Subject: majority voting
Posted by Mort Canty on Wed, 11 Feb 2009 16:14:16 GMT
View Forum Message <> Reply to Message

Hi all,

Given a 2-D array such as

```
    0    1    1    2    1
    0    2    1    1    1
    1    0    2    2    1
```

where the entries are labels, the columns represent items and the rows
are voters, I want a IDL function that returns the majority vote labels.
So here I should get

0 ? 1 2 1

as output, where ? = "don't care". There must _not_ be a loop over
columns. I've got a clumsy solution, but I'm sure there's an elegant one
somewhere?

Cheers,

Mort

Subject: Re: majority voting
Posted by ben.bighair on Wed, 11 Feb 2009 22:03:59 GMT
View Forum Message <> Reply to Message

On Feb 11, 4:53 pm, Mort Canty <m.ca...@fz-juelich.de> wrote:
> ben.bighair schrieb:
>
>
>
>> On Feb 11, 11:14 am, mort canty <m.ca...@fz-juelich.de> wrote:
>>> Hi all,
>
>>> Given a 2-D array such as
>

```
>>>      0    1    1    2    1
>>>      0    2    1    1    1
>>>      1    0    2    2    1
>
```

>>> where the entries are labels, the columns represent items and the rows
>>> are voters, I want a IDL function that returns the majority vote labels.
>>> So here I should get

```
>
>>>  0  ?  1  2  1
>
>>>  as output, where ? = "don't care". There must _not_ be a loop over
>>>  columns. I've got a clumsy solution, but I'm sure there's an elegant one
>>>  somewhere?
>
>>  Hi,
>
>>  This is incomplete as it doesn't flag the "don't care" crowd.  I can't
>>  noodle that part out without column looping.  Looping would make it
>>  easy to use something like...
>
>>  for i = 0, ncol-1 do dontCare[i] = ARRAY_EQUAL(votes[i,*],votes[i,0])
>
>>  but by your rules, that is out of bounds.
>
>>  ***BEGIN
>>  x=[[0,1,1,2,1],$
>>  [0,2,1,1,1],$
>>  [1,0,2,2,1]]
>>  sz = SIZE(x, /DIM)
>>  votes = [[TOTAL(x EQ 0, 2)],$
>>    [TOTAL(x EQ 1,  2)], $
>>    [TOTAL(x EQ 2, 2)]]
>>  mx = MAX(votes, mxIdx,dim = 2)
>>  majority = (array_indices(sz, mxIdx, /dim))[1,*]
>>  print, majority
>>  ***END
>
>>  Cheers,
>>  Ben
>
>  Thanks Ben. What I meant by "don't care" is that I don't care which of
>  the labels that got equal votes is output. I think my solution is
>  essentially the same as yours, certainly not more elegant:
>
>  function majority_vote, A, num_labels
>     n = n_elements(A[*,0])
>     B = intarr(n,num_labels)
>     for i=0,num_labels-1 do begin
>        C = A*0
>        idx = where(A eq i,count)
>        if count gt 0 then C[idx] = 1
>        B[*,i] = total(C,2)
>     endfor
>     void = max(B,labels,dimension=2)
>     return, labels/n
```

> end
>
> I was probably hoping for some HISTOGRAM magic :-)
> Mort

Oooo. Histogram... well my mojo just isn't working that way today.
But deep inside Jean's addition is sort_nd (I had never heard of it
before - slick!) which has a histogram *with* reverse indices. Double
the mojo!

Cheers,
Ben

Subject: Re: majority voting
Posted by JD Smith on Wed, 11 Feb 2009 23:20:59 GMT
View Forum Message <> Reply to Message

You're trying to compute the n-dimensional mode. For the 1D case,
see:

  http://www.dfanning.com/code_tips/mode.html

You could of course use SORT_ND and a 2D version of the "find the
longest string of the same number" method on this page. This is very
safe against sparse input distributions -- notice the HISTOGRAM in
SORT_ND is of the sort indices, not the data themselves. It's also
easy to recognize a mode length of 1 as "no mode", and (if you like)
detect multiple modes in the data (aka "tie votes").

However, your problem is a special case, since (I presume) your vote
labels are always low contiguous integers, you don't have to worry
about sparseness. In this case, using HIST_ND directly without
sorting will likely be much faster:

```
  n=n_elements(array)
  s=size(array,/DIMENSIONS)
   h=hist_nd([lindgen(1,n)/s[1],reform(transpose(array),1,n)],1 )
  m=max(h,DIMENSION=2,mode) gt 1
  mode=mode/s[0] * m + m - 1
```

All I'm doing here is forming a 2d space, with the first dimension the
integer ID of the vote (aka column number in your example), and the
second the actual votes cast.

Had you organized your array with the rows as your "vote items" you
could save the transpose. Also note I specifically test for and set
to -1 any mode with frequency of 1 (your "don't care"). I do not

check for multiple modes, but you could do this as well in a
straightforward way. The fact that the dimensions of the 2D histogram
are the same as the input array is incidental, by the way.

JD

Subject: Re: majority voting
Posted by Mort Canty on Thu, 12 Feb 2009 09:15:49 GMT
View Forum Message <> Reply to Message

JD Smith schrieb:
> You're trying to compute the n-dimensional mode. For the 1D case,
> see:
>
>   http://www.dfanning.com/code_tips/mode.html
>
> You could of course use SORT_ND and a 2D version of the "find the
> longest string of the same number" method on this page. This is very
> safe against sparse input distributions -- notice the HISTOGRAM in
> SORT_ND is of the sort indices, not the data themselves. It's also
> easy to recognize a mode length of 1 as "no mode", and (if you like)
> detect multiple modes in the data (aka "tie votes").
>
> However, your problem is a special case, since (I presume) your vote
> labels are always low contiguous integers, you don't have to worry
> about sparseness. In this case, using HIST_ND directly without
> sorting will likely be much faster:
>
```
>   n=n_elements(array)
>   s=size(array,/DIMENSIONS)
>    h=hist_nd([lindgen(1,n)/s[1],reform(transpose(array),1,n)],1 )
>   m=max(h,DIMENSION=2,mode) gt 1
>   mode=mode/s[0] * m + m - 1
```
>
> All I'm doing here is forming a 2d space, with the first dimension the
> integer ID of the vote (aka column number in your example), and the
> second the actual votes cast.
>
> Had you organized your array with the rows as your "vote items" you
> could save the transpose. Also note I specifically test for and set
> to -1 any mode with frequency of 1 (your "don't care"). I do not
> check for multiple modes, but you could do this as well in a
> straightforward way. The fact that the dimensions of the 2D histogram
> are the same as the input array is incidental, by the way.
>
> JD
>

Great newsgroup! Ask for magic and you get ... magic. Thanks very much
for the thorough explanation, JD.

Cheers,

Mort

PS. For some reason HIST_ND isn't in David's zipped COYOTE library
package, although SORT_ND is.

Subject: Re: majority voting
Posted by Allan Whiteford on Thu, 12 Feb 2009 12:17:18 GMT
View Forum Message <> Reply to Message

mort canty wrote:
```
> Hi all,
>
> Given a 2-D array such as
>
>       0     1     1     2     1
>       0     2     1     1     1
>       1     0     2     2     1
>
> where the entries are labels, the columns represent items and the rows
> are voters, I want a IDL function that returns the majority vote labels.
> So here I should get
>
> 0  ?  1  2  1
>
> as output, where ? = "don't care". There must _not_ be a loop over
> columns. I've got a clumsy solution, but I'm sure there's an elegant one
> somewhere?
>
> Cheers,
>
> Mort
```

Hi Mort,

It might be less efficient than JD's histogram solution (I didn't check)
but the following also fits the problem specification:

```
x=[ [0,1,1,2,1],$
 [0,2,1,1,1],$
 [1,0,2,2,1]]
```

```
voters=(size(x,/dim))[1]
items=(size(x,/dim))[0]
max_label=max(x)+1

f=intarr(max_label,items)
++f[max_label*(indgen(voters*items) / voters)+ $
    reform(transpose(x),voters*items)]
junk=max(f,idx,dim=1)
print,idx - max_label*findgen(items)
```

Note that the above solution will also blow up when you end up with sparse arrays (e.g. if you have someone voting for label 1000000 then f will end up being an items x 1000000 array even if nobody votes for any labels between 3 and 1000000).

I think all the discussions on finding the mode (either in 1D or nD) probably pre-dated the ++ operator. It could be that using the vectorised ++ operator is a better way to do it - I doubt it though, normally if histogram can do something then histogram will be the best way! You'd also need to introduce a clumsy offset to deal with negative selections (Not an issue for you here but would be if finding the mode in a more general way).

It would make David's 1D example from his webpage into something like this:

```
array = [1, 1, 2 , 4, 1, 3, 3, 2, 4, 5, 3, 2, 2, 1, 2, 6,-3]
f=intarr(max(array)-min(array)+1)
f[array-min(array)]++
junk=max(f,idx)
mode=idx + min(array)
print,mode
```

again, with no idea on what would be more efficient. If you're doing analysis on measurements (typically non-integers) then you'd need to invoke histogram anyway to bin them before trying to find the mode.

Thanks,

Allan

Subject: Re: majority voting
Posted by David Fanning on Thu, 12 Feb 2009 13:43:31 GMT
View Forum Message <> Reply to Message

Mort Canty writes:

> PS. For some reason HIST_ND isn't in David's zipped COYOTE library

> package, although SORT_ND is.

Odd, I remember thinking yesterday that I had never *heard*
of SORT_ND. What is *it* doing in my library!?

Anyway, they are both there now. Thanks. :-)

Cheers,

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

Subject: Re: majority voting
Posted by JD Smith on Thu, 12 Feb 2009 15:08:53 GMT
View Forum Message <> Reply to Message

On Feb 12, 8:43 am, David Fanning <n...@dfanning.com> wrote:
> Mort Canty writes:
>> PS. For some reason HIST_ND isn't in David's zipped COYOTE library
>> package, although SORT_ND is.
>
> Odd, I remember thinking yesterday that I had never *heard*
> of SORT_ND. What is *it* doing in my library!?
>
> Anyway, they are both there now. Thanks. :-)

Recent versions of both routines can be found on my page:

  http://tir.astro.utoledo.edu/jdsmith/scraps.php

JD

Subject: Re: majority voting
Posted by Mort Canty on Thu, 12 Feb 2009 15:45:09 GMT
View Forum Message <> Reply to Message

Allan Whiteford schrieb:
> mort canty wrote:
>> Hi all,
>>
>> Given a 2-D array such as

```
>>
>>        0      1      1      2      1
>>        0      2      1      1      1
>>        1      0      2      2      1
>>
>>  where the entries are labels, the columns represent items and the rows
>>  are voters, I want a IDL function that returns the majority vote
>>  labels. So here I should get
>>
>>  0  ?  1  2  1
>>
>>  as output, where ? = "don't care". There must _not_ be a loop over
>>  columns. I've got a clumsy solution, but I'm sure there's an elegant
>>  one somewhere?
>>
>>  Cheers,
>>
>>  Mort
>
>  Hi Mort,
>
>  It might be less efficient than JD's histogram solution (I didn't check)
>  but the following also fits the problem specification:
>
>  x=[    [0,1,1,2,1],$
>      [0,2,1,1,1],$
>      [1,0,2,2,1]]
>
>  voters=(size(x,/dim))[1]
>  items=(size(x,/dim))[0]
>  max_label=max(x)+1
>
>  f=intarr(max_label,items)
>  ++f[max_label*(indgen(voters*items) / voters)+ $
>      reform(transpose(x),voters*items)]
>  junk=max(f,idx,dim=1)
>  print,idx - max_label*findgen(items)
>
>  Note that the above solution will also blow up when you end up with
>  sparse arrays (e.g. if you have someone voting for label 1000000 then f
>  will end up being an items x 1000000 array even if nobody votes for any
>  labels between 3 and 1000000).
>
>  I think all the discussions on finding the mode (either in 1D or nD)
>  probably pre-dated the ++ operator. It could be that using the
>  vectorised ++ operator is a better way to do it - I doubt it though,
>  normally if histogram can do something then histogram will be the best
>  way! You'd also need to introduce a clumsy offset to deal with negative
```

> selections (Not an issue for you here but would be if finding the mode
> in a more general way).
>
> It would make David's 1D example from his webpage into something like this:
>
> array = [1, 1, 2 , 4, 1, 3, 3, 2, 4, 5, 3, 2, 2, 1, 2, 6,-3]
> f=intarr(max(array)-min(array)+1)
> f[array-min(array)]++
> junk=max(f,idx)
> mode=idx + min(array)
> print,mode
>
> again, with no idea on what would be more efficient. If you're doing
> analysis on measurements (typically non-integers) then you'd need to
> invoke histogram anyway to bin them before trying to find the mode.
>
> Thanks,
>
> Allan

Hi Allan,

Thanks! Not only have I learned that something called vectorized ++
exists, but that it bumps the indexed value multiple times if that index
is repeated. Live and learn! But where the hell is all that on the IDL Help?

Anyway, what I had in mind was trying to program an ensemble image
classifier, so that the items are rows of pixels (lots and lots), the
labels are land cover classes (contiguous small integers) and the voters
are classifiers (e.g. neural networks, also not too many). Hence the
wish to avoid the loop over items. I certainly got my money's worth :-)

Mort

Subject: Re: majority voting
Posted by JD Smith on Thu, 12 Feb 2009 18:49:02 GMT
View Forum Message <> Reply to Message

On Feb 12, 10:45 am, Mort Canty <m.ca...@fz-juelich.de> wrote:
> Allan Whiteford schrieb:
>
>
>
>> mort canty wrote:
>>> Hi all,
>
>>> Given a 2-D array such as

```
>
>>>       0     1     1     2     1
>>>       0     2     1     1     1
>>>       1     0     2     2     1
>
>>>  where the entries are labels, the columns represent items and the rows
>>>  are voters, I want a IDL function that returns the majority vote
>>>  labels. So here I should get
>
>>>  0  ?  1  2  1
>
>>>  as output, where ? = "don't care". There must _not_ be a loop over
>>>  columns. I've got a clumsy solution, but I'm sure there's an elegant
>>>  one somewhere?
>
>>>  Cheers,
>
>>>  Mort
>
>>  Hi Mort,
>
>>  It might be less efficient than JD's histogram solution (I didn't check)
>>  but the following also fits the problem specification:
>
>>  x=[    [0,1,1,2,1],$
>>      [0,2,1,1,1],$
>>      [1,0,2,2,1]]
>
>>  voters=(size(x,/dim))[1]
>>  items=(size(x,/dim))[0]
>>  max_label=max(x)+1
>
>>  f=intarr(max_label,items)
>>  ++f[max_label*(indgen(voters*items) / voters)+ $
>>      reform(transpose(x),voters*items)]
>>  junk=max(f,idx,dim=1)
>>  print,idx - max_label*findgen(items)
>
>>  Note that the above solution will also blow up when you end up with
>>  sparse arrays (e.g. if you have someone voting for label 1000000 then f
>>  will end up being an items x 1000000 array even if nobody votes for any
>>  labels between 3 and 1000000).
>
>>  I think all the discussions on finding the mode (either in 1D or nD)
>>  probably pre-dated the ++ operator. It could be that using the
>>  vectorised ++ operator is a better way to do it - I doubt it though,
>>  normally if histogram can do something then histogram will be the best
>>  way! You'd also need to introduce a clumsy offset to deal with negative
>>  selections (Not an issue for you here but would be if finding the mode
>>  in a more general way).
>
>>  It would make David's 1D example from his webpage into something like this:
>
>>  array = [1, 1, 2 , 4, 1, 3, 3, 2, 4, 5, 3, 2, 2, 1, 2, 6,-3]
>>  f=intarr(max(array)-min(array)+1)
>>  f[array-min(array)]++
>>  junk=max(f,idx)
>>  mode=idx + min(array)
>>  print,mode
>
>>  again, with no idea on what would be more efficient. If you're doing
>>  analysis on measurements (typically non-integers) then you'd need to
>>  invoke histogram anyway to bin them before trying to find the mode.
>
>>  Thanks,
>
>>  Allan
>
>  Hi Allan,
>
>  Thanks! Not only have I learned that something called vectorized ++
>  exists, but that it bumps the indexed value multiple times if that index
>  is repeated. Live and learn! But where the hell is all that on the IDL Help?
>
>  Anyway, what I had in mind was trying to program an ensemble image
>  classifier, so that the items are rows of pixels (lots and lots), the
>  labels are land cover classes (contiguous small integers) and the voters
>  are classifiers (e.g. neural networks, also not too many). Hence the
>  wish to avoid the loop over items. I certainly got my money's worth :-)
```

This was big news to me as well, since normally this does *not* work
like this. Example:

```
IDL> x=[0,0]
IDL> x[[1,1]]+=1
IDL> print,x
      0      1
```

This limitation is even called out in HISTOGRAM's help. Now let's try
the new operators:

```
IDL> x=[0,0]
IDL> x[[1,1]]++
IDL> print,x
      0      2
```

I'm not sure if this is intentionally or accidentally inconsistent,
but it is indeed very useful. Allan's solution using this is actually
quite powerful. It easily bested the speed of my HIST_ND version by a
factor of ~5. Not surprising, given that HIST_ND is optimized for
floating point data, and spends lots of unnecessary (for this problem)
time scaling the (already integer) item number.

I was interested to see how a bare HISTOGRAM would perform on straight
integer data compared to the "sensibly" vectorized '++' operator. I
sped up Allan's solution somewhat by avoiding the TRANSPOSE:

```
   s=size(x,/DIMENSIONS)
   f=lonarr(issues,items)
   ++f[x+issues*rebin(lindgen(s[0]),s,/SAMPLE)]
   junk=max(f,vote,dim=1)
   vote mod= issues
```

where 'issues' is the range of the possible "vote" (3 in the original
example). I simplified my example to use only HISTOGRAM (for now as
in Allan's case forgetting about "no preference" or tie votes):

```
   s=size(x,/DIMENSIONS)
   h=reform(histogram(x+issues*rebin(lindgen(s[0]),s,/SAMPLE), $
                    MIN=0,MAX=issues*s[0]-1L),issues,items,/
OVERWRITE)
   m=max(h,DIMENSION=1,mode)
   mode mod= issues
```

The results were almost identical in all cases of input array size and
data range.

This result is not at all specific to this problem. Here's an even
rawer example, doing a very simple integer histogram of a vector:

```
  n=100000L
  cnt=3
  x=long(randomu(sd,n)*cnt)
  t=systime(1)
  h1=histogram(x,MIN=0,MAX=cnt-1L)
  print,'H1:', systime(1)-t

  t=systime(1)
  h2=lonarr(cnt)
  ++h2[x]
  print,'H2:',systime(1)-t

H1:   0.00035190582
H2:   0.00035905838
```

Almost identical! Changing the number and max cnt puts one over the
other by a few % at most, consistent with no difference whatsoever.
In some ways, it's not entirely surprising, given that both operations
are doing the exact same thing.

So the bottom line is that '++f[inds] ' and 'f=histogram(inds)' for
integer 'inds' are essentially identical in results and performance.
For floating point data, you can simply pre-scale to integer (which is
what HIST_ND actually does). Obviously, you don't get reverse
indices, but otherwise you could quite literally rewrite HIST_ND
without once using HISTOGRAM! Even if it's not any faster, the ++
syntax may be clearer for certain problems.

Good find.

JD

Subject: Re: majority voting
Posted by pgrigis on Thu, 12 Feb 2009 19:58:06 GMT
View Forum Message <> Reply to Message

JD Smith wrote:

>
> So the bottom line is that '++f[inds] ' and 'f=histogram(inds)' for
> integer 'inds' are essentially identical in results and performance.
> For floating point data, you can simply pre-scale to integer (which is
> what HIST_ND actually does). Obviously, you don't get reverse
> indices, but otherwise you could quite literally rewrite HIST_ND
> without once using HISTOGRAM! Even if it's not any faster, the ++
> syntax may be clearer for certain problems.
>
> Good find.
>
> JD

And in case any other person a bit more slow on the intake
(like me) is wondering what the heck ++ and -- do in IDL,
they can be found in the documentation under "Mathematical Operators"
(searching for "++" did not return anything useful for me).

Ciao,
Paolo

Subject: Re: majority voting
Posted by David Fanning on Thu, 12 Feb 2009 20:11:58 GMT
View Forum Message <> Reply to Message

Paolo writes:

> And in case any other person a bit more slow on the intake
> (like me) is wondering what the heck ++ and -- do in IDL,
> they can be found in the documentation under "Mathematical Operators"
> (searching for "++" did not return anything useful for me).

Thank you. I have been searching for it off and on all
morning. :-)

There is nothing in the documentation, though, to imply this
is a vectorized operation. In fact, just the opposite for the
discussion on postfix operations. And insights, JD, on
how that can be so?

Cheers,

David

--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: majority voting
Posted by Foldy Lajos on Thu, 12 Feb 2009 20:47:52 GMT
View Forum Message <> Reply to Message

On Thu, 12 Feb 2009, JD Smith wrote:

```
> This was big news to me as well, since normally this does *not* work
> like this.  Example:
>
> IDL> x=[0,0]
> IDL> x[[1,1]]+=1
> IDL> print,x
>       0      1
>
> This limitation is even called out in HISTOGRAM's help.  Now let's try
> the new operators:
>
> IDL> x=[0,0]
> IDL> x[[1,1]]++
```

> IDL> print,x
>   0   2
>
> I'm not sure if this is intentionally or accidentally inconsistent,
> but it is indeed very useful.

It may be useful, but I think X=X+1, X+=1 and X++ should give the same
result for any X. Most programmers use these interchangeably.

regards,
lajos

Subject: Re: majority voting
Posted by JD Smith on Thu, 12 Feb 2009 22:45:18 GMT
View Forum Message <> Reply to Message

On Feb 12, 3:11 pm, David Fanning <n...@dfanning.com> wrote:
> Paolo writes:
>> And in case any other person a bit more slow on the intake
>> (like me) is wondering what the heck ++ and -- do in IDL,
>> they can be found in the documentation under "Mathematical Operators"
>> (searching for "++" did not return anything useful for me).
>
> Thank you. I have been searching for it off and on all
> morning. :-)
>
> There is nothing in the documentation, though, to imply this
> is a vectorized operation. In fact, just the opposite for the
> discussion on postfix operations. And insights, JD, on
> how that can be so?

The surprise isn't that it's vectorized: all of IDL's mathematical,
relational, and bit operators are vectorized. The surprise is in how
it treats repeated indices. I can only guess that ITT regards the old
"no repeats" behavior as undesirable but inalterable due to legacy
code, whereas when ++ and -- were introduced recently, no such legacy
baggage existed, so they were free to improve the behavior (at the
cost of consistency). That said, I know IDL has accumulated lots of
cruft over the years, but I can imagine getting different answers for a
[i]++ and a[i]+=1 might turn some people off.

I also noticed that a[i]+=1, in addition to being plagued by the "no
repeats" issue, is at least 4x slower than a[i]++. As for ++a vs. a++
in expressions, when you don't care about the order of evaluation,
using the former is somewhat faster, since it saves making a temporary
copy of a.

JD

Subject: Re: majority voting
Posted by Allan Whiteford on Fri, 13 Feb 2009 09:31:22 GMT
View Forum Message <> Reply to Message

Fï¿½LDY Lajos wrote:
>
> On Thu, 12 Feb 2009, JD Smith wrote:
>
>> I'm not sure if this is intentionally or accidentally inconsistent,
>> but it is indeed very useful.
>
>
> It may be useful, but I think X=X+1, X+=1 and X++ should give the same
> result for any X. Most programmers use these interchangeably.
>

Lajos, JD,

I think X+=1 should give the same as X++ (which they don't). In a vectorised form of ++ (which most languages don't have) you're saying: "here's a list of things, increment them all by one", it's reasonable to assume that if the same thing is repeated then it will be acted on twice.

X=X+1 more means, in my mind, "evaluate X+1 and set X to the result of that evaluation" - it's clearer in my mind that everything on the right of the = will happen before the set operation is performed.

I do agree, of course, that this difference could lead to confusion.

The issue is that many languages like C explicitly don't allow you to change the same variable twice on one operation. Languages like Perl positively encourage it. I've never seen a general statement about what IDL can and can't do in this regard. I could argue that "++x[[0,0,1]]" doesn't really fall into this category since we're only using one operator.

IDL doesn't have a formal specification like C (the manual doesn't count) nor has anyone ever said (at least that I've heard) that the implementation is the specification (like, say, Perl 5).

I remember the awful day when the result of "help,({a:[1]}).a" changed and all my widget based code collapsed in a heap - I live in fear of that day coming again so share your worries. It's possible that someone will look at ++ and think they can split it across multiple cores - that will be a bad day.

Maybe someone should write to ITTVIS and ask if they intended this as a
feature and if they can guarantee that it will remain. I'll do this and
report back.

Thanks,

Allan

> regards,
> lajos
>

Subject: Re: majority voting
Posted by David Fanning on Sat, 14 Feb 2009 16:44:33 GMT
View Forum Message <> Reply to Message

Allan Whiteford writes:

> I remember the awful day when the result of "help,({a:[1]}).a" changed
> and all my widget based code collapsed in a heap - I live in fear of
> that day coming again so share your worries. It's possible that someone
> will look at ++ and think they can split it across multiple cores - that
> will be a bad day.

I'm still trying to wrap my head around this whole
issue, but I confess I don't understand the first thing
about this paragraph. Do you think you could elaborate
a bit, Allan?

Thanks,

David

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

Subject: Re: majority voting
Posted by Allan Whiteford on Mon, 16 Feb 2009 12:11:05 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

> Allan Whiteford writes:
>
>> I remember the awful day when the result of "help,({a:[1]}).a" changed
>> and all my widget based code collapsed in a heap - I live in fear of
>> that day coming again so share your worries. It's possible that someone
>> will look at ++ and think they can split it across multiple cores - that
>> will be a bad day.
>
> I'm still trying to wrap my head around this whole
> issue, but I confess I don't understand the first thing
> about this paragraph. Do you think you could elaborate
> a bit, Allan?
>
> Thanks,
>
> David
>

David,

Previously (I dunno, sometime before IDL 5.something) if you extracted a
one element array from a structure then it would return as a scalar
rather than a array. This behaviour was clearly a bug but I had stacks
of code which relied upon it. Namely stuff like:

```
widget_control,textwidget,get_value=myval ; myval 1-element strarr
myval = float(myval) ; myval 1-element fltarr
results = {myval:myval}
```

inside an event handler then much later...

```
myval=results.myval ; myval now magically a scalar

answer=myval+[1.,2.,3.]
```

"answer" would end up being a 3-element fltarr because myval had turned
into a scalar. With the improvement answer now ends up as a 1 element
fltarr. Plotting this isn't so useful!

I never actually knew that get_value from a text widget returned a
1-element array in the case of only one line of text because I never
looked at the value explicitly and all subsequent testing meant that I
never noticed the problem in my code.

"help,({a:[1]}).a" demonstrates the problem since it would return either:

```
<Expression>    INT       = Array[1]
```

or

```
<Expression>    INT      =       1
```

depending on your version of IDL.

I have a concern that someone at ITTVIS will one day decide that the
behaviour that I was relying on in the ++ operator can be changed
because they want to make is consistent with += or because they decide
that if they don't treat repeated indices as they are just now they can
parallelise it across multiple CPU cores.

I don't like it when an IDL expression can change in result just due to
a version change of IDL.

Thanks,

Allan