Subject: Re: Sky Falling, etc. : Array substitution + addition with plus-equal (+=)
Posted by JD Smith on Tue, 13 Mar 2007 17:46:24 GMT

View Forum Message <> Reply to Message

On Mon, 12 Mar 2007 14:17:57 -0700, Ed Hyer wrote:

> Tentatively filed under "The Sky Is Falling!," and I hope it will be
> resolved in the same fashion.

```
> Here is another case, with results somewhat different
> IDL> test=fltarr(2,2,4)
> IDL> testadd=fltarr(2,2,2)
> IDL> testadd[0,0,*]=1
> IDL> print,total(test),total(testadd)
>       0.00000      2.00000
> IDL> test[0,0,1] += testadd
> IDL> print,total(test),total(testadd)
>       2.00000      2.00000
> IDL> print,test
>       0.00000      0.00000
>       0.00000      0.00000
>
>       1.00000      0.00000
>       0.00000      0.00000
>
>       1.00000      0.00000
>       0.00000      0.00000
>
>       0.00000      0.00000
>       0.00000      0.00000                ; So far, so good
> IDL> test[0,0,1] += testadd                ; Now, let's add again
> IDL> print,total(test),total(testadd)
>       10.0000      2.00000                ; Pardon the
> expression, WTF?
> IDL> print,test
>       0.00000      0.00000
>       0.00000      0.00000
>
>       2.00000      1.00000
>       1.00000      1.00000
>
>       2.00000      1.00000
>       1.00000      1.00000
>
>       0.00000      0.00000
>       0.00000      0.00000                ; Not even JD Smith
> expected _that_.
```

Sure I did. In the first case, you are adding test[0,0,1], i.e. "0", to 'testadd', then setting the entire resulting 2x2x2 array en masse into 'test' at offset [0,0,1]. In the second case, you are adding test[0,0,1], i.e. "1", to 'testadd', resulting in:

```
 2 1
 1 1

 2 1
 1 1
```

and then setting this resulting 2x2x2 array into 'test' at offset [0,0,1].

What's surprising and perhaps nonintuitive here is the ambiguity between a single array index on the LHS of an assignment, and a single array index on the RHS. When on the RHS, an array element is treated simply as a scalar, and so is threaded across every element of any other arrays in the RHS calculation. When on the LHS, a single array index is treated as an *offset* into an array, into which to set the RHS array if, and only if, it "fits". How do you determine if it fits? Think of cutting out different sized rectangles of colored paper and inserting one into the other at some arbitrary offset. You can easily arrange for it *not* to fit (in the geometric sense):

```
IDL> testadd=fltarr(2,3,2)
IDL> test[0,0,1]+=testadd
% Out of range subscript encountered: TEST.
% Execution halted at: $MAIN$
```

Of these two behaviors (array element as scalar vs. array element as offset position for setting new array), 'foo[x,y,z]+=' actually invokes *both*, since it expands to 'f[x,y,z] = foo[x,y,z] + ', i.e. an indexed array on both sides of the assignment.

Also note that there is a bit of an "escape clause" for the "does it fit" geometric argument, in the sense that a 1D vector is treated as a simple block of memory to copy in linearly, index by index, if the LHS indexing is single element, e.g.:

```
IDL> testadd=reform(testadd,product(size(testadd,/DIMENSIONS)))
IDL> testadd[*]=1
IDL> test[0]+=testadd
```

Here we've added 12 elements in memory order into test. This is different from the notionally equivalent:

```
IDL> test[0,0,0]+=testadd
```

```
% Out of range subscript encountered: TEST.
% Execution halted at: $MAIN$
```

As you see, as soon as you specify a multi-dimensional index on the LHS or multi-dimensional array on the RHS, the geometry argument comes into play. In either of these cases, it will check for a "fit", and then thread along the specified dimension(s) in the LHS index until it runs out of room:

```
IDL> test=fltarr(2,2,4)
IDL> testadd=findgen(2,2,4)+100
IDL> test[0]+=testadd
IDL> print,test
     100.000      101.000
     0.00000      0.00000

     0.00000      0.00000
     0.00000      0.00000

     0.00000      0.00000
     0.00000      0.00000

     0.00000      0.00000
     0.00000      0.00000
```

Notice that data along dimensions higher than specified in the LHS index are simply dropped. Note also that it's *not* sufficient for the array to "fit" just along the considered dimension(s). It must fit in its entirety, in *all* dimensions:

```
IDL> test=fltarr(2,2,4)
IDL> testadd=findgen(2,2,5)+100
IDL> test[0]+=testadd
% Out of range subscript encountered: TEST.
% Execution halted at: $MAIN$
```

This error occurs even though, as we saw above, it was only going to assign the first two elements (100 and 101). How about two dimensions indexed on the LHS?

```
IDL> test=fltarr(2,2,4)
IDL> testadd=findgen(2,2,4)+100
IDL> test[0,0]+=testadd
IDL> print,test
     100.000      101.000
     102.000      103.000

     0.00000      0.00000
```

```
      0.00000      0.00000

      0.00000      0.00000
      0.00000      0.00000

      0.00000      0.00000
      0.00000      0.00000
```

and so on:

```
IDL> test=fltarr(2,2,4)
IDL> test[0,0,0]+=testadd
IDL> print,test
      100.000      101.000
      102.000      103.000

      104.000      105.000
      106.000      107.000

      108.000      109.000
      110.000      111.000

      112.000      113.000
      114.000      115.000
```

You can even add more fake "shallow" trailing dimensions if you are so inclined, for instance if you don't know in advance what the number of dimensions will be, but know you want to insert the RHS array in a specific position:

```
IDL> testadd=findgen(2,2,2)+100
IDL> test=fltarr(3,2,4)
IDL> test[0,0,1,0,0,0,0,0]+=testadd
IDL> test=fltarr(3,2,3,2)
IDL> test[0,0,1,0,0,0,0,0]+=testadd
```

One additional point is worth mentioning: when setting large arrays, the "offset" method of specifying a single index on the LHS is much faster than the '*' method of building the full index list to match up the dimensions of left and right-hand side arrays:

```
IDL> testadd=randomu(sd,100,100,100,5)
IDL> test=fltarr(200,100,100,10)
IDL> t=systime(1) & test[100,0,0,0]=testadd & print,systime(1)-t
    0.067754984
IDL> t=systime(1) & test[100:*,*,*,0:4]=testadd & print,systime(1)-t
     0.21291494
```

Not only is it a simpler notation, it avoid the construction of large, memory-hogging index arrays that the "*" method requires, and is thus usually much faster. This is why you may have seen lots of assignment of arrays to single array indices in optimized code.

JD