Subject: Re: Arrays of Structures
Posted by Mick Brooks on Fri, 09 Feb 2007 10:13:56 GMT
View Forum Message <> Reply to Message

On Feb 8, 8:42 pm, JD Smith <jdsm...@as.arizona.edu> wrote:

> The problem is that structs.a is not an array until the ".a" part has been
> applied.

<snip>

> Creating "structs.a" would cause a large temporary
> array to be created, only to finally index a single element. Compare the
> memory usage of the following:
>
> IDL> struct=replicate({a:lindgen(100,100,100)},100)
> IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"
> 382.472MB
> IDL> val=struct[10].a[4]
> IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
> 0.00000MB extra
> IDL> val2=(struct.a)[10,4]
> IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
> 381.470MB extra
>
> So the latter method first creates a temporary variable of size
> 100,100,100,100, and then pulls a single element out of it. Not
> exactly good form.

It seems that this is a reason to prefer my first "workaround" to my second one, but it doesn't tell us anything about my problem ("unholy notation" - I like that), which here would be represented by val3=struct.a[10,4] i.e. leaving off the temporary-creating parentheses.

If I try, I get the following:

```
IDL> struct=replicate({a:lindgen(100,100,100)},100)
IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"
      385.836MB
IDL> val=struct[10].a[4]
IDL> print,((n=memory(/HIGHWATER))-m)/1024/1024.,"MB Extra"
     -3.81445MB Extra
IDL> val3=struct.a[10,4]
IDL> print,(memory(/HIGHWATER)-n)/1024/1024.,"MB Extra"
      0.00000MB Extra
IDL> HELP, val, val3
VAL             LONG      =           4
```

```
VAL3            LONG      = Array[100]
IDL> PRINT, val3
        410  [+ another 99 of the same]
```

The problem case doesn't use any extra memory (great!), but it gives a different result (boo!).

Bob's and Mike's posts made me think that my original "structs.a" has a leading shallow dimension, but that IDL elides it when evaluating it.
So,

```
IDL> structs = replicate({a:1},50)
IDL> HELP, structs.a[0]
<Expression>    INT       = Array[50]
```

works, because our array subscript is within bounds on our leading shallow dimension, but

```
IDL> HELP, structs.a[1]
% Subscript range values of the form low:high must be >= 0, < size,
with low
  <= high: <No name>.
% Execution halted at: $MAIN$
```

is out of range.

If we ask for everything from the array that is structs.a

```
IDL> HELP, structs.a[*]
<Expression>    INT       = Array[1, 50]
```

we see the entire thing, leading shallow dimension included.

However, if we simply evaluate structs.a, IDL drops the leading dimension, like so:

```
IDL> HELP, structs.a
<Expression>    INT       = Array[50]
```

Creating a temporary with parentheses also causes IDL to drop the leading dimension too:

```
IDL> HELP, (structs.a)[*]
<Expression>    INT       = Array[50]
```

Does this make any more sense to anyone? I still can't use this idea to work out what's going on
with val3 above though...

Thanks again for everybody's help,

Mick