Subject: Memory management when concatenating arrays
Posted by rjp23 on Wed, 28 Oct 2015 13:49:35 GMT
View Forum Message <> Reply to Message

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

Subject: Re: Memory management when concatenating arrays
Posted by Alain Kattnig on Wed, 28 Oct 2015 13:58:27 GMT
View Forum Message <> Reply to Message

Le mercredi 28 octobre 2015 14:49:38 UTC+1, rj...@le.ac.uk a écrit :
> 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

Use ASSOC, it associates a variable to a file, thus allowing you to address unlimited siezd variables

---

Subject: Re: Memory management when concatenating arrays
Posted by greg.addr on Wed, 28 Oct 2015 14:16:49 GMT
View Forum Message <> Reply to Message

While your method requires creating a new copy of the data-so-far at each concatenation, the problem is more likely that you just can't hold the entire dataset in memory. If you want to create a single file containing all the data, read each file in turn and write it out to a concatenated file. If you can read the whole file at the end - fine. If not, you have to think about whether you can process it serially.

Greg

Subject: Re: Memory management when concatenating arrays
Posted by Phillip Bitzer on Wed, 28 Oct 2015 14:41:55 GMT
View Forum Message <> Reply to Message

On Wednesday, October 28, 2015 at 8:49:38 AM UTC-5, rj...@le.ac.uk wrote:
> I have a large multi-dimensional array that is split across several files by time.
>
> Is there an efficient way of doing this?
>
> Cheers

This is a great job for a list, assuming you enough enough memory to create an array of this size.

```
l=  LIST()

FOR iFile=0, nFiles-1 DO
  data = READ_IN_DATA_FROM_FILE(iFile)
  l->Add, data
END

;finished reading in files? Convert the list to an array
```

all_data = l->ToArray(DIM=3)

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

Subject: Re: Memory management when concatenating arrays
Posted by Yngvar Larsen on Wed, 28 Oct 2015 14:54:43 GMT
View Forum Message <> Reply to Message

On Wednesday, 28 October 2015 15:44:05 UTC+1, Yngvar Larsen wrote:
> I ran the following script:

Bug fix for the last two cases follows. Still same conclusion, but

```
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*nt] = 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*nt] = data
toc
```

```
Concatenation:
Process took 6.524 seconds
**************
Preallocation with zero initialization:
Process took 1.506 seconds
**************
Preallocation without zero initialization
Process took 1.244 seconds
```

I also ran with double precision arrays, and got this:

```
Concatenation:
Process took 21.428 seconds
**************
Preallocation with zero initialization:
Process took 5.366 seconds
**************
Preallocation without zero initialization
Process took 2.953 seconds
```

--
Yngvar