Subject: array concatenation and optimization
Posted by Sean Raffuse on Wed, 26 Sep 2001 18:18:37 GMT
View Forum Message <> Reply to Message

Hello.

I am trying to read a bunch of data from a file to a structure array. I'm not sure many data entries the file will have until I have read it and so I am increasing the size of the structure array after reading each line. I do this by concatenating.

adp_struct_single is the structure as a "scalar"
adp_struct is the array

I concatenate like so:
adp_struct =[adp_struct, adp_struct_single]

This is working but it has increased the processing time of my loop by an order of magnitude. Is there a better way to do this? Is there a reason this is so slow?

Thanks in advance.

-Sean Raffuse

Subject: Re: array concatenation and optimization
Posted by Mark Hadfield on Thu, 27 Sep 2001 22:25:03 GMT
View Forum Message <> Reply to Message

From: "Craig Markwardt" <craigmnet@cow.physics.wisc.edu>
> ...
> My pet favorite is to read the file line by line, but grow the array
> in chunks. I usually grow it by powers of two until a certain limit.
> Example (not tested),

I built essentially the same logic into my MGH_Vector class, see

http://katipo.niwa.cri.nz/~hadfield/gust/software/idl/mgh_ve ctor__define.pro

The data are stored in a pointer array which is initialised with spare capacity. Elements can be added one at a time; every time the capacity of the array is reached it is extended (which means it is replaced by a larger one). After some trial & error I set the initial size to 1000 & the resizing algorithm to

new_size = round(1.5*old_size) > (new_size+1000)

The advantage of doing this inside an object, of course, is that all the details can be hidden and forgotten about.

Performance is acceptable: creating the object, adding 10^6 items (5-char strings), retrieving them all and then destroying the object takes 20 s (Pentium III 800). This compares with about 3 s to do the same operations with a plain string array, using the same logic to extend the array when necessary. Either way, the time varies more-or-less linearly with the number of items to be processed.

The timing code is in

http://katipo.niwa.cri.nz/~hadfield/gust/software/idl/mgh_ex ample_container.pro

---
Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield
National Institute for Water and Atmospheric Research

--

Subject: Re: array concatenation and optimization
Posted by Stein Vidar Hagfors H[1] on Fri, 28 Sep 2001 13:56:59 GMT
View Forum Message <> Reply to Message

"Mark Hadfield" <m.hadfield@niwa.cri.nz> writes:

> From: "Craig Markwardt" <craigmnet@cow.physics.wisc.edu>
>> ...
>> My pet favorite is to read the file line by line, but grow the array
>> in chunks. I usually grow it by powers of two until a certain limit.
>> Example (not tested),
>
> I built essentially the same logic into my MGH_Vector class, see
>
> http://katipo.niwa.cri.nz/~hadfield/gust/software/idl/mgh_ve ctor__define.pro
>
> The data are stored in a pointer array which is initialised with spare
> capacity. Elements can be added one at a time; every time the capacity of
> the array is reached it is extended (which means it is replaced by a larger
> one). After some trial & error I set the initial size to 1000 & the resizing
> algorithm to

>   new_size = round(1.5*old_size) > (new_size+1000)
>
> The advantage of doing this inside an object, of course, is that all the
> details can be hidden and forgotten about.
>
> Performance is acceptable: creating the object, adding 10^6 items (5-char
> strings), retrieving them all and then destroying the object takes 20 s
> (Pentium III 800). This compares with about 3 s to do the same operations
> with a plain string array, using the same logic to extend the array when
> necessary. Either way, the time varies more-or-less linearly with the number
> of items to be processed.

If you're taking the trouble of hiding it all inside an object, why
not go further to use e.g. lists, dropping the need for replacing
anything until possibly after as part of a "reconstitution" operation
(would need the user program to signal when building is finished - or
possibly trigger it automatically when a first read access is made ?)

--
---------- ----------
Stein Vidar Hagfors Haugan
ESA SOHO SOC/European Space Agency Science Operations Coordinator for SOHO

NASA Goddard Space Flight Center, Email: shaugan@esa.nascom.nasa.gov
Mail Code 682.3, Bld. 26, Room G-1, Tel.: 1-301-286-9028/240-354-6066
Greenbelt, Maryland 20771, USA. Fax: 1-301-286-0264
---------- ----------

Subject: Self-sizing arrays (was array concat and opt)
Posted by ngls on Thu, 04 Oct 2001 14:22:42 GMT
View Forum Message <> Reply to Message

shaugan@esa.nascom.nasa.gov (Stein Vidar Hagfors Haugan) wrote in
<xmzsnd7ju50.fsf@esa.nascom.nasa.gov>:

> If you're taking the trouble of hiding it all inside an object, why
> not go further to use e.g. lists, dropping the need for replacing
> anything until possibly after as part of a "reconstitution" operation
> (would need the user program to signal when building is finished - or
> possibly trigger it automatically when a first read access is made ?)
>

I've also written a "self sizing array" (Vector) class very similar to
Mark's.

(For those not in the know, C++ and Java - at least - both offer self
sizing array classes (objects) called Vectors - hence the use of the term.

My class is similarly called c_vector. This does not mean the arrays must be 1-dimensional, as the name might suggest!)

To answer Stein's question, I did consider writing it using lists, but to me it was important to be able to access array slices of the data stored in the 'vector'. For this reason I store all the data in a single array (with an arbitrary number of dimensions). Whilst access to single array elements causes no change to the internal array size, if the user requests all the data (either a pointer or copy of the whole array) it is trimmed to size. This allows the user to get slices of the array. It also works with arrays of structures.

To answer the orginal question (but not with structures) you could do something like this:

```
PRO test_vector

 file = '5_cols_test.txt'
 size_guess = 500
 capacity_incr = 101

 line = FLTARR(5) ;Each line in file contains 5 floats
 data = OBJ_NEW('c_vector', line, size_guess, capacity_incr)

 OPENR, lun, file, /GET_LUN

 WHILE NOT(EOF(lun)) DO BEGIN
  READF, lun, line
  data -> ADD_ELEMENT, line
 ENDWHILE

 FREE_LUN, lun ;Free lun and close

 PRINT, 'Numbers of lines read in:', data -> GET_SIZE() ;Displays 1000
 r = data -> GET_DATA_REF()
 PRINT, 'Dimensions of data array:', SIZE(*r, /DIM) ;Displays 5 1000
 PRINT, 'Mean of third column:', MEAN( (*r)[2,*] ) ;Displays 0.504

 OBJ_DESTROY, data

END
```

If anyone is interested I can post the c_vector code...

Justin