

Re: Fortran memory allocation (stack/heap) issues

Source: <http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2004-04/1239.html>

From: glen herrmannsfeldt (*gah_at_ugcs.caltech.edu*)

Date: 04/27/04

Date: Tue, 27 Apr 2004 21:18:59 GMT

Andy Nelson wrote:

> *These questions also have quite a bit to do with OS architecture*
> *(which I know much less about) rather than Fortran, but am*
> *asking here since the code and issues are mostly Fortran*
> *related. Partly, I'm asking because I'm not sure the advice my*
> *colleague is getting is accurate from his systems folk who may not*
> *be specifically Fortran folk. Partly for my own sake of*
> *understanding how Fortran memory implementation stuff really*
> *works, both in general and what might be the actual implementation*
> *on various real machines.*

Well, they have a little bit to do with the OS, but mostly Fortran. One reason they are OS related, as discussed here not long ago, has to do with the way C programs do memory allocation, and the design of OS's to follow that. One is that C programs tend to do more dynamic allocation, and relatively little stack allocation.

(snip)

> *All arrays are*
> *dimensioned at the beginning and stay that way throughout the*
> *execution of the code. Nearly all are originally declared in some*
> *sort of common block but are often passed to a subroutine as an*
> *argument, rather than through their common block.*

COMMON are very likely statically allocated, so that should be fine.

> *The specific problem I'm wondering about is with how variables*
> *and arrays are transferred from a caller to a callee in*
> *the argument of a subroutine, and how they might be allocated*
> *memory space (heap/stack) if a copy is made.*

(snip)

> *–Under what circumstances would one expect that the call*
> *might trigger a copyin/copyout sort of*
> *arrangement and what might trigger a call by reference sort*

> *of arrangement?*

Fortran, at least in the F66 and F77 days, generally worked with either pass by value return (local copy) or pass by reference. Compilers I used to know did scalars as pass by value return and arrays by reference. It was slightly more efficient that way. I would be very surprised to see pass by value return for arrays on an F77 compiler.

(snip)

> *-Under what circumstances would one expect that the copy (if one is made) goes to which sort of memory...stack or heap?*
> *As I understand it, stack memory is far more efficient (fast) than heap...how might I (as a code writer) avoid such issues in favor of just using some original version or if some copy is needed, to make it a copy-to-stack flavor rather than copy-to-heap flavor.*

Speed is certainly system dependent, more hardware than OS, but I don't believe the differences are large. Well, stack might be slightly more efficient as long as it stays small.

In the 16 bit DOS days, when the stack could only be 64K, it was slightly easier to address than heap.

(snip)

> *-There are also occasions when a routine might be called with one or the other npart=0 (see code below). As a completely separate question, my copy of M&R says zero length arrays are allowed, but I've run across compilers (can't remember which) where the code crashed on trying to enter a subroutine with such a situation, and worked when I changed the array declaration to a fixed parameter value.*

I believe you would be safer to make it 1. As Richard said, 0 length arrays weren't legal in F77, so you can't be passing one that is actually zero. The rightmost subscript of dummy arrays is normally only used for bounds checking, if any is done. It is not needed to do the subscript calculation.

> *Now below there is a bit of hearsay involved below, since I don't have direct access to the systems folk who made the original claims. Bear with me. The situation is that the code will not run due to insufficient stack size, given a problem of some size that is still much smaller than the total available memory.*

I do remember when I first started using 32 bit versions of OS/2 that the linker still allocated a default 2K stack. The stack was in virtual memory, and not even allocated until used, so there

comp.lang.fortran: Re: Fortran memory allocation (stack/heap) issues

was almost no cost to a very large stack. It might just be that someone has set a small stack size.

- > *The machines where problems are found at the moment are a*
- > *Hitachi SR8000 and an Itanium2 machine put together by*
- > *some small more or less no-name company (I never heard of them) and*
- > *using the Intel compiler, but I'm at least as interested in the*
- > *answers in general as compared specifically to one machine, since we*
- > *would like to be able to run the code easily/portably/efficiently*
- > *on lots of different machines and provide it to others as well.*

Interesting machines, and neither very popular. It is possible that you found a simple bug that just hasn't been noticed yet.

(snip)

- > *program myprog*
- > *parameter(npart1_max=<somenumber>)*
- > *parameter(npart2_max=<someothernumber>)*
- >
- > *integer iarray(npart1_max) !various of these have*
- > *double precision array(npart2_max) !different npart1 or npart2*

These may be allocated on the stack in myprog. They are local variables in this case. As Richard says, if you put them in a SAVE statement they are more likely to be statically allocated, or put them in COMMON with the COMMON block name in a SAVE statement.

(snip)

- > *subroutine someroutine(npart1, npart2, array, iarray)*
- > *parameter(npart1_max=<somenumber>)*
- > *parameter(npart2_max=<someothernumber>)*
- >
- > *integer iarray(npart1_max)*
- > *double precision(npart2_max)*

Changing the dimension from npart1 to npart1_max is unlikely to change where they are allocated.

If you want them stack allocated you can make your own local arrays, dimensioned with a constant or PARAMETER, and copy them yourself.

I would look carefully for local (not dummy arguments to the subroutine) arrays that might be causing stack overflow.

- > *Now the problem/question I have with this is that all of these*
- > *arrays are declared with a fixed length in whatever main routine*
- > *they come from. There are no cases of allocate/free arrays in*
- > *any routines in the whole code. So I'm at least partly confused*

comp.lang.fortran: Re: Fortran memory allocation (stack/heap) issues

> *as to why there might be issues of heap vs stack allocation to*
> *begin with. That lead to this post and my questions above.*

Both heap and stack are foreign concepts to F66 and F77, and you will even find discussion of them gets shot down in comp.lang.c.

It might just be that you need to make the stack bigger, or it might be that you should use SAVE to make sure they are statically allocated.

— glen