Subject: Re: Optimizing memory usage on Windows NT
Posted by Peter Mason on Fri, 01 Aug 1997 07:00:00 GMT
View Forum Message <> Reply to Message

On Wed, 30 Jul 1997, Michael A. Wirth wrote:
> I am running IDL5 on a new Pentium 166 with 64MB RAM, doing some 2D but
> also 3D medical image analysis and was wondering if anyone knew of any
> ways
> to optimize memory usage, ie. allocating memory to IDL, manipulating
> large 3D
> arrays (eg. 256x256x50)? Any help would be greatly appreciated.

Consider yourself lucky that you aren't using Unix!
As far as I can tell, IDL on Windows (win95, at least) is free of a nasty
curse that we see on Unix: memory allocated doesn't get given back to the
system until IDL exits, and doesn't necessarily get reused optimally by
IDL after it gets "freed".   There's a section on this in the FAQ.
(I was told that this behaviour - efficient reusing "freed" memory - would be
much improved in IDL5, but I haven't checked it out myself.)
This is something to bear in mind, especially if your app will be run on Unix
sometime - try to be cirumspect in creating and destroying large variables.
(Note that this may still be a problem under Windows - Windows' memory / page
file might be getting fragmented by all that allocating and freeing.   But I'm
not sure how this stuff works, and I don't have the strength to pull that tooth
out of the maw of the Microsoft documentation just now.)

Here's a hotch-potch collection of tips.   Some come from the docs, some from
far and wide, some are only opinions or guesses.
(Some of these tips relate to the Unix / fragmentation problem.)

Beware of mixed types.   IDL will re-type variables to the highest type in
the RHS of an expression.   This means that new variables will be created
for "lower-type" variables, while the original variables may still exist (for
a time, at least).   This can be pretty expensive.
e.g., A=BYTARR(1024L,1024L,64L) &A(0,0,0)=A+1
Here, "1" is an integer type.   I think that an integer copy of A will get
created, get 1 added to itself, then converted back to byte to overwrite A.
Ouch.
(We're forcing the conversion back to byte by using () on the LHS.   Without
it, A would become type int.)

Make good use of TEMPORARY().   TEMPORARY() can be used anytime to delete a
variable that you're accessing for the "last time", not just for when a
variable appears on both the LHS and RHS of an expression.

Avoid using "*" on the LHS of an expression if at all possible.   It is
slow and apparently wasteful of memory.   e.g., avoid A(*,*,2)=...

Rather use absolute indices (e.g., A(0,0,2)=...) and, if necessary, use
REFORM() on the RHS to get the dimensions to match the LHS.   (And use
REFORM's /OVERWRITE switch.)

Avoid using multiple instances of the "array casting" operator [].
I am given to believe that it is inefficient (for multiple ['s).
e.g., Avoid a=[[b],[c]].   Rather make a new variable, and "insert" the
components into it, using REFORM if necessary.
e.g., B=findgen(20,20) &C=findgen(30,20)
  Want: A=[[B][C]]
  Use: A=fltarr(20,50,/nozero) &A(0,0)=B &A(0,20)=C
  (And, of course, use temporary() on B & C if they're no longer required!)

Don't use WHERE() unless you really need it.   It can create a huge LONARR.
(Indeed, does it work in 2 passes to first find out how large its result
array must be, or does it work in 1 pass, assuming the worst and allocatng
for the largest possible result and then triming the result down when it's
done?)
e.g., Sometimes people use WHERE for things like WHERE(a EQ -10000.) and
then just take the first instance thereof.   You can get this index with
MIN(a EQ -10000.,inx).   (BTW, Note that "a EQ -10000." is a BYTARR.)
e.g., Sometimes people use WHERE to set tags, as in:
  t=BYTARR(n) &t(WHERE(a GE 400.))=255b
This can be accomplished with: t=(a GE 400.)*255b

Avoid reallocating arrays if you can.   e.g., If you implement a "working
array", like an array to hold a line of image data in an image-processing
routine that processes 1 line at a time (in a loop), allocate it once and
insert data into it using the A(x,y)= style, rather than always reallocating
with A=.   For very big arrays, it may be worth your while to allocate them
once and SAVE THEM (in handles or such), and only reallocate them if
their sizes must change.

Use the /NO_COPY switch with uvals and handles when dealing with variables
of any significant size.   This means you have to match every "get" with a
"set", but it's usually worthwhile.

Be careful with structures.
I think that it might not be good practice to have large arrays as structure
members - you can't change their dimensions, and I don't trust IDL not to
make (invisible) copies when such members are used in the RHS of an
expression.
Also, passing a structure member to a function/procedure causes the member
to be copied into a temporary var and passed by value.   Passing the whole
structure (or an array on its own) goes by reference, I believe (much more
efficient).

Try to avoid common blocks.   If you really need large global variables, put

them in HANDLES, with the handle IDs in a common block.
(There are programming techniques that make common blocks unnecessary.)

Think about what IDl is doing in array expressions and look for shortcuts.
(But think in terms of memory efficiency, not elegance!)

Peter Mason