is so slow, there really is no need to speculate, given how *easy* using the IDL profiling tool is. Just make sure your routines of interest are compiled (typically just by running the operation once), and type:

```
IDL> profiler,/system & profiler
IDL> run_my_code
IDL> profiler,/report
```

and you get instant access to a nice table summarizing the number of times each routine (user or system) is called, how long that took per call and in total, etc. Then work a bit on the slowest function(s), re-compile, type

```
IDL> profiler,/reset
```

to clear the last report, run the slow operation again, and get another report, checking whether your changes improved things.

Obviously, operations like a=[a,findgen(1000)] don't get a separate line item in the PROFILER report, but it's usually easy to figure out their impact by examining the "Only" column of the report, which tallies the time spent only in the body of a routine, not in routines it calls. So if you have a routine like

```
pro my_routine
  a=do_something(FOO=foo)
  b=readu(a,foo)
  ...
  for i=0,n_elements(foo)-1 do b=[b,randomu(sd,1000)]
end
```

it's fairly easy to figure out what that repetitive array concatenation is doing to you by looking at the "Only" column for MY_ROUTINE. If there is too much "body" in a routine to easily tell where the problem spots are using this method, consider breaking the body up into some temporary routines which PROFILER can track.

I've just begun using the PROFILER regularly, and really appreciate how useful it is. For instance, if you have a complex GUI operation which is slow, running in a widget in the background, just invoke profiler, run the slow operation, generate the report, find and fix the slow parts, recompile, and repeat -- all without ever quitting the running widget program! Try that in other languages.

Note that unfortunately the docs for PROFILER are a bit difficult to parse. It took me forever to understand that:

IDL> profiler

adds all compiled user routines to the list of things to profile,
while:

IDL> profiler,/SYSTEM

adds only system routines, and nothing else. So, if you want *both*
system routines and user routines to be profiled, you need two calls
to PROFILER. This is also true of clearing the profiling, i.e., to
totally shut down all profiling, you need:

IDL> profiler,/CLEAR,/SYSTEM & profiler,/CLEAR,/RESET

to do the job properly. That is, "profiler,/SYSTEM" and "profiler"
function like two completely separate tools, except for /RESET, which
resets everything. Not exactly intuitive, but I can forgive them that
for providing us such a quasi-miraculous profiling tool.

JD

Subject: Re: Speeding up multiple file reading
Posted by news.qwest.net on Thu, 02 Feb 2006 17:51:27 GMT
View Forum Message <> Reply to Message

<clivecook59@gmail.com> wrote in message
 news:1138898534.745134.104580@g44g2000cwa.googlegroups.com.. .
> interp_height = 0 + INDGEN(int)*(16 - 0)/FLOAT(int - 1)
>
> for i =0,count1 -1 do begin
>
> data = read_binary_function(binary_file(i))
>
> ch1x = data.ch1x ;1-D
> ch2x = data.ch1x ;1-D
> ch3x = data.ch1x ;1-D
>
> correction = ...
>
> sigheight(i,*) = height * correction
>
> ch1(i,*) = interpol(ch1x,sig_height(i,*),interp_height)
> ch2(i,*) = interpol(ch2x,sig_height(i,*),interp_height)
> ch3(i,*) = interpol(ch3x,sig_height(i,*),interp_height)
>
> endfor

This looks find to me. Perhaps your 90 seconds is a reasonable amount of time. You could try to see how long each step takes i.e. is most time spent in your "read_binary_function()" call? If so, probably not much you can do about it.

One optimization may be as follows;

```
tempsigheight = height * correction
ch1(i,*)  = interpol(ch1x,tempsigheight,interp_height)
ch2(i,*)  = interpol(ch2x,tempsigheight,interp_height)
ch3(i,*)  = interpol(ch3x,tempsigheight,interp_height)
sigheight(i,*) = tempsigheight
```

Then you are not selecting a subarray in each of the 3 function calls, (in total, 3*6000 function calls) but that would only be a minor effect.

Cheers,
bob

Subject: Re: Speeding up multiple file reading
Posted by Paul Van Delst[1] on Thu, 02 Feb 2006 17:53:22 GMT
View Forum Message <> Reply to Message

clivecook59@gmail.com wrote:

```
>  The code for reading the binary files was not written by me and reads a
>  proprietary data format.  I don't think that there is much i can do
>  with that code. However i do perform some operations on the data within
>  the loop and have been trying to relocate them outside of the loop. An
>  example of this is,
>
>  interp_height = 0 + INDGEN(int)*(16 - 0)/FLOAT(int - 1)
>
>  for i =0,count1 -1 do begin
>
>  data = read_binary_function(binary_file(i))
>
>  ch1x = data.ch1x ;1-D
>  ch2x = data.ch1x ;1-D
>  ch3x = data.ch1x ;1-D
>
>  correction = ......
>
>  sigheight(i,*) = height * correction
>
>  ch1(i,*)  = interpol(ch1x,sig_height(i,*),interp_height)
```

> ch2(i,*) = interpol(ch2x,sig_height(i,*),interp_height)
> ch3(i,*) = interpol(ch3x,sig_height(i,*),interp_height)

It's probably a small gain, but you might think about changing the order of your arrays from
(i,*)
to
(*,i)
IIRC, IDL accesses array elements like Fortran and the latter form is contigous memory.

Also, it can be a pain in the rear, but if you can replace the INTERPOL function with the INTERPOLATE function, things can go a lot faster. It's really only of benefit if the interpolation indices are all the same, and it looks like they are in your case. You reduce your

```
ch1(i,*) = interpol(ch1x,sig_height(i,*),interp_height)
ch2(i,*) = interpol(ch2x,sig_height(i,*),interp_height)
ch3(i,*) = interpol(ch3x,sig_height(i,*),interp_height)
```

to

```
...compute interpolation_index array once....
ch1(i,*) = interpolate(ch1x,interpolation_index)
ch2(i,*) = interpolate(ch2x,interpolation_index)
ch3(i,*) = interpolate(ch3x,interpolation_index)
```

Computing the interpolation index array is /very/ easy if sig_height(i,*) is regularly spaced. If it's not, it's a bit more complicated, but using INTERPOLATE still means you only have to do it once.

And, you may even be able to use INTERPOLATE outside the loop since you are doing linear interp and you can supply both X and Y interpolation indices (where the latter would just be the 0,1,2,3,4,... of your "i" dimension). Not sure about that though.

This is all of the top of my head, so of course check the manual and test.

paulv

>
> endfor
>
> So (not sure if this is explained very well) i am using the interpol
> function to grid the data to a regaular grid governed by the
> interp_height. I have tried to remove these interpol steps from the
> loop but with no luck,
>
> ch1 = interpol(ch1x,sig_height,interp_height)

>
> In this case ch1x,sig_height and interp_height have the same dimensions
> but does not produce the same results as in the loop. I use rebin to
> produce a new 2-D array with the same dimension for interp_height as
> ch1x and sig_height, (transpose(rebin(interp_height,400,100))).
>
> I hope this is clear.
>
> thanks
>
> Clive
>

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
