Subject: Re: The continuing saga of WHERE and 2D
Posted by Sean Raffuse on Thu, 27 Feb 2003 20:55:54 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.18c829481a588402989b05@news.frii.com...
> Sean Raffuse (sean@me.wustl.edu) writes:
>
>> Ok, so I've found the nifty little WhereToMulti program that converts 1D
>> where() results back to the original 2d or 3d indices. My question, and
I
>> am sensing that it is a dumb one, is how can I use these returned
indices
>> properly?
>>
>> Example:
>>
>> ;the part that works (stolen wholesale from D Fanning's website)
>> index = WHERE(image EQ test)
>> s = SIZE(image)
>> ncol = s(1)
>> col = index MOD ncol
>> row = index / ncol
>>
>> ;the part I am confused about
>> image[col, row] = PassedTheTest
>>
>> *** Error, too many elements in array. . . and you're ugly. ***
>
> Say what!? I don't think so. :-)
>
> Are you sure all your variables are defined here, Sean?
> Let's see your real code and not this pseudo code.
>

Ok, you may disagree and same I look fabulous, but I still get the error.
Here is my code snippet. Let me know if you need more.

```
HighRedorGreenLand = where(MinArray8[*,*] GT default_Bd8WaterLimit $ ;pixel
is not water
                    AND ((Tau[*,*,5] GT Tau[*,*,0]) OR (Tau[*,*,4] GT
Tau[*,*,0])))
WhereToMulti, MinArray8, HighRedorGreenLand, x_mask, y_mask
Tau[x_mask,y_mask,*] = default_BelowDetLimit

ENVI> help, MinArray8
MINARRAY8 INT = Array[3600, 1680]
ENVI> help, Tau
```

```
TAU FLOAT = Array[3600, 1680, 8]
ENVI> help, HighRedorGreenLand
HIGHREDORGREENLAND
LONG = Array[1407735]
ENVI> help, x_mask
X_MASK LONG = Array[1407735]
ENVI> help, y_mask
Y_MASK LONG = Array[1407735]
```

Subject: Re: The continuing saga of WHERE and 2D
Posted by David Fanning on Thu, 27 Feb 2003 20:55:54 GMT

View Forum Message <> Reply to Message

Sean Raffuse (sean@me.wustl.edu) writes:

```
> Ok, so I've found the nifty little WhereToMulti program that converts 1D
> where() results back to the original 2d or 3d indices.  My question, and I
> am sensing that it is a dumb one, is how can I use these returned indices
> properly?
>
> Example:
>
> ;the part that works (stolen wholesale from D Fanning's website)
> index = WHERE(image EQ test)
> s = SIZE(image)
> ncol = s(1)
> col = index MOD ncol
> row = index / ncol
>
> ;the part I am confused about
> image[col, row] = PassedTheTest
>
> *** Error, too many elements in array. . . and you're ugly. ***
```

Say what!? I don't think so. :-)

Are you sure all your variables are defined here, Sean?
Let's see your real code and not this pseudo code.

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: The continuing saga of WHERE and 2D
Posted by Liam E. Gumley on Thu, 27 Feb 2003 21:07:55 GMT
View Forum Message <> Reply to Message

"Sean Raffuse" <sean@me.wustl.edu> wrote in message
news:b3lt4h$5gr$1@newsreader.wustl.edu...
> Ok, so I've found the nifty little WhereToMulti program that converts 1D
> where() results back to the original 2d or 3d indices. My question, and I
> am sensing that it is a dumb one, is how can I use these returned indices
> properly?
>
> Example:
>
> ;the part that works (stolen wholesale from D Fanning's website)
> index = WHERE(image EQ test)
> s = SIZE(image)
> ncol = s(1)
> col = index MOD ncol
> row = index / ncol
>
> ;the part I am confused about
> image[col, row] = PassedTheTest
>
> *** Error, too many elements in array. . . and you're ugly. ***

Maybe I'm missing something, but why not use

image[index] = PassedTheTest

You're allowed to use 1D indices to access any n-dimensional array.

Cheers,
Liam.
Practical IDL Programming
http://www.gumley.com/

Subject: Re: The continuing saga of WHERE and 2D
Posted by David Fanning on Thu, 27 Feb 2003 21:29:12 GMT
View Forum Message <> Reply to Message

Sean Raffuse (sean@me.wustl.edu) writes:

> Ok, you may disagree and same I look fabulous, but I still get the error.

> Here is my code snippet. Let me know if you need more.
>
> HighRedorGreenLand = where(MinArray8[*,*] GT default_Bd8WaterLimit $;pixel
> is not water
>   AND ((Tau[*,*,5] GT Tau[*,*,0]) OR (Tau[*,*,4] GT
> Tau[*,*,0])))
> WhereToMulti, MinArray8, HighRedorGreenLand, x_mask, y_mask
> Tau[x_mask,y_mask,*] = default_BelowDetLimit
>
> ENVI> help, MinArray8
> MINARRAY8 INT = Array[3600, 1680]
> ENVI> help, Tau
> TAU FLOAT = Array[3600, 1680, 8]
> ENVI> help, HighRedorGreenLand
> HIGHREDORGREENLAND
> LONG = Array[1407735]
> ENVI> help, x_mask
> X_MASK LONG = Array[1407735]
> ENVI> help, y_mask
> Y_MASK LONG = Array[1407735]

Well, I don't see any error here. Nor do I see anything
that should cause an error. Is there more somewhere?

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
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: The continuing saga of WHERE and 2D
Posted by Sean Raffuse on Thu, 27 Feb 2003 21:30:12 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.18c8310e9a99b4ce989b06@news.frii.com...
> Sean Raffuse (sean@me.wustl.edu) writes:
>
>> Ok, you may disagree and same I look fabulous, but I still get the
error.
>> Here is my code snippet. Let me know if you need more.
>>

```
>>  HighRedorGreenLand = where(MinArray8[*,*] GT default_Bd8WaterLimit $
;pixel
>>  is not water
>>                        AND ((Tau[*,*,5] GT Tau[*,*,0]) OR (Tau[*,*,4]
GT
>>  Tau[*,*,0])))
>>  WhereToMulti, MinArray8, HighRedorGreenLand, x_mask, y_mask
>>  Tau[x_mask,y_mask,*] = default_BelowDetLimit
>>
>>  ENVI> help, MinArray8
>>  MINARRAY8 INT = Array[3600, 1680]
>>  ENVI> help, Tau
>>  TAU FLOAT = Array[3600, 1680, 8]
>>  ENVI> help, HighRedorGreenLand
>>  HIGHREDORGREENLAND
>>  LONG = Array[1407735]
>>  ENVI> help, x_mask
>>  X_MASK LONG = Array[1407735]
>>  ENVI> help, y_mask
>>  Y_MASK LONG = Array[1407735]
```
>
> Well, I don't see any error here. Nor do I see anything
> that should cause an error. Is there more somewhere?
>
> Cheers,
>
> David

No, that's it. Although here is the error

```
 Tau[x_mask,y_mask,*] = default_BelowDetLimit
% Array has too many elements.
```

Is this a memory problem?

-Sean

Subject: Re: The continuing saga of WHERE and 2D
Posted by David Fanning on Thu, 27 Feb 2003 22:47:12 GMT
View Forum Message <> Reply to Message

Sean Raffuse (sean@me.wustl.edu) writes:

> No, that's it. Although here is the error
>
>   Tau[x_mask,y_mask,*] = default_BelowDetLimit
> % Array has too many elements.

>
> Is this a memory problem?

Humm. A memory problem? I don't know. 1.4 million
array elements seems like a lot, but I can
easily do this:

```
IDL> a=FltArr(3660, 1680)
IDL> b = randomu(seed, 1407735L) * 3600L * 1680L
IDL> a[b] = 5
```

I have a problem, however, when I add a third
dimension:

```
IDL> DelVar, a
IDL> a = FltArr(3660, 1680, 8)
IDL> a[b,b, *] = 5
% Array has too many elements.
IDL> a[b,b, 0] = 5
% Array has too many elements.
```

I don't know what that is. I think you are going to have
to ask RSI.

I do notice that if I do this:

```
IDL> c= b[0:4999]
IDL> a[c,c,*] = 5
```

That my computer gets very, VERY unhappy. :-(
I had to reboot to get some response back.

I think this means there is something going on here
that I don't understand. :-)

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
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: The continuing saga of WHERE and 2D
Posted by marc schellens[1] on Fri, 28 Feb 2003 06:43:00 GMT
View Forum Message <> Reply to Message

Sean Raffuse wrote:
> "David Fanning" <david@dfanning.com> wrote in message
> news:MPG.18c829481a588402989b05@news.frii.com...
>
>> Sean Raffuse (sean@me.wustl.edu) writes:
>>
>>
>>> Ok, so I've found the nifty little WhereToMulti program that converts 1D
>>> where() results back to the original 2d or 3d indices. My question, and
>>>
> I
>
>>> am sensing that it is a dumb one, is how can I use these returned
>>>
> indices
>
>>> properly?
>>>
>>> Example:
>>>
>>> ;the part that works (stolen wholesale from D Fanning's website)
>>> index = WHERE(image EQ test)
>>> s = SIZE(image)
>>> ncol = s(1)
>>> col = index MOD ncol
>>> row = index / ncol
>>>
>>> ;the part I am confused about
>>> image[col, row] = PassedTheTest
>>>
>>> *** Error, too many elements in array. . . and you're ugly. ***
>>>
>> Say what!? I don't think so. :-)
>>
>> Are you sure all your variables are defined here, Sean?
>> Let's see your real code and not this pseudo code.
>>
>>
>
> Ok, you may disagree and same I look fabulous, but I still get the error.
> Here is my code snippet. Let me know if you need more.
>
> HighRedorGreenLand = where(MinArray8[*,*] GT default_Bd8WaterLimit $;pixel
> is not water
>   AND ((Tau[*,*,5] GT Tau[*,*,0]) OR (Tau[*,*,4] GT

```
> Tau[*,*,0])))
> WhereToMulti, MinArray8, HighRedorGreenLand, x_mask, y_mask
> Tau[x_mask,y_mask,*] = default_BelowDetLimit
>
> ENVI> help, MinArray8
> MINARRAY8 INT = Array[3600, 1680]
> ENVI> help, Tau
> TAU FLOAT = Array[3600, 1680, 8]
> ENVI> help, HighRedorGreenLand
> HIGHREDORGREENLAND
> LONG = Array[1407735]
> ENVI> help, x_mask
> X_MASK LONG = Array[1407735]
> ENVI> help, y_mask
> Y_MASK LONG = Array[1407735]
>
>
>
```

Your problem here is that you index only TWO dimensions of a THREE
dimensional array with the same number of indices.
If ALL indexing arrays have the same size, the result is a one-
dimensional array of that size.
If only one of the dimensions is different, you get an
ix1 * ix2 * ix3 array.
1407735^2 (*1) is a huge number, therefore the error.
To prevent it you may use as third index an lonarr(1407735),
or copy the slice from the three dim array to two dim and later copy
back (or use Liam Gumley's suggestions)
Hope that helps,
marc

Subject: Re: The continuing saga of WHERE and 2D
Posted by marc schellens[1] on Fri, 28 Feb 2003 06:50:52 GMT

View Forum Message <> Reply to Message

For completeness:
of course for indexing with '*' you get the same as if
the size of the indexing arrays differ.
And also notice, that the pseudo code of your initial
posting actually works as there the array is 2d only.

Subject: Re: The continuing saga of WHERE and 2D
Posted by JD Smith on Fri, 28 Feb 2003 17:35:38 GMT

View Forum Message <> Reply to Message

On Thu, 27 Feb 2003 15:47:12 -0700, David Fanning wrote:

```
> Sean Raffuse (sean@me.wustl.edu) writes:
>
>> No, that's it.  Although here is the error
>>
>>  Tau[x_mask,y_mask,*] = default_BelowDetLimit
>> % Array has too many elements.
>>
>> Is this a memory problem?
>
> Humm. A memory problem? I don't know. 1.4 million array elements seems
> like a lot, but I can easily do this:
>
> IDL> a=FltArr(3660, 1680)
> IDL> b = randomu(seed, 1407735L) * 3600L * 1680L IDL> a[b] = 5
>
> I have a problem, however, when I add a third dimension:
>
> IDL> DelVar, a
> IDL> a = FltArr(3660, 1680, 8)
> IDL> a[b,b, *] = 5
> % Array has too many elements.
> IDL> a[b,b, 0] = 5
> % Array has too many elements.
>
> I don't know what that is. I think you are going to have to ask RSI.
>
> I do notice that if I do this:
>
> IDL> c= b[0:4999]
> IDL> a[c,c,*] = 5
>
> That my computer gets very, VERY unhappy. :-( I had to reboot to get
> some response back.
>
> I think this means there is something going on here that I don't
> understand. :-)
```

I think it's pretty simple, if subtle. When it encounters a
multi-dimensional subscript, IDL look to see if all subscript vectors
have the same dimensions. If so, it "threads the list" and constructs
indices from them on the fly as:

```
 [vec1,vec2,vec3,...] ==> vec1+vec2*n1+vec3*n1*n2+...
```

where n1,n2, etc. are the sizes of the 1st,2nd, etc. dimension of the
array being indexed. I.e., you've essentially specified a short list

of index pairs, triples, etc., of length n_elements(vec1).

If, however, any of the subscripts are unspecified or zero-dimensional, by virtue of using a single index or one of the higher-order range operations (e.g. `*' or '0:5'), a temporary large array of indices has to be pre-created. Why? Because you can no longer "thread the list". Example: if I say

```
  a[ [1,2,3] , [4,5,6], 0]
```

you might think I mean for IDL to generate a list like:

```
  a[1, 4, 0]
  a[2, 5, 0]
  a[3, 6, 0]
```

but it actually expands this to:

```
  a[1, 4, 0] a[1, 5, 0] a[1, 6, 0]
  a[2, 4, 0] a[2, 5, 0] a[2, 6, 0]
  a[3, 4, 0] a[3, 5, 0] a[3, 6, 0]
```

This can be a very important distinction when subscripting with large index vectors.

Here's an example demonstrating this:

```
IDL> a=randomu(sd,100,100,100)
IDL> help,/memory
heap memory used:    4392171, max:    4392190, gets:     1895, frees:     1475
IDL> a[*,*,*]=1
IDL> help,/memory
heap memory used:    4392203, max:    8392260, gets:     1899, frees:     1477
IDL> print,(8392260-4392190)/4
     1000017
```

Ahah, it seems a temporary index array of 100*100*100 indices was made. Make sense. What if we use three index vectors of the same size?

```
IDL> a=randomu(sd,100,100,100)
IDL> r1=randomu(sd,100) & r2=randomu(sd,100) & r3=randomu(sd,100)
IDL> help,/memory
heap memory used:    4385413, max:    4385432, gets:     1212, frees:      807
IDL> a[r1,r2,r3]=1
IDL> help,/memory
heap memory used:    4385448, max:    4386374, gets:     1218, frees:      811
```

In this case, a temporary index array was not needed; the three r
vectors were used together directly as a threaded list, and no extra
memory was used. How about:

```
IDL> a=randomu(sd,100,100,100)
IDL> help,/memory
heap memory used:    4383915, max:    4383934, gets:     1206, frees:      806
IDL> r1=randomu(sd,100) & r2=randomu(sd,100)
IDL> help,/memory
heap memory used:    4384920, max:    4384939, gets:     1211, frees:      807
IDL> a[r1,r2,*]=1
IDL> help,/memory
heap memory used:    4384954, max:    8385481, gets:     1217, frees:      811
```

It seems 100*100*100 indices where created here too. Looks right.
Now, let's stress things a bit:

```
IDL> a=randomu(sd,100,100,100)
IDL> r1=randomu(sd,1000) & r2=randomu(sd,1000) & r3=randomu(sd,1000)
IDL> help,/memory
heap memory used:    4396199, max:    4396218, gets:     1210, frees:      806
IDL> a[r1,r2,r3]=1
IDL> help,/memory
heap memory used:    4396234, max:    4404360, gets:     1216, frees:      810
```

Wait a minute, what's happening here? The subscript vectors, despite
being larger than the dimensions of the array they're accessing, are
still just being used directly, with no additional overhead required
for creating a temporary index array. The assignment to 1 occurs 1000
times.

What about:

```
IDL> a=randomu(sd,100,100,100)
IDL> r1=randomu(sd,1000) & r2=randomu(sd,1000)
IDL> help,/memory
heap memory used:    4392105, max:    4392124, gets:     1209, frees:      806
IDL> a[r1,r2,*]=1
IDL> help,/memory
heap memory used:    4400378, max:  404396266, gets:     1902, frees:     1478
IDL> print,(404396266-4392124)/4
   100001035
```

Uh oh. You can see that IDL had to pre-allocate a temporary index
entry on the fly with 1000*1000*100 elements, despite the fact that it
was used to index a much smaller array. The assignment to 1 occurs
100,000,000 times! Quite a difference. I can take this to the
extreme:

```
IDL> a=randomu(sd,1,1,1)
IDL> r1=randomu(sd,10000L) & r2=randomu(sd,10000L)
IDL> help,/memory
heap memory used:     464107, max:     464126, gets:     1209, frees:      806
IDL> a[r1,r2,0]=1  ; long delay
IDL> help,/memory
heap memory used:     472380, max:  400504268, gets:     1902, frees:     1478
IDL> print,(400504268-464126)/4
   100010035
```

Oh my, nearly 1/2 Gb was allocated for the temporary index array just
to assign a value to a *single* element (over and over again). What
if I pre-build my index vector:

```
IDL> a=randomu(sd,1,1,1)
IDL> r1=randomu(sd,10000L) & r2=randomu(sd,10000L)
IDL> r=r1+1*r2
IDL> help,/memory
heap memory used:     504193, max:     504212, gets:     1211, frees:      806
IDL> a[r]=1   ; no delay
IDL> help,/memory
heap memory used:     504221, max:     544282, gets:     1215, frees:      808
```

What a difference this makes.

Bottom line? Keep in mind this duality in how IDL treats arrays as
subscripts, and be very careful when mixing array subscripts with
other types. If you mean for, e.g.

```
 [ [1,2], [3,4], 0 ] ==> [1,3,0], [2,4,0]
```

instead of

```
 [ [1,2], [3,4], 0 ] ==> [1,3,0], [1,4,0],
                  [2,3,0], [2,4,0]
```

Then you should use:

```
 [ [1,2], [3,4], [0,0] ]
```

or just pre-build your indices as a single index vector beforehand.

JD

Subject: Re: The continuing saga of WHERE and 2D

Posted by JD Smith on Fri, 28 Feb 2003 17:43:04 GMT
View Forum Message <> Reply to Message

On Thu, 27 Feb 2003 15:47:12 -0700, David Fanning wrote:

> IDL> c= b[0:4999]
> IDL> a[c,c,*] = 5

This assignement allocates about 800MB of memory for the index array.
Read my other post to find out why ;).

JD

Subject: Re: The continuing saga of WHERE and 2D
Posted by David Fanning on Fri, 28 Feb 2003 17:52:11 GMT
View Forum Message <> Reply to Message

JD Smith (jdsmith@as.arizona.edu) writes:

> Bottom line? Keep in mind this duality in how IDL treats arrays as
> subscripts, and be very careful when mixing array subscripts with
> other types.

Well, there you go, Sean. Thanks, JD! :-)

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
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: The continuing saga of WHERE and 2D
Posted by Pavel Romashkin on Fri, 28 Feb 2003 18:15:11 GMT
View Forum Message <> Reply to Message

Hi JD,
Is the same kind of allocation taking place when one simply calls
FLTARR? I bet yes.
I am still puzzled by the fact I can allocate 1.25 times more in the
form of pointer arrays than as a single large array, like

```
a = FLTARR(290000000) ; This is the limit, out of RAM over this (1.32 Gb)
a = PTRARR(3600, /allocate)
temp = FLTARR(100000)
for i = 0, 3599 do *a[i] = temp ; No problem at all
```

I am not fully convinced it is not the memory fragmentation issue, but
the difference is too significant. Besides, all this is tested with a
fresh instance of IDL. I tried with fragmented memory and of course, the
difference is far more dramatic (like twice the size using smaller arrays).
In any case, knowing it turned out to be very practical to me. Although
it allows to bring the computer really close to crashing (Mac OS tries
to dynamically reallocate for itself, too, and hates to find out it only
has 4 Mb left).
Pavel

JD Smith wrote:
>
> On Thu, 27 Feb 2003 15:47:12 -0700, David Fanning wrote:
>
>> IDL> c= b[0:4999]
>> IDL> a[c,c,*] = 5
>
> This assignement allocates about 800MB of memory for the index array.
> Read my other post to find out why ;).
>
> JD

Subject: Re: The continuing saga of WHERE and 2D
Posted by Paul Van Delst[1] on Fri, 28 Feb 2003 18:26:40 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> JD Smith (jdsmith@as.arizona.edu) writes:
>
>> Bottom line? Keep in mind this duality in how IDL treats arrays as
>> subscripts, and be very careful when mixing array subscripts with
>> other types.
>
> Well, there you go, Sean. Thanks, JD! :-)

Are you going to transplant JD's analysis to your webpage? (please) [*]

That's important stuff.

paulv

[*] after your ATAN nap. :o)

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7274
Fax:(301)763-8545