Subject: Re: speed of n_elements
Posted by Pavel Romashkin on Wed, 03 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

> * I can't repeat your experience on Sun or Linux. On both those
> machines n_elements(data.flag) is much slower than n_elements(data),
> at least in a loop.

Thank you for the test, Craig. I repeated it and n_elements(data) was way faster. But I was certain that I saw what i saw, so I reconstructed the entire data system in the same way it was in my program. DATA in my program is itself passed as a field of State structure inside a widget tree. Try the following code:

```
**************
pro test, a

start=systime(1)
for i=0, 100 do temp = n_elements(a.data)
print, systime(1)-start

start=systime(1)
for i=0, 100 do temp = n_elements(a.data.flag)
print, systime(1)-start

end
************* ; Now, the outputs:
IDL> a=fltarr(1000)
IDL> b={a:a, b:a, c:a, d:a, flag:0L, name:'', other:0.0}
IDL> c=replicate(b, 500)
% Compiled module: TEST.
IDL> k={a:0L, data:c}
IDL> test,k
       33.133333
     0.033333302
```

> * I've always found that putting large data arrays into structures is
> a big loser. In my experience it's slow to create such structures
> and slow to extract the fields later.

I agree it is not nearly as neat as pointers. I don't find it noticeably slow, at least with my array sizes. However, there are things that are really advantageous in placing arrays directly as structure fields. For instance, I can shift all arrays in the array of structures in one line:

```
data.(k) = shift(data.(k), n_pts)
```

while with pointers this does not work. In my opinion, the only drawback in

using pointers and (built-in) objects is that they take away from you some of the tremendous advantages that IDL has in handling arrays.

Cheers,
Pavel

P.S. Not to mention the humiliating use of HEAP_GC after a crash of an app full of pointers :-)
