Option - Session 7: Incorrect Results - F90 Example

The next example is for Fortran users – it has two bugs, both are left as an exercise for you.

The code we will use is the Array example.

```
cd Array
make
```

Exercise

The code is a simple convolution code. It is an MPI code, although it doesn't do any communication – and one or two processes is enough to show the problem.

A matrix B has a 3x3 so-called convolution matrix M applied to each cell. By this we mean that the new value in matrix C at cell (i,j) is the arithmetic sum of the products of each cell surrounding B(i,j), and B(i,j) itself – with a multiplication mask given by M.

Thus the 3x3 convolution matrix with all values zero except M(2,2), the central element, which is 1, is an identity matrix for convolution.

The developer of this code was kind enough to create and include a test case involving the identity convolution matrix, which applies it to an 5x5 array B – but something strange happens, the output is not the same as the input.

```
mpirun -np 8 ./array
```

The output snippets below should match values.

```
 A real convoluted example code
 Start of input b
   1.0000000       0.0000000       0.0000000       0.0000000
0.0000000
…
```

```
 Output of the convolution of b
   0.0000000       0.0000000       0.0000000       0.0000000
0.0000000
…
```

However, they are different. Can you find out why, and fix the problem?

Hint

- Use watchpoints to help you determine when the new values are written. There are some cells which you know are wrong, watch one of those.
- Once you have found where things happen, double click on the line at the start of the relevant loop to set a breakpoint there (line 161 looks a good one for this).
- Restart the example and step through the offending loop to see what happens.

You will probably kick yourself when you see the answer, maybe a different font would help – try using the Session/Options/Appearance menu and change the code viewer font settings.

- Be really careful when you fix the problem so that you change the code in both loops – when you assign the elements of C to B, remember which elements you did not compute!

Exercise

The code may now run successfully, but there is still a bug.

This bug might, or might not, appear for you today – it's a memory bug that is triggered at random. The chance of it appearing is related to a number of things – a random number seed, and the machine page size (usually 4k for x86_64 Linux), and the current layout of the heap, the last of which is dependent on the compiler.

One run of this code might, on 1,000 cores, cause a segmentation fault on 10 random cores, which is exactly when you need a real debugger.

On our Open MPI x86_64 system, with the gfortran compiler, one run of 8 processes produced a single segmentation fault. On another run, none of the processes errored.

Disable memory debugging in Allinea DDT, and run through to the end on a few cores inside Allinea DDT.

Try this two or three times to see if you always get the same processes erroring.

A bug like this would be easier to fix if it happened every time? Allinea DDT's memory debugging gives exactly the help we need here.

Can you fix the problem?

Hints

- Using the underflow/overflow detection in the memory debugging settings, find where the program crashes.
- Edit the code to fix the problem, compile and re-run.
- You might notice that there are actually two similar bugs – re-run the code with the “below” protection instead of “above” (or vice-versa, if you started with below initially) and see if the problem shows again.

If you started with “above” guard pages, you might have been shown two different error messages at first – this is because some of the processes crashed before they got to the end of the loop – when they wrote below the array – which obviously happens during earlier iterations in the loop. Other processes were “lucky” and only crashed when Allinea DDT forced the problem to show with guard pages, at the last iterations of the loop. This just goes to show how random memory bugs can be!