Subject: Re: Modifying an array while conserving memory
Posted by David Fanning on Fri, 24 May 2002 02:58:28 GMT
View Forum Message <> Reply to Message

Randall Skelton (rhskelto@atm.ox.ac.uk) writes:

> I have a large array and I would like to 'insert' some data into the
> middle of it. Imagine an array of 1000 points and having 100 points to
> insert beginning at index 500 (the resulting array will have 1100 points).
> Typically, I do not know the length of data I wish to insert until after
> 'a' is defined.
>
> a = findgen(1000)
> b = randomu(seed,100)
> c = fltarr(1100) ; seems wasteful to use more memory
> c[0:499] = a[0:499]
> c[500:599] = b
> c[600:1099] = a[500:999]
>
> In reality, 'a' is of order 2e7 so I would like to avoid making
> multiple copies of it. Does anyone have any suggestions regarding the
> most memory efficient way of doing this?

I think you are going to have to use some memory to
do this. I always do it like this:

```
a = [a[0:499], array_to_insert, a[500;*]]
```

There are usually some tests to tell if the insertion
point is the first point, the last point, or something
in between.

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Modifying an array while conserving memory
Posted by Craig Markwardt on Fri, 24 May 2002 03:03:36 GMT
View Forum Message <> Reply to Message

Randall Skelton <rhskelto@atm.ox.ac.uk> writes:

> Hi all,
>
> I have a large array and I would like to 'insert' some data into the
> middle of it. Imagine an array of 1000 points and having 100 points to
> insert beginning at index 500 (the resulting array will have 1100 points).
> Typically, I do not know the length of data I wish to insert until after
> 'a' is defined.
>
> a = findgen(1000)
> b = randomu(seed,100)
> c = fltarr(1100) ; seems wasteful to use more memory
> c[0:499] = a[0:499]
> c[500:599] = b
> c[600:1099] = a[500:999]
>
> In reality, 'a' is of order 2e7 so I would like to avoid making
> multiple copies of it. Does anyone have any suggestions regarding the
> most memory efficient way of doing this?

Hi Randall--

At one time I devised a set of functions called ARRINSERT and
ARRDELETE, in which I tried to do insertion and deletion operations as
efficiently as I thought possible.

I don't think I was able to get around your concern of necessarily
allocating at least as much storage as the original data. What is
really needed at the IDL internal level, is the ability to **augment**
the storage of an existing array efficiently. I tend to do that alot,
say when I'm building a list element by element.

Also, to this day I have strong doubts whether the TEMPORARY()
operator actually saves memory, *during* the operation. [Of course
it saves memory afterwards since the variable's memory is released.]

Craig

http://cow.physics.wisc.edu/~craigm/idl/idl.html (under arrays)

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: Modifying an array while conserving memory
Posted by R.Bauer on Fri, 24 May 2002 06:44:46 GMT
View Forum Message <> Reply to Message

Randall Skelton wrote:
>
> Hi all,
>
> I have a large array and I would like to 'insert' some data into the
> middle of it. Imagine an array of 1000 points and having 100 points to
> insert beginning at index 500 (the resulting array will have 1100 points).
> Typically, I do not know the length of data I wish to insert until after
> 'a' is defined.
>
> a = findgen(1000)
> b = randomu(seed,100)
> c = fltarr(1100) ; seems wasteful to use more memory
> c[0:499] = a[0:499]
> c[500:599] = b
> c[600:1099] = a[500:999]
>
> In reality, 'a' is of order 2e7 so I would like to avoid making
> multiple copies of it. Does anyone have any suggestions regarding the
> most memory efficient way of doing this?
>
> Many thanks,
> Randall

Why not using pointer:

```
ptr1 = PTR_NEW(FINDGEN(1000))
insert = PTR_NEW(RANDOMU(seed,100))
a = PTR_NEW([(*ptr1)[0:499], (*insert), (*ptr1)[500:*]])


HELP,*a
```

regards
Reimar

--
Reimar Bauer

Institut fuer Stratosphaerische Chemie (ICG-I)
Forschungszentrum Juelich
email: R.Bauer@fz-juelich.de
---------- -------
a IDL library at ForschungsZentrum Juelich
http://www.fz-juelich.de/icg/icg1/idl_icglib/idl_lib_intro.h tml

Subject: Re: Modifying an array while conserving memory
Posted by Randall Skelton on Fri, 24 May 2002 09:06:54 GMT
View Forum Message <> Reply to Message

> Why not using pointer:
>
> ptr1 = PTR_NEW(FINDGEN(1000))
> insert = PTR_NEW(RANDOMU(seed,100))
> a = PTR_NEW([(*ptr1)[0:499], (*insert), (*ptr1)[500:*]])
>
>
> HELP,*a

The problem with using pointers as above is that you are not actually using the pointer, but copying the data contained within. Take a look at the heap after doing the above:

```
IDL> help, /heap
Heap Variables:
   # Pointer: 3
   # Object : 0

<PtrHeapVar1>   FLOAT     = Array[1000]
<PtrHeapVar2>   FLOAT     = Array[100]
<PtrHeapVar3>   FLOAT     = Array[1100]
```

This shows that until I physically free the pointers 'ptr1' and 'insert', I have used exactly double the memory as I now have a copy of each variable.

Rather than inserting the data into the middle, I would (at this point) be happy enough just concatenating to arrays...

```
IDL> ptr1 = PTR_NEW(FINDGEN(1000))
IDL> ptr2 = PTR_NEW(RANDOMU(seed,100))
IDL> a = [ptr1,ptr2]
IDL> print, *a ; fails
IDL> print, *(a) ; fails
IDL> print, *a(*) ; fails
IDL> print, *a[0] ; prints findgen(1000) (i.e. not what I want)
IDL> print, *(a)(*) ; fails... Score: IDL 5 ; Randall 0
```

Because IDL doesn't keep track of what type of data is in a pointer, the above is protecting me from doing silly things:

IDL> ptr1 = PTR_NEW(FINDGEN(1000))
IDL> insert = ptr_new('test')
IDL> a = [ptr1, insert]

Perhaps this is something for dlm's. If I pass 'ptr1', 'insert' and the indices for insertion into C I may be able to resize using 'ptr1' realloc, shift the data around using pointers and trick the IDL variable structure when sending the data back. This sounds risky but at this point all my alternatives read, '% Unable to allocate memory: to make array'.

Cheers,
Randall

Subject: Re: Modifying an array while conserving memory
Posted by R.Bauer on Fri, 24 May 2002 09:49:14 GMT
View Forum Message <> Reply to Message

Randall Skelton wrote:
>
>> Why not using pointer:
>>
>> ptr1 = PTR_NEW(FINDGEN(1000))
>> insert = PTR_NEW(RANDOMU(seed,100))
>> a = PTR_NEW([(*ptr1)[0:499], (*insert), (*ptr1)[500:*]])
>>
>>
>> HELP,*a
>
> The problem with using pointers as above is that you are not actually
> using the pointer, but copying the data contained within. Take a look at
> the heap after doing the above:
>
> IDL> help, /heap
> Heap Variables:
> # Pointer: 3
> # Object : 0
>
> <PtrHeapVar1> FLOAT = Array[1000]
> <PtrHeapVar2> FLOAT = Array[100]
> <PtrHeapVar3> FLOAT = Array[1100]
>
> This shows that until I physically free the pointers 'ptr1' and 'insert',
> I have used exactly double the memory as I now have a copy of each
> variable.
>
> Rather than inserting the data into the middle, I would (at this point) be
> happy enough just concatenating to arrays...

>
> IDL> ptr1 = PTR_NEW(FINDGEN(1000))
> IDL> ptr2 = PTR_NEW(RANDOMU(seed,100))
> IDL> a = [ptr1,ptr2]
> IDL> print, *a ; fails
> IDL> print, *(a) ; fails
> IDL> print, *a(*) ; fails
> IDL> print, *a[0] ; prints findgen(1000) (i.e. not what I want)
> IDL> print, *(a)(*) ; fails... Score: IDL 5 ; Randall 0

If you use my dref function

http://www.fz-juelich.de/icg/icg-i/idl_icglib/idl_source/idl _html/dbase/download/dref.tar.gz

; PURPOSE:
; This functions derefences pointers. If last dimension is 1 this is returned and not the
; standard IDL Array without last dimension 1.
; If value is a array of pointer the values are concatinated by concatinate_arrays at the last dimension
; If value isn't a pointer this value is returned but with the right dimensions.

e.g.
IDL> help,dref(a)
<Expression> FLOAT = Array[1100]

e.g.:
IDL> help,dref(a,/free)
<Expression> FLOAT = Array[1100]

; free means ptr_free

this is solved but during operation the memory is double times allocated.

regards
Reimar

>
> Because IDL doesn't keep track of what type of data is in a pointer, the
> above is protecting me from doing silly things:
>

> IDL> ptr1 = PTR_NEW(FINDGEN(1000))
> IDL> insert = ptr_new('test')
> IDL> a = [ptr1, insert]
>
> Perhaps this is something for dlm's. If I pass 'ptr1', 'insert' and the
> indices for insertion into C I may be able to resize using 'ptr1' realloc,
> shift the data around using pointers and trick the IDL variable structure
> when sending the data back. This sounds risky but at this point all my
> alternatives read, '% Unable to allocate memory: to make array'.
>
> Cheers,
> Randall

--
Reimar Bauer

Institut fuer Stratosphaerische Chemie (ICG-I)
Forschungszentrum Juelich
email: R.Bauer@fz-juelich.de
---------- -------
a IDL library at ForschungsZentrum Juelich
http://www.fz-juelich.de/icg/icg1/idl_icglib/idl_lib_intro.h tml
========== =======

Subject: Re: Modifying an array while conserving memory
Posted by Randall Skelton on Fri, 24 May 2002 11:45:28 GMT
View Forum Message <> Reply to Message

On 23 May 2002, Craig Markwardt wrote:

> At one time I devised a set of functions called ARRINSERT and
> ARRDELETE, in which I tried to do insertion and deletion operations as
> efficiently as I thought possible.
>
> I don't think I was able to get around your concern of necessarily
> allocating at least as much storage as the original data. What is
> really needed at the IDL internal level, is the ability to **augment**
> the storage of an existing array efficiently. I tend to do that alot,
> say when I'm building a list element by element.

It seems this will be my 3rd feature request of the week as my long-shot
attempts at using realloc on an IDL passed array have failed miserably :(

Cheers,
Randall

--

```
void IDL_CDECL am_resize(int argc, IDL_VPTR argv[], char *argk) {

    /* Local */
    float *A;
    long n, l;

    /* IDL inputs: Ensure we get a float array from IDL */
    IDL_ENSURE_ARRAY(argv[0]);

    if (argv[0]->value.arr->n_dim != 1) {
        IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
            "Passed array must only contain one dimension");
    }

    if (argv[0]->type != IDL_TYP_FLOAT) {
        IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
            "Passed array must be of float precision");
    }

    /* Get a local pointer to the passed array */
    A = (float *) argv[0]->value.arr->data;

    /* Get number of elements */
    n = (long) argv[0]->value.arr->n_elts;

    /* IDL inputs: Ensure we get a scalar from IDL */
    IDL_ENSURE_SCALAR(argv[1]);

    /* Handle different precisions for new length */
    switch (argv[1]->type) {
        case IDL_TYP_FLOAT:  l = (long) argv[1]->value.f;
                             break;
        case IDL_TYP_DOUBLE: l = (long) argv[1]->value.l;
                             break;
        case IDL_TYP_INT:    l = (long) argv[1]->value.i;
                             break;
        case IDL_TYP_LONG:   l = (long) argv[1]->value.l;
                             break;
        default: IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
                    "Type of passed length not recognized");
    }

    /* printf("Got: %li elemnts.  New length: %li.", n, l); */

    /* Watch me Seg fault... */
    if (realloc(A, 0) == NULL) {
        IDL_Message(IDL_M_NAMED_GENERIC, IDL_MSG_LONGJMP,
            "Cannot allocate memory");
```

```
    }

    /* Reset the number of elements in the IDL variable structure */
    /* argv[0]->value.arr->n_elts = (IDL_MEMINT) l; */

}
```

Subject: Re: Modifying an array while conserving memory
Posted by Pavel A. Romashkin on Fri, 24 May 2002 16:33:16 GMT
View Forum Message <> Reply to Message

I think I have evidence that it does. I have had on several occasions ran out of ram (on a Mac with 1.5 Gb of RAM), and Temporary did help. But, again, Temporary only is useful when data is changed, not for reallocation of any kind or operations on array subsets.
When working with large datasets, efficient use of RAM is a must, otherwise you simply can't run the program.
I can relate to the original question, and think that if you really run into RAM limitations, you have to know first what is the size of the inserted array and allocate that ahead of time. Reallocating large arrays is, first, very slow, and secondly is almost impossible once you reach the limit of unfragmented memory page.
I found that once you are in this realm, it makes a big difference when and how you allocate even small variables because contiguous RAM becomes more important than the total RAM.
Cheers,
Pavel

Craig Markwardt wrote:
>
> Also, to this day I have strong doubts whether the TEMPORARY()
> operator actually saves memory, *during* the operation. [Of course
> it saves memory afterwards since the variable's memory is released.]
