Subject: Storing data in an array inside a structure MUCH slower in IDL 8.2.2
Posted by Mark[1] on Thu, 21 Feb 2013 04:58:36 GMT
View Forum Message <> Reply to Message

The routine below creates up a long-integer array with one million elements, either as a variable or as a tag embedded in a structure, then loads data values one at a time in a loop.

On IDL 8.1.0 (Windows, 64-bit) loading the data into the bare array took 0.047s (who said IDL loops were slow?) and loading the same data into the structure took 0.465s, as the following session log indicates:

IDL> mgh_test_structure, USE_STRUCTURE=0
IDL 8.1: loading 1000000 long-integer values into bare array took 0.0470 s
IDL> mgh_test_structure, USE_STRUCTURE=1
IDL 8.1: loading 1000000 long-integer values into structure-field array took 0.4800 s

On IDL 8.2.2, loading the data into the bare array took 0.050s and loading it into the structure took so ridiculously long that I aborted it and tried again with a smaller number, eventually finding that 10000 (10^4) elements took 0.062s and 100000 (10^5) took 7.56s

IDL> mgh_test_structure, USE_STRUCTURE=0
IDL 8.2.2: loading 1000000 long-integer values into bare array took 0.0500 s
IDL> mgh_test_structure, USE_STRUCTURE=1, N_DATA=10000
IDL 8.2.2: loading 10000 long-integer values into structure-field array took 0.0620 s
IDL> mgh_test_structure, USE_STRUCTURE=1, N_DATA=100000
IDL 8.2.2: loading 100000 long-integer values into structure-field array took 7.6540 s

The super-linear slowdown with IDL 8.2.2 suggests that a temporary copy of the data is being produced every time the data is accessed. Why would this be? Is there a syntax that will avoid it?

I ran into this problem with some actual code of mine and rewrote it so that the structure is built after the data are loaded, not before. (This is probably not a bad thing to do in any case.) I wonder if all such cases can be avoided with a simple rewrite?

---------- ------

```
pro mgh_test_structure, N_DATA=n_data, USE_STRUCTURE=use_structure

  compile_opt DEFINT32
  compile_opt STRICTARR
  compile_opt STRICTARRSUBS
  compile_opt LOGICAL_PREDICATE

  if n_elements(n_data) eq 0 then n_data = 1000000

  if keyword_set(use_structure) then begin
    my_struct = {data: lonarr(n_data)}
```

```
  t0 = systime(1)
  for i=0,n_data-1 do my_struct.data[i] = i
  t1 = systime(1)
  fmt = '(%"IDL %s: loading %d long-integer values into ' + $
        'structure-field array took %0.4f s")'
  print, FORMAT=fmt, !version.release, n_data, t1-t0
 endif else begin
  my_data = lonarr(n_data)
  t0 = systime(1)
  for i=0,n_data-1 do my_data[i] = i
  t1 = systime(1)
  fmt = '(%"IDL %s: loading %d long-integer values into ' + $
        'bare array took %0.4f s")'
  print, FORMAT=fmt, !version.release, n_data, t1-t0
 endelse

end
```