Subject: Re: Pass by value and performance
Posted by Paolo Grigis on Fri, 16 Dec 2005 11:08:08 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
> This isn't correct. De-referenced pointer variables (aka "heap"
> variables) are passed by reference, just like regular variables (which
> they are, really). E.g. in Ken's original example:
>
> result = INTERPOLATE(*data.array, x, y, z) ; by reference
>
> would indeed pass the pointer heap variable by reference and not by
> value. As such it would be much faster (for large arrays) than
> INTERPOLATE(data.array,x,y,z), which would require copying the full
> array to a local variable, and would be equivalent to a simple
> INTERPOLATE(array,x,y,z).

Since we are on the subject of performance, there's nothing like
a little benchmark to bring some light to shine upon the issue...

Let's try this (using rebin for simplicity):

Benchmark 1:

```
;initialize large arrays of data
N=2L^27

data={a:lindgen(N),b:ptr_new(lindgen(N))}

c=lonarr(N/2)

nrounds=10
nrebins=10

timevar=fltarr(nrounds)
timeptr=fltarr(nrounds)

;do benchmark
.run
FOR j=0,nrounds-1 DO BEGIN

    print,'Now doing round '+strtrim(string(j+1),2)

    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(data.a,N/2)
    tend=systime(1)
```

```
    timevar[j]=tend-tstart

    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(*data.b,N/2)
    tend=systime(1)
    timeptr[j]=tend-tstart
ENDFOR
end
```

This compares the data.array vs. *data.array performance.
As correctly claimed by JD, there is indeed a difference
between the two approaches:

```
;"data.array" case
IDL> print,timevar
      32.7094      34.0300      34.7631      33.0446      33.9109
      34.2302      34.2145      33.8960      34.2056      34.2010
;"*data.array" case
IDL> print,timeptr
      18.1812      18.4961      18.5838      17.8924      18.4376
      18.4548      18.0502      18.3959      18.7219      18.0366
```

However, if we don't have structures, is there a difference
between passing pointers and regular variables? How does
this compare with the structure case?

Benchmark 2:

```
;initialize large arrays of data
N=2L^27

a=lindgen(N)
b=ptr_new(lindgen(N))

c=lonarr(N/2)

nrounds=10
nrebins=10

timevar=fltarr(nrounds)
timeptr=fltarr(nrounds)

;do benchmark
```

```
.run
FOR j=0,nrounds-1 DO BEGIN

    print,'Now doing round '+strtrim(string(j+1),2)

    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(a,N/2)
    tend=systime(1)
    timevar[j]=tend-tstart

    tstart=systime(1)
    FOR i=0,nrebins DO c=rebin(*b,N/2)
    tend=systime(1)
    timeptr[j]=tend-tstart
ENDFOR
end
```

Here we get:

```
;"array" case
IDL> print,timevar
      17.6973      17.6340      17.6237      17.7584      17.6499
      17.7070      17.6797      17.6858      17.6515      17.6766
;"*array" case
IDL> print,timeptr
      17.6719      17.7895      17.6816      17.6413      17.7822
      17.6556      18.0883      17.6746      17.6907      18.1122
```

No difference (motto: "dereferenced pointer behave like normal variables" thus both passed by reference), and the performance is the same as the fastet of the previous case.

Summarizing: rebin(*data.array) is indeed faster than rebin(data.array), but rebin(*data.array), rebin(array) and rebin(*array) have all the same speed.

Again, JD was indeed absolutely right. I just thought it was nice to have an experimental confirmation... and it helped me to grasp the issue.

Cheers,
Paolo

>

> As pointed out in the pointer tutorial
> (http://www.dfanning.com/misc_tips/pointers.html), there is no
> difference between pointer heap variables and ordinary variables,
> except in how you access them. Of course, that also means that a
> structure member (or array element, etc.) of a dereferenced pointer
> variable is (just like a member of an ordinary variable), still passed
> by value:
>
> result = INTERPOLATE((*data).array, x, y, z) ; by value
>
> Here `data' is a pointer to a structure with member "array", which is
> passed here by value.
>
> This equivalence also means that standard IDL variable tricks, like
> re-assigning the memory contents of one variable to another without
> copying, work just fine for pointer heap variables (and in between
> plain old variables and pointer heap variables).
>
> JD
>
