INTERACTIVE DEBUGGING

T. Daniel Crawford, Virginia Tech

Why Interactive Debuggers?

- No need to recompile the code multiple times!
- In the absence of compiler optimization (which complicates the code in unpredictable ways), one can often rapidly identify subtle bugs more rapidly.
- Simple to use, and much more convenient than inserting "printf()" statements.
- One can actually change the values of variables during execution to either force or avoid exceptions.

How to Prepare Your Code?

- Re-build all components you want to investigate with a “-g” flag passed to the compiler.
- Ideally, one should avoid all optimization flags (either “-O0” or no “-On” flag at all).
- NB: Unoptimized code may (will!) run much, much more slowly than with compiler optimizations on. This can be problematic for bugs that occur only for larger computations.

Basic GDB Commands

run	Execute the program
list	List source code lines near the current execution position.
break <loc>	Stop the program at a specified location, <loc>.
print <var>	Print the value of the specified variable, <var>.
cont	Continue program execution from the current position.
next	Step forward one line, but remain at the current stack level.
step	Step forward one line, even into a function.

Basic GDB Commands

where	Examine the stack frame.
up/down	Shift up/down one level in the stack frame.
whatis <var>	Identify the type of <var>.
watch <var>	Stop program execution if then value of <var> changes.
set ...	Set the value of, e.g., a variable.
info <arg>	Give information on current watchpoints, breakpoints, etc.

Debugging Large Calculations

What if your code fails only for a very large computation? What if it takes hours, days, or even weeks to reach a bug in a large program? How can you access your debugging session from any computer?

The “screen” command available on most (all?) UNIX systems and allows you to start a shell session to/from which you can attach/detach at any time without destroying the session.