Common Causes of Segmentation Faults (Segfaults)

A segmentation fault (often called a *segfault*) can occur if a program you are running attempts to access an invalid memory location. When a segmentation fault occurs, the program will terminate abnormally with an error similar to the following message:

```
SIGSEGV: Segmentation fault - invalid memory reference.
forrtl: severe (174): SIGSEGV, segmentation fault occurred
```

The program may generate a core file, which can help with debugging.

If you use an Intel compiler, and you include the **`-g -traceback`** options, the runtime system will usually point out the function and line number in your code where a segmentation fault occurred. However, the location of the segmentation fault might not be the root problemâ a segfault is often a symptom, rather than the cause of a problem.

Common Segfault Scenarios

Common scenarios that can lead to segmentation faults include running out of stack space and issues resulting from bugs in your code.

Running Out of Stack Space

Stack space is a segment of program memory that is typically used by temporary variables in the program's subroutines and functions. Attempting to access a variable that resides beyond the stack space boundary will cause segmentation faults.

The usual remedy is to increase the stack size and re-run your program. For example, to set the stack size to unlimited, run:

For csh

```
unlimit stacksize
```

For bash

```
ulimit -s unlimited
```

On the Pleiades front-end nodes (PFEs), the default stack size is set to 300,000 kilobytes (KB). On the compute nodes, PBS sets the stack size to **`unlimited`**. However, if you use **`ssh`** to connect from one compute node to another (or several others) in order to run programs, then the stack size on the other node(s) is set to 300,000 KB.

Note: Setting the stack size to **`unlimited`** on the PFEs might cause problems with Tecplot. For more information, see Tecplot.

Bugs in Your Fortran Code

In Fortran programs, the most common bugs that cause segmentation faults are array bounds violationsâ attempts to write past the declared bounds of an array. Occasionally, uninitialized data can also cause segmentation faults.

Array Bounds Violations

To find array bounds violations, re-run your code with the Intel ifort compiler using the **-check** (or **-check all**) option in combination with your other compiler options. When you use the **-check** option, the Fortran runtime library will flag occurrences of array bounds violations (and some other programming errors).

When the runtime library encounters the first array bounds violation, it will halt the program and provide an error message indicating where the problem occurred. You may need to re-run the code multiple times if there is more than one array bounds violation.

Note: Code compiled with the **-check** option may run significantly slower than code compiled with normal optimization (without the **-check** option).

Uninitialized Variables

You can use the **-init=keyword** option (available in the 2015 Intel Fortran compiler and later versions) to check uninitialized variables. The following keywords can be used with the **-init** option:

[no]arrays
: Determines whether the compiler initializes variables that are arrays or scalars. Specifying **arrays** initializes variables that are arrays or scalars. Specifying **noarrays** initializes only variables that are scalars. You must also specify either **init snan** or **init zero** when you specify **init [no]arrays**.

[no]snan
: Determines whether the compiler initializes to signaling NaN all uninitialized variables of intrinsic type REAL or COMPLEX that are saved, local, automatic, or allocated.

[no]zero
: Determines whether the compiler initializes to zero all uninitialized variables of intrinsic type REAL, COMPLEX, INTEGER, or LOGICAL that are saved, local, automatic, or allocated.

Note: The **-init** compiler option does not catch all possible uninitialized variables. To find more, you can use the NAS-developed **uninit** tool. For information about using this tool, see the NAS training presentation uninit: Locating Use of Uninitialized Data in Floating Point Computation in Big Applications.

For more information about segmentation faults, see:

- Determining Root Cause of Segmentation Faults SIGSEGV or SIGBUS errors (Intel Developer Zone)
- Segmentation Fault (Wikipedia)

Article ID: 524
Last updated: 14 Dec, 2015
Revision: 40
Porting/Building Code -> Optimizing/Troubleshooting -> Debugging -> Common Causes of Segmentation Faults (Segfaults)
https://www.nas.nasa.gov/hecc/support/kb/entry/524/