Subject: Merits of different ways of 'extending' arrays
Posted by Andy Sayer on Thu, 29 Aug 2013 15:44:51 GMT
View Forum Message <> Reply to Message

Hi all,

I am writing some code where I am loading a whole bunch of files one by one, querying them from valid data, and putting valid data from each file into an array (for later use). I don't know ahead of time how many files there will be, or how many valid data points there will be in a file.

The way I have written my code so far is like this:

```
var_1_arr=[!values.f_nan]
var_2_arr=[!values.f_nan]
var_3_arr=[!values.f_nan]

f=file_search( [path and identifier to files],count=nfiles)

for i=0l,nfiles-1 do begin

  [load contents of file f[i] into a structure]

  is_valid=where(blah blah,n_valid)

  if n_valid gt 0 then begin
    var_1_arr=[var_1_arr,f.var_1[is_valid]]
    var_2_arr=[var_2_arr,f.var_2[is_valid]]
    var_3_arr=[var_3_arr,f.var_3[is_valid]]

  endif

endfor
```

So, hopefully you get the idea. I only have a small subset of the test data to work with at the moment (the rest is a few months off).

It occurs to me that I could code it something like this:

```
max_points=1.e7

var_1_arr=fltarr(max_points)
var_1_arr(*)=!values.f_nan
var_2_arr=var_1_arr
var_3_arr=var_1_arr

f=file_search( [path and identifier to files],count=nfiles)

ctr=0l
```

```
for i=0l,nfiles-1 do begin

  [load contents of file f[i] into a structure]

  is_valid=where(blah blah,n_valid)

  if n_valid gt 0 then begin
    var_1_arr[ctr:ctr+n_valid]=f.var_1[is_valid]
    var_2_arr[ctr:ctr+n_valid]=f.var_2[is_valid]
    var_3_arr[ctr:ctr+n_valid]=f.var_3[is_valid]
    ctr=ctr+n_valid
  endif

endfor
```

This has the drawback that I have to know in advance the maximum number of data points I could have (but I can set max_points to some arbitrary high number to be safe). Does anyone know whether any one method is better/less memory-intensive than the other, when it comes to largeish data volumes (tens of millions of points)? I only have a few percent of the final data so far, so am interested in the likely merits of each method. Google didn't help but perhaps I was using the wrong search keywords.

In case relevant, this is IDL 7.1.1. or 8.2.2.

Thanks,

Andy

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Andy Sayer on Thu, 29 Aug 2013 15:57:09 GMT
View Forum Message <> Reply to Message

I always find typos after I click the button to post. :) The second code snippet should probably be [ctr:ctr_n_valid-1] rather than [ctr:ctr_n_valid]. And also, by 'better/less memory-intensive' I really mean 'faster/less memory-intensive'.

On Thursday, August 29, 2013 11:44:51 AM UTC-4, AMS wrote:
> Hi all,
>
>
>
> I am writing some code where I am loading a whole bunch of files one by one, querying them from valid data, and putting valid data from each file into an array (for later use). I don't know ahead of time how many files there will be, or how many valid data points there will be in a file.
>
>

>
> The way I have written my code so far is like this:
>
>
>
> var_1_arr=[!values.f_nan]
>
> var_2_arr=[!values.f_nan]
>
> var_3_arr=[!values.f_nan]
>
>
>
> f=file_search([path and identifier to files],count=nfiles)
>
>
>
> for i=0l,nfiles-1 do begin
>
>
>
> [load contents of file f[i] into a structure]
>
>
>
> is_valid=where(blah blah,n_valid)
>
>
>
> if n_valid gt 0 then begin
>
> var_1_arr=[var_1_arr,f.var_1[is_valid]]
>
> var_2_arr=[var_2_arr,f.var_2[is_valid]]
>
> var_3_arr=[var_3_arr,f.var_3[is_valid]]
>
>
>
> endif
>
>
>
> endfor
>
>
>
> So, hopefully you get the idea. I only have a small subset of the test data to work with at the

moment (the rest is a few months off).
>
>
>
> It occurs to me that I could code it something like this:
>
>
>
> max_points=1.e7
>
>
>
> var_1_arr=fltarr(max_points)
>
> var_1_arr(*)=!values.f_nan
>
> var_2_arr=var_1_arr
>
> var_3_arr=var_1_arr
>
>
>
> f=file_search([path and identifier to files],count=nfiles)
>
>
>
> ctr=0l
>
>
>
> for i=0l,nfiles-1 do begin
>
>
>
> [load contents of file f[i] into a structure]
>
>
>
> is_valid=where(blah blah,n_valid)
>
>
>
> if n_valid gt 0 then begin
>
> var_1_arr[ctr:ctr+n_valid]=f.var_1[is_valid]
>
> var_2_arr[ctr:ctr+n_valid]=f.var_2[is_valid]
>

> var_3_arr[ctr:ctr+n_valid]=f.var_3[is_valid]
>
> ctr=ctr+n_valid
>
> endif
>
>
>
> endfor
>
>
>
> This has the drawback that I have to know in advance the maximum number of data points I could have (but I can set max_points to some arbitrary high number to be safe). Does anyone know whether any one method is better/less memory-intensive than the other, when it comes to largeish data volumes (tens of millions of points)? I only have a few percent of the final data so far, so am interested in the likely merits of each method. Google didn't help but perhaps I was using the wrong search keywords.
>
>
>
> In case relevant, this is IDL 7.1.1. or 8.2.2.
>
>
>
> Thanks,
>
>
>
> Andy

Subject: Re: Merits of different ways of 'extending' arrays
Posted by David Fanning on Thu, 29 Aug 2013 16:29:43 GMT
View Forum Message <> Reply to Message

AMS writes:

> This has the drawback that I have to know in advance the maximum number of data points I could have (but I can set max_points to some arbitrary high number to be safe). Does anyone know whether any one method is better/less memory-intensive than the other, when it comes to largeish data volumes (tens of millions of points)? I only have a few percent of the final data so far, so am interested in the likely merits of each method. Google didn't help but perhaps I was using
the wrong search keywords.

You are MUCH better off to allocate memory in large chucks and then trim
or add to your arrays (in more large chunks) as necessary. This will

keep you from fragmenting your memory space, which is the single biggest
problem when working with large arrays.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Andy Sayer on Thu, 29 Aug 2013 16:59:51 GMT
View Forum Message <> Reply to Message

Thanks; happily, this was a simple recode to make. :)

Andy

On Thursday, August 29, 2013 12:29:43 PM UTC-4, David Fanning wrote:
> AMS writes:
>
>
>
>> This has the drawback that I have to know in advance the maximum number of data points I could have (but I can set max_points to some arbitrary high number to be safe). Does anyone know whether any one method is better/less memory-intensive than the other, when it comes to largeish data volumes (tens of millions of points)? I only have a few percent of the final data so far, so am interested in the likely merits of each method. Google didn't help but perhaps I was using
>
> the wrong search keywords.
>
>
>
> You are MUCH better off to allocate memory in large chucks and then trim
>
> or add to your arrays (in more large chunks) as necessary. This will
>
> keep you from fragmenting your memory space, which is the single biggest
>
> problem when working with large arrays.
>

>
>
> Cheers,
>
>
>
> David
>
>
>
>
>
>
>
>
> --
>
> David Fanning, Ph.D.
>
> Fanning Software Consulting, Inc.
>
> Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
>
> Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Merits of different ways of 'extending' arrays
Posted by chris_torrence@NOSPAM on Thu, 29 Aug 2013 17:11:25 GMT
View Forum Message <> Reply to Message

Actually, if you use lists, then you can add each individual chunk of data to each list, and then use ToArray() with the DIMENSION keyword. For example:

```
l = list(findgen(20))
l.add, findgen(20) + 20
help, l.ToArray(DIM=1)
<Expression>    FLOAT     = Array[40]
```

This will be both the fastest way and will use the least memory.

Cheers,
Chris
ExelisVIS

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Andy Sayer on Thu, 29 Aug 2013 17:13:12 GMT
View Forum Message <> Reply to Message

Thanks; some of our machines are on IDL 7.1.1. though so I don't think we can use lists for code portability. :)

Andy

On Thursday, August 29, 2013 1:11:25 PM UTC-4, Chris Torrence wrote:
> Actually, if you use lists, then you can add each individual chunk of data to each list, and then use ToArray() with the DIMENSION keyword. For example:
>
>
>
> l = list(findgen(20))
>
> l.add, findgen(20) + 20
>
> help, l.ToArray(DIM=1)
>
> <Expression> FLOAT = Array[40]
>
>
>
> This will be both the fastest way and will use the least memory.
>
>
>
> Cheers,
>
> Chris
>
> ExelisVIS

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Fabzi on Fri, 30 Aug 2013 09:54:23 GMT
View Forum Message <> Reply to Message

On 08/29/2013 07:13 PM, AMS wrote:
> Thanks; some of our machines are on IDL 7.1.1. though so
> I don't think we can use lists for code portability.:)
>
> Andy

for versions before 8. you have the List from Michael Galloy also, which
is very fast:

http://docs.idldev.com/idllib/collection/dir-overview.html

Cheers

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Michael Galloy on Fri, 30 Aug 2013 21:08:25 GMT
View Forum Message <> Reply to Message

On 8/30/13 3:54 AM, Fabien wrote:
> On 08/29/2013 07:13 PM, AMS wrote:
>> Thanks; some of our machines are on IDL 7.1.1. though so
>> I don't think we can use lists for code portability.:)
>>
>> Andy
>
> for versions before 8. you have the List from Michael Galloy also, which
> is very fast:
>
> http://docs.idldev.com/idllib/collection/dir-overview.html
>
> Cheers

MGcoArrayList implements the block strategy that David was talking about. Set the BLOCK_SIZE keyword to the size of chunks you want to allocate.

Also, my library has moved to GitHub, so you should go there to make sure you always get the most recent versions:

https://github.com/mgalloy/mglib

It's in src/collection and you need several other of the files in the collection directory.

Mike
--
Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)
Research Mathematician
Tech-X Corporation

Subject: Re: Merits of different ways of 'extending' arrays
Posted by suicidaleggroll on Mon, 09 Sep 2013 22:41:10 GMT
View Forum Message <> Reply to Message

Another option is to set up a pointer array nfiles long before the loop, inside the loop load the file and find the valid points, then put that array into that file's pointer, while incrementing a counter to keep track of the total number of points. When you're done, you have all of your data saved in pointers (one per file), and a count of the total number of valid points. Then you allocate your array, loop back through the elements of the pointer array, and fill the array as necessary.

Something like:

```
f = file_search(path, count=nfiles)
ptrs = ptrarr(nfiles)
num = 0l
for i=0l,nfiles-1 do begin
  ;; load contents of file
  is_valid = where(stuff, n_valid)
  if n_valid gt 0 then begin
     num += n_valid
     ptrs[i] = ptr_new(f.var_1[is_valid])
  endif
endfor

data = fltarr(num)
idx = 0l
for i=0l,nfiles-1 do begin
   if ptr_valid(ptrs[i]) then begin
      num = n_elements(*ptrs[i])
      data[idx:idx+num-1] = *ptrs[i]
      ptr_free, ptrs[i]
   endif
endfor
```

Subject: Re: Merits of different ways of 'extending' arrays
Posted by suicidaleggroll on Tue, 10 Sep 2013 20:08:40 GMT
View Forum Message <> Reply to Message

On Monday, September 9, 2013 4:41:10 PM UTC-6, suicida...@gmail.com wrote:
> Another option is to set up a pointer array nfiles long before the loop, inside the loop load the file and find the valid points, then put that array into that file's pointer, while incrementing a counter to keep track of the total number of points. When you're done, you have all of your data saved in pointers (one per file), and a count of the total number of valid points. Then you allocate your array, loop back through the elements of the pointer array, and fill the array as necessary. Something like:
>
>
>
> f = file_search(path, count=nfiles)
>
> ptrs = ptrarr(nfiles)
>
> num = 0l
>
> for i=0l,nfiles-1 do begin
>
> ;; load contents of file

```
>
>   is_valid = where(stuff, n_valid)
>
>   if n_valid gt 0 then begin
>
>     num += n_valid
>
>     ptrs[i] = ptr_new(f.var_1[is_valid])
>
>   endif
>
> endfor
>
>
>
> data = fltarr(num)
>
> idx = 0l
>
> for i=0l,nfiles-1 do begin
>
>   if ptr_valid(ptrs[i]) then begin
>
>     num = n_elements(*ptrs[i])
>
>     data[idx:idx+num-1] = *ptrs[i]
>
>     ptr_free, ptrs[i]
>
>   endif
>
> endfor
```

Wish I could edit my post...

There should be an "idx += num" next to the ptr_free at the end of the second loop.

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Andy Sayer on Sun, 15 Sep 2013 01:11:07 GMT
View Forum Message <> Reply to Message

Thanks for the continuing tips!

The first suggestion (allocate a 'big enough' array up-front, rather than continually extend) worked great for my purposes, so that's what I stuck with, given that it was also very simple. Although I appreciate the continued suggestions.

Andy

On Tuesday, September 10, 2013 4:08:40 PM UTC-4, suicida...@gmail.com wrote:
> On Monday, September 9, 2013 4:41:10 PM UTC-6, suicida...@gmail.com wrote:
>
>> Another option is to set up a pointer array nfiles long before the loop, inside the loop load the file and find the valid points, then put that array into that file's pointer, while incrementing a counter to keep track of the total number of points. When you're done, you have all of your data saved in pointers (one per file), and a count of the total number of valid points. Then you allocate your array, loop back through the elements of the pointer array, and fill the array as necessary. Something like:
>
>>
>
>>
>
>>
>
>> f = file_search(path, count=nfiles)
>
>>
>
>> ptrs = ptrarr(nfiles)
>
>>
>
>> num = 0l
>
>>
>
>> for i=0l,nfiles-1 do begin
>
>>
>
>> ;; load contents of file
>
>>
>
>> is_valid = where(stuff, n_valid)
>
>>
>
>> if n_valid gt 0 then begin
>
>>
>
>> num += n_valid
>

```
>>
>
>>       ptrs[i] = ptr_new(f.var_1[is_valid])
>
>>
>
>>    endif
>
>>
>
>>  endfor
>
>>
>
>>
>
>>
>
>>
>
>>  data = fltarr(num)
>
>>
>
>>  idx = 0l
>
>>
>
>>  for i=0l,nfiles-1 do begin
>
>>
>
>>     if ptr_valid(ptrs[i]) then begin
>
>>
>
>>        num = n_elements(*ptrs[i])
>
>>
>
>>        data[idx:idx+num-1] = *ptrs[i]
>
>>
>
>>        ptr_free, ptrs[i]
>
>>
>
>>     endif
>
```

>>
>
>> endfor
>
>
>
> Wish I could edit my post...
>
>
>
> There should be an "idx += num" next to the ptr_free at the end of the second loop.

Subject: Re: Merits of different ways of 'extending' arrays
Posted by suicidaleggroll on Mon, 16 Sep 2013 14:47:38 GMT
View Forum Message <> Reply to Message

The only problem with that is, what is "big enough"? It's going to change from application to application. What happens when your assumption of "big enough" breaks down? Do you have support for re-allocating the arrays when you hit their limits? In order to avoid this you have to make the array SO big that you can start to run into significant memory allocation delays, even when loading just a small amount of data.

Allocating and expanding in fixed "blocks" as suggested before is a way to elegantly handle this problem, however the block size needs to be tuned for every application or you can start to get some big slowdowns.

Just some things to consider when choosing your approach.

On Saturday, September 14, 2013 7:11:07 PM UTC-6, AMS wrote:
> Thanks for the continuing tips!
>
>
>
> The first suggestion (allocate a 'big enough' array up-front, rather than continually extend) worked great for my purposes, so that's what I stuck with, given that it was also very simple. Although I appreciate the continued suggestions.
>
>
>
> Andy
>
>
>
> On Tuesday, September 10, 2013 4:08:40 PM UTC-4, suicida...@gmail.com wrote:
>
>> On Monday, September 9, 2013 4:41:10 PM UTC-6, suicida...@gmail.com wrote:
>

>>
>
>>> Another option is to set up a pointer array nfiles long before the loop, inside the loop load the file and find the valid points, then put that array into that file's pointer, while incrementing a counter to keep track of the total number of points. When you're done, you have all of your data saved in pointers (one per file), and a count of the total number of valid points. Then you allocate your array, loop back through the elements of the pointer array, and fill the array as necessary. Something like:
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> f = file_search(path, count=nfiles)
>
>>
>
>>>
>
>>
>
>>> ptrs = ptrarr(nfiles)
>
>>
>
>>>
>
>>
>
>>> num = 0l
>
>>
>
>>>
>
>>
>
>>> for i=0l,nfiles-1 do begin

```
>
>>
>
>>>
>
>>
>
>>>    ;; load contents of file
>
>>
>
>>>
>
>>
>
>>>    is_valid = where(stuff, n_valid)
>
>>
>
>>>
>
>>
>
>>>    if n_valid gt 0 then begin
>
>>
>
>>>
>
>>
>
>>>       num += n_valid
>
>>
>
>>>
>
>>
>
>>>       ptrs[i] = ptr_new(f.var_1[is_valid])
>
>>
>
>>>
>
>>
>
>>>    endif
```

```
>
>>
>
>>>
>
>>
>
>>>  endfor
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>  data = fltarr(num)
>
>>
>
>>>
>
>>
>
>>>  idx = 0l
>
>>
>
>>>
>
>>
>
>>>  for i=0l,nfiles-1 do begin
>
>>
>
>>>
>
>>
>
>>>     if ptr_valid(ptrs[i]) then begin
```

>
>>
>
>>>
>
>>
>
>>>   num = n_elements(*ptrs[i])
>
>>
>
>>>
>
>>
>
>>>   data[idx:idx+num-1] = *ptrs[i]
>
>>
>
>>>
>
>>
>
>>>   ptr_free, ptrs[i]
>
>>
>
>>>
>
>>
>
>>>   endif
>
>>
>
>>>
>
>>
>
>>> endfor
>
>>
>
>>
>
>>
>
>> Wish I could edit my post...

>
>>
>
>>
>
>>
>
>> There should be an "idx += num" next to the ptr_free at the end of the second loop.

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Andy Sayer on Tue, 17 Sep 2013 17:40:35 GMT
View Forum Message <> Reply to Message

Right, but for this particular piece of code, I do know the maximum possible size (it's a standalone function I will call for one piece of analysis, as opposed to something I am plugging into the guys of a larger project), so it's good for this application. :)

On Monday, September 16, 2013 10:47:38 AM UTC-4, suicida...@gmail.com wrote:
> The only problem with that is, what is "big enough"? It's going to change from application to application. What happens when your assumption of "big enough" breaks down? Do you have support for re-allocating the arrays when you hit their limits? In order to avoid this you have to make the array SO big that you can start to run into significant memory allocation delays, even when loading just a small amount of data.
>
>
>
> Allocating and expanding in fixed "blocks" as suggested before is a way to elegantly handle this problem, however the block size needs to be tuned for every application or you can start to get some big slowdowns.
>
>
>
> Just some things to consider when choosing your approach.
>
>
>
> On Saturday, September 14, 2013 7:11:07 PM UTC-6, AMS wrote:
>
>> Thanks for the continuing tips!
>
>>
>
>>
>
>>
>
>> The first suggestion (allocate a 'big enough' array up-front, rather than continually extend)

worked great for my purposes, so that's what I stuck with, given that it was also very simple. Although I appreciate the continued suggestions.
>
>>
>
>>
>
>>
>
>> Andy
>
>>
>
>>
>
>>
>
>> On Tuesday, September 10, 2013 4:08:40 PM UTC-4, suicida...@gmail.com wrote:
>
>>
>
>>> On Monday, September 9, 2013 4:41:10 PM UTC-6, suicida...@gmail.com wrote:
>
>>
>
>>>
>
>>
>
>>>> Another option is to set up a pointer array nfiles long before the loop, inside the loop load the file and find the valid points, then put that array into that file's pointer, while incrementing a counter to keep track of the total number of points. When you're done, you have all of your data saved in pointers (one per file), and a count of the total number of valid points. Then you allocate your array, loop back through the elements of the pointer array, and fill the array as necessary. Something like:
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>

```
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>  f = file_search(path, count=nfiles)
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>  ptrs = ptrarr(nfiles)
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
```

```
>>
>
>>>>  num = 0l
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>  for i=0l,nfiles-1 do begin
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>    ;; load contents of file
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
```

```
>>
>
>>>>    is_valid = where(stuff, n_valid)
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>    if n_valid gt 0 then begin
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>      num += n_valid
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
```

```
>>
>
>>>>       ptrs[i] = ptr_new(f.var_1[is_valid])
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>    endif
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>  endfor
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
```

```
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>  data = fltarr(num)
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>  idx = 0l
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
```

```
>>
>
>>>>  for i=0l,nfiles-1 do begin
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>    if ptr_valid(ptrs[i]) then begin
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>       num = n_elements(*ptrs[i])
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
```

```
>>
>
>>>>       data[idx:idx+num-1] = *ptrs[i]
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>       ptr_free, ptrs[i]
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
>>
>
>>>>    endif
>
>>
>
>>>
>
>>
>
>>>>
>
>>
>
>>>
>
```

>>
>
>>>> endfor
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> Wish I could edit my post...
>
>>
>
>>>
>
>>
>
>>>
>
>>
>
>>> There should be an "idx += num" next to the ptr_free at the end of the second loop.

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Yngvar Larsen on Tue, 17 Sep 2013 19:34:35 GMT
View Forum Message <> Reply to Message

On Monday, 16 September 2013 15:47:38 UTC+1, suicida...@gmail.com wrote:

> Allocating and expanding in fixed "blocks" as suggested before is a way to elegantly handle this problem, however the block size needs to be tuned for every application or you can start to get some big slowdowns.

In the generic case when "big enough" is not known, the best algorithm is to double the size of the

array every time you hit the current capacity. (Or 3x or 1.5x, does not matter as long as the growth is exponential in as a function of the number of resizes.)

See

http://en.wikipedia.org/wiki/Dynamic_array

--
Yngvar

Subject: Re: Merits of different ways of 'extending' arrays
Posted by Michael Galloy on Tue, 17 Sep 2013 21:47:17 GMT
View Forum Message <> Reply to Message

On 9/17/13 1:34 PM, Yngvar Larsen wrote:
> On Monday, 16 September 2013 15:47:38 UTC+1, suicida...@gmail.com
> wrote:
>
>> Allocating and expanding in fixed "blocks" as suggested before is a
>> way to elegantly handle this problem, however the block size needs
>> to be tuned for every application or you can start to get some big
>> slowdowns.
>
> In the generic case when "big enough" is not known, the best
> algorithm is to double the size of the array every time you hit the
> current capacity. (Or 3x or 1.5x, does not matter as long as the
> growth is exponential in as a function of the number of resizes.)
>
> See
>
> http://en.wikipedia.org/wiki/Dynamic_array
>
>
>

This is what MGcoArrayList does.

https://github.com/mgalloy/mglib/blob/master/src/collection/ mgcoarraylist__define.pro

I used to add capacity in increments of a BLOCK_SIZE property set by the user, but I think the doubling is the way to go.

Mike
--

Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)
Research Mathematician
Tech-X Corporation