Subject: Re: Reducing an array.
Posted by Craig Markwardt on Mon, 30 Sep 2002 23:17:39 GMT
View Forum Message <> Reply to Message

"Joe" <foosej@hotmail.com> writes:

> Hi- I'm somewhat new to IDL and was wondering what the most effiecient way
> is to 'OR' all the elements of an array together resulting in a scalar
> value. I'm hoping IDL has a built-in way of doing this rather than using a
> FOR-LOOP. Similar to how IDL has the TOTAL function which sums all the
> elements of an array together. I've used other languagues which allow you
> to 'reduce' arrays to a scalar using an arbitrary function (i.e. Python's
> reduce function).
>
> What I am doing is taking a lot of integer data which is either 0's or 1's
> and compressing it into the bits of 64-bit unsigned integers. Here is a bit
> of sample code:
>
> data = [1,0,0,0,1,1,1,0,1,0,1,0,0, ... , 0, 1, 0, 1] ; bunch of data, assume
> # of elements is multiple of 64
> shifts = reverse(indgen(n_elements(data))) MOD 64
> compressed_data = ishft(data,shifts)
> ; here is where I want to take the compressed_data array and make it into a
> ; bunch (n_elements(data)/64, to be exact) of unsigned 64-bit integers by
> OR'ing
> ; every 64 elements of compressed data togeter

In this case you can use TOTAL() directly. First you REFORM() your
data into a 2-d array, 64xN, then then total the 1st dimension. This
works because each of your values has only one data bit set, so
summing and ORing are equivalent.

```
  compressed_data = reform(compressed_data, 64, n_elements(compressed_data)/64)
  result = total(compressed_data, 1)
```

That's it! For JD, I could have combined both statements onto one
line, but this is more readable.

Craig

--
 -- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 -- --------------

Subject: Re: Reducing an array.
Posted by MKatz843 on Tue, 01 Oct 2002 06:15:46 GMT

View Forum Message <> Reply to Message

In order to perform an OR on 64 bits stored in an array, could you do
something like this:

```
a = [0,0,0,1,0,1,0,1,1]
b = total(a) GT 0
```

Then, if any of the elements of a are non-zero, b will be TRUE, or 1.
At least I hope that's what you meant by OR.

Likewise AND would be

```
c = total(a) EQ n_elements(a)
```

M. Katz

Subject: Re: Reducing an array.
Posted by Craig Markwardt on Tue, 01 Oct 2002 14:25:42 GMT

View Forum Message <> Reply to Message

MKatz843@onebox.com (M. Katz) writes:

```
>  In order to perform an OR on 64 bits stored in an array, could you do
>  something like this:
>
>  a = [0,0,0,1,0,1,0,1,1]
>  b = total(a) GT 0
>
>  Then, if any of the elements of a are non-zero, b will be TRUE, or 1.
>  At least I hope that's what you meant by OR.
>
>  Likewise AND would be
>
>  c = total(a) EQ n_elements(a)
```

The original poster was asking for a BITWISE "and" operation, not a
logical "and" operation. He was interested in all the bits of the
result, not just whether any bit was set.

Incidentally, "M"'s formulation is pretty much exactly what I use in
my routine CMAPPLY, which applies selected operations, like AND and
OR, to arrays.

Craig

--
 -- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 -- --------------

Subject: Re: Reducing an array.
Posted by Dick Jackson on Tue, 01 Oct 2002 20:53:26 GMT
View Forum Message <> Reply to Message

"Craig Markwardt" <craigmnet@cow.physics.wisc.edu> wrote in message
news:on65wnysmk.fsf@cow.physics.wisc.edu...
>
> "Joe" <foosej@hotmail.com> writes:
>
>> Hi- I'm somewhat new to IDL and was wondering what the most
effiecient way
>> is to 'OR' all the elements of an array together resulting in a
scalar
>> value. I'm hoping IDL has a built-in way of doing this rather than
using a
>> FOR-LOOP. Similar to how IDL has the TOTAL function which sums all
the
>> elements of an array together. I've used other languages which
allow you
>> to 'reduce' arrays to a scalar using an arbitrary function (i.e.
Python's
>> reduce function).
>>
>> What I am doing is taking a lot of integer data which is either 0's
or 1's
>> and compressing it into the bits of 64-bit unsigned integers. Here
is a bit
>> of sample code:
>>
>> data = [1,0,0,0,1,1,1,0,1,0,1,0,0, ... , 0, 1, 0, 1] ; bunch of
data, assume
>> # of elements is multiple of 64
>> shifts = reverse(indgen(n_elements(data))) MOD 64
>> compressed_data = ishft(data,shifts)
>> ; here is where I want to take the compressed_data array and make it
into a
>> ; bunch (n_elements(data)/64, to be exact) of unsigned 64-bit
integers by
>> OR'ing
>> ; every 64 elements of compressed data togeter

>
> In this case you can use TOTAL() directly. First you REFORM() your
> data into a 2-d array, 64xN, then then total the 1st dimension. This
> works because each of your values has only one data bit set, so
> summing and ORing are equivalent.
>
>   compressed_data = reform(compressed_data, 64,
n_elements(compressed_data)/64)
>   result = total(compressed_data, 1)
>
> That's it! For JD, I could have combined both statements onto one
> line, but this is more readable.

There's one problem with this, in that Total() returns a Double result
at best (with the /Double keyword), but Joe wanted 64-bit integers. A
64-bit Double value with some bits used as exponent cannot represent as
many distinct values as the 64-bit integer, so we will lose information
here.

Looks to me like this all has to be done in 64-bit integers. I'm sorry I
can't find a *really* elegant solution for you right now, but if your
data array is very large, then a single loop over 64 columns might not
be too inefficient. Here's my best attempt (it does 100000 ints in 2.2
seconds here, fast enough?):

```
=====

data = RandomU(seed, 640) LT 0.5        ; Byte array of [0|1] values

nInts = N_Elements(data)/64
data2D = Reform(data, 64, nInts)
Print, 'binary:'
Print, data2D                           ; Show [0|1] values

result = Replicate(0ULL, 1, nInts)      ; column array

;;   If byte 0 is your high-order bit:
FOR i=0, 63 DO result = result OR ([0LL, 2ULL^(63-i)])[data2D[i, *]]
;;                         this lookup is faster than
multiplying:
;                          data2D[i, *] * 2ULL^(63-i)

;;   If byte 0 is your low-order bit:
;FOR i=0, 63 DO result = result OR ([0LL, 2ULL^i])[data2D[i, *]]

Print, 'hex:'
Print, result, Format='(Z)'             ; Show hex values (will
correspond
```

```
                              ; to [0|1] values above
Print, 'decimal:'
Print, result                 ; Show in base 10
```

=====

and one sample from the output:

```
binary:

1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0

1 1 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 1 0 0 1 0 1

1 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1

hex:

FF10ADDC5796B157

decimal:

18379381241172898135
```

Hope this helps!

Cheers,
--
-Dick

Dick Jackson / dick@d-jackson.com
D-Jackson Software Consulting / http://www.d-jackson.com
Calgary, Alberta, Canada / +1-403-242-7398 / Fax: 241-7392