

Subject: Re: zero-padding an array of arbitrary dimensionality (replacing execute in vm)

Posted by [Vince Hradil](#) on Fri, 20 Jul 2007 13:34:11 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Jul 19, 9:14 pm, JD Smith <jdsm...@as.arizona.edu> wrote:

> On Thu, 19 Jul 2007 08:51:58 -0700, hradilv wrote:

>> I would like to zero-pad an array programmatically without knowing in

>> advance what the dimensionality is of the array.

>

> Position [1,1,1,...] can be expressed in 1D as:

>

> ind=total(product((size(data,/dimensions))[0:1],/CUMULATIVE,/PRESERVE_TYPE),\$

> /PRESERVE_TYPE) + 1

>

> Some other arbitrary position p=[p0,p1,p2,...,pn] can also be expressed:

>

> d=shift(product(size(data,/dimensions),/CUMULATIVE,/PRESERVE_TYPE),1)

> d[0]=1

> ind_p=total(p*d,/PRESERVE_TYPE)

>

> Note that this is the one-dimensional position within the array, so it

> can be used to access an arbitrary element without resorting to EXECUTE.

>

> That said, assigning an array of greater than one dimension to this

> position will *not* do what you intend. Only when using an explicit

> list of indices on the left-hand side of an assignment (e.g.

> array[x,y,z,...]=sub_array) will IDL invoke the rules to preserve the

> structure of the inserted array. If a "one-dimensional" offset index is

> used, only the first dimension will be preserved in this way, and the rest

> will be silently ignored. Whatever to do?

>

> To solve this, you need instead to calculate the indices of the full

> sub-array itself, offset within the larger target array at starting

> position p=[p0,p1,p2,...,pn]. This is a more general problem than zero

> padding. Essentially you are computing an array of indices of an

> arbitrarily sized rectangular (cube, hyper-cubic, etc.) "chunk" within a

> larger array, of a given size and offset. This is a bit more complicated

> than the above, but conceptually similar:

>

> ;; Compute indices of the "chunk"

> ss=size(sub_array,/DIMENSIONS)

> l=lindgen(ss)

> inds=make_array(VALUE=p[0],ss)

> d=product(size(data,/dimensions),/CUMULATIVE,/PRESERVE_TYPE)

> ds=product(ss,/CUMULATIVE,/PRESERVE_TYPE)

> for i=n_elements(d)-2,0,-1 do begin

> inds+=(p[i+1]+l/ds[i])*d[i]

```
> l-=l/ds[i]*ds[i]
> endfor
> inds+=l
>
> ;; Apply them
> data[inds]=sub_array
>
> This is essentially the calculation IDL does for you when you use the
> subscripted assignment of unlike arrays. Too bad it doesn't expose that
> functionality via some function. You might like to check first that the
> sub-array can actually fit within the larger array, and warn or truncate
> if not (for your case of padding with zeroes, this isn't a problem).
>
> For those interested, this method works on the highest dimension
> downward, (e.g. z, y, x in the 3D case), removing that dimension's
> signature after it has been applied. E.g. a z position of 4 in the
> sub_array, inserted at a z offset offset of 2, should have 6*(nx*ny)
> added. It also demonstrates the trick HIST_ND uses to accomplish a very
> similar task.
>
> JD
```

Thanks everyone. Especially Mike and JD - this thread is definitely a "keeper" for me.
