Subject: Re: Array Subscripting Memory Usage (watch out!)
Posted by Dick Jackson on Fri, 04 Oct 2002 20:13:14 GMT
View Forum Message <> Reply to Message

"Reimar Bauer" <R.Bauer@fz-juelich.de> wrote in message
news:ankqrb$ce3e$1@zam602.zam.kfa-juelich.de...

```
> Dick Jackson wrote:
>
>> I was surprised to find how much memory is used during access to a
>> subset of an array. I ran this, which makes a 1000x1000 array, and
>> accesses a subset of it using an array of subscripts:
>>
>> a = bindgen(1000, 1000)
>> subscripts = Long(RandomU(seed, 500)*1000)
>> baseMem = (memory())[0]
>> help, a[subscripts, *]
>> highWaterMem = (memory())[3]
>> Print, 'Memory used during access: ', highWaterMem-baseMem
>>
>> IDL> .GO
>> <Expression>    BYTE      = Array[500, 1000]
>> Memory used during access:      2500076
>
> Dear all, especially David please can you put this on your tips pages.
>
> My last posting wasn't enough detailed to describe what is going on.
> So here is the next one.
>
> a = bindgen(1000, 1000)
> then size of a is 1 MB
>
> If you now want to do something by subscribts the subscribts never
passed by
> reference they are passed by value of type of the indices.
>
> For example:
> help,a[indgen(10)] takes during execution 50*2+ 50 bytes.
>
> and
> help,a[indgen(10,/long)] takes during execution 50*4 +50 bytes
>
>
> Now what happens if we set a *. This means nothing else as from start
to end
> and it is an alias for an index array of type long.
>
> help,a[subscripts,*]
>
```

> is the same as help, a[subscripts[0:499],0:999]
>
> this means the subscripts of a are
> 500 x 1000 x 4 and this is 2 MB
>
> plus the result after the index operation in byte of 500 x 1000 which
gives
> 0.5MB is the measured 2.5 MB.
>
>
> During the operation you need this amount of memory.
> All the array functionalities of idl goes by cost of memory.

Very clearly put. This is exactly what seems to happen, but I don't
think there is any documentation that explains that this is the case.
There are certainly other ways that this could be handled behind the
scenes that don't use so much memory, possibly at the expense of time.
<sound muffled by talking through hat>
Yes, you have to get at each element at the intersection of the n sets
of indices (for an n-dim array) in order to fill the result array, but
the sets are independent, so the calculation of indices should be
trivial. I could imagine, for each dimension's set of indices, a vector
giving the offset required for each element, then the offset for each
resulting array element is just a sum of n offsets. If I were writing
this process in *IDL*, I might use a large subscript array in order to
do a single lookup operation and avoid calculations in loops, but this
is IDL internals, presumably written in C!

> Now something about byte. The byte or string isn't handled the same as
the
> others.
> I don't know why. May be they are internal first defined in something
> different.
>
>
> That's for a new discussion:
>
> BYTE:
>
> .reset
> s=memory()&x=indgen(20,/byte)&e=memory()& print,e[3]-s[0]
>   188
> .reset
> s=memory()&x=indgen(10,/byte)&e=memory()& print,e[3]-s[0]
>   180
>
> The difference is 8 normally it should be 10 ?!:-(
> where are the two bytes?

I might guess that memory on your system is allocated in 4-byte words,
we are always seeing multiples of 4 here.

```
>  .reset
>  s=memory()&x=indgen(2,/byte)&e=memory()& print,e[3]-s[0]
>       172
```
> And 2 byte less then 4 byte costs the same memory as 4 byte.
> Did this mean 2 byte costs nothing?

The second two cost nothing, the first two cost four! :-)

```
>  STRING:
>  .reset
>  s=memory()&x=indgen(20,/string)&e=memory()& print,e[3]-s[0]
>       748
>  .reset
>  s=memory()&x=indgen(10,/string)&e=memory()& print,e[3]-s[0]
>       458
```
> The difference is 290. How to explain, the string length is 12 of each
> element. Normally I believe 1 char is 1 byte. So first result should
be
> 12 byte * 20 = 240 and the second one 120. The differnce should be
120.
> So I have no explanation why it needs so much more memory. Any ideas?

I think each string would be terminated by a null character, making it
13 bytes each, then increase that to 16 because of 4-byte words, but
that still doesn't match... I give up at this point!