Subject: Re: Memory management when concatenating arrays
Posted by Yngvar Larsen on Wed, 28 Oct 2015 14:44:04 GMT
View Forum Message <> Reply to Message

I ran the following script:

```
nx = 360
ny = 180
nt = 1000
N = 10
data = fltarr(nx,ny,nt)

print, 'Concatenation:'
tic
all_data = []
for ii=0, N-1 do all_data = [[[all_data]],[[data]]]
toc
print, '**************'
print, 'Preallocation with zero initialization:'
tic
all_data = fltarr(nx,ny,nt*10)
for ii=0, N-1 do all_data[0,0,ii] = data
toc

print, '**************'
print, 'Preallocation without zero initialization'
tic
all_data = fltarr(nx,ny,nt*10, /NOZERO)
for ii=0, N-1 do all_data[0,0,ii] = data
toc
```

I get the following:

```
IDL> .r test
% Compiled module: $MAIN$.
Concatenation:
% Time elapsed: 6.3571460 seconds.
**************
Preallocation with zero initialization:
% Time elapsed: 1.5204742 seconds.
**************
Preallocation without zero initialization
% Time elapsed: 0.62908983 seconds.
```

My script excludes the I/O part which should be the same for all three versions.

Bottom line: I think preallocating your array with the /NOZERO flag set is your best option for the scenario you describe.
Of course, as has already been commented in the thread, you need to make sure that your data fit in memory. Your example is 2.4GB in single precision float, and 4.8GB in double precision. And even if this fits in memory, you will likely want to do operations on this big array, and then you will quickly run out of memory. On my 8GB laptop, the following would be enough to run out of memory:

all_data = dblarr(360,180,10000)
all_data_scaled = all_data*!dpi

On Wednesday, 28 October 2015 14:49:38 UTC+1, rj...@le.ac.uk wrote:
> I have a large multi-dimensional array that is split across several files by time.
>
> i.e. file1 contains the first 1000 timesteps, [360,180,1000], file2 contains the next 1000 timesteps [360,180,1000], etc.
>
> What I want to end up with is one big array that's read say all 10 files in and is (360, 180, 10000).
>
> What I'm doing is this in a loop:
>
> all_data=[[[all_data]], [[data]]]
>
> But I quickly run out of memory trying to concatenate in this way.
>
> I tried using temporary
>
> all_data=[[[temporary(all_data)]], [[data]]]
>
> but this doesn't help.
>
> Is there an efficient way of doing this?
>
> Cheers