The Distributed Debugging Tool

Quickstart Guide
for Fortran programs
## Contents

Contents..................................................................................................................................................2  
1. Introduction.......................................................................................................................................3  
2. Starting A Session..............................................................................................................................4  
   Checking Your Program Status...........................................................................................................5  
3. Controlling Your Processes...............................................................................................................7  
   Advancing Processes Forward By A Line............................................................................................7  
   Stepping Into A Function or Subroutine............................................................................................7  
   Stepping Out Of A Function or Subroutine.........................................................................................7  
   Following If Branches..........................................................................................................................8  
   Pausing Processes At A Breakpoint......................................................................................................8  
   Manually Pausing Processes................................................................................................................8  
4. Viewing Data........................................................................................................................................9  
   Current Line(s)...................................................................................................................................9  
   Local Data..........................................................................................................................................9  
   Keeping An Expression In View..........................................................................................................9  
   Finding And Fixing An Error..............................................................................................................10  
5. Conclusion..........................................................................................................................................12
1. Introduction

In this document you will use the example program provided with DDT to gain an overview of DDT and its features. You will see how DDT can be used to debug a fortran program using the basic features found within DDT. After following this guide you should be able to start a program, control your processes, examine data, set breakpoints and learn how to find and recover from errors in your programs.
2. Starting A Session

By now you should have installed DDT successfully. In your DDT directory you will find the examples subdirectory. We will now compile this:

F77:

```
[examples]$ make -f fmake.makefile
mpif77 -g -O0 -o hellof77.exe hello.f
[examples]$
```

F90:

```
[examples]$ make -f f90make.makefile
mpif90 -g -O0 -o hellof90.exe hello.f90
[examples]$
```

To verify that you have MPI working correctly, run the example without DDT first.

F77:

```
[examples]$ mpirun -np 4 ./hellof77
```

F90:

```
[examples]$ mpirun -np 4 ./hellof90
```

On termination, the hello program will, if your MPI supports I/O (input/output) from remote processes, have produced output that ends:

```
[....]
All done... 3
All done... 0
[examples]$
```

We now start DDT debugging this program:

F77:

```
[examples]$ ../bin/ddt hellof77
```

F90:

```
[examples]$ ../bin/ddt hellof90
```

Note: if DDT is in your path, you can just type `ddt ./hellof77` or `ddt ./hellof90`. If this is the first time that you have used DDT you will be greeted with the configuration wizard. This will walk you through some of the steps necessary for setting up DDT to recognize your MPI implementation and the remote access to nodes.
Once configuration is complete, DDT will present a dialog box; you should now ensure that the right number of processes are selected. The advanced menu can be used for supplying extra parameters to your executable of your MPI, as well as to enable memory debugging.

![Session Control Window](image)

**Fig.1 The session control window for an F77 program**

You should now see the DDT session control window as in Fig.1 (for an F90 program the Application will read “hellof90” not “hellof77”). Select “Run” and DDT will connect to your processes, load your source files and take you into the main DDT window, ready to begin your debugging.

### Checking Your Program Status

After loading the example program into DDT as described above your main DDT window should look like the one seen in Fig.2. The source files that have been found are shown, and the current file, that from which MPI_Init is called, will be loaded and shown. You can examine any of your source files at any time by selecting it from the list. Note that for some compilers and MPI implementations the line after MPI_Init will be highlighted instead, as it is the next one to be executed.

At the top of the screen is a collection of coloured numbered boxes, and three lines: “All”, “Root” and “Workers”. These are process groups. Process groups are used to control your processes en-masse. Each numbered box represents a process. One of the lines will be brighter than the others, and one of the boxes may be brighter and have a dotted border. These are the current group and the current process (if one is selected) respectively.

The process boxes are either red or green. A green process is running, and a red process is stopped, it could be paused or it may have terminated.

You should see a red bar on the line containing MPI_Init. This indicates that some processes in the red process group are on that line, and the red process group is the current group. If the line is bright, this also means that the currently selected process is on that line. Hover your mouse pointer over this line and DDT will tell you which processes are at that point. Select a different process group by clicking on “Workers”. The red line will change to the colour of the Workers group.

Information about which line a process is on is updated every time the program is paused.
Fig. 2 DDT main window
3. Controlling Your Processes

Process groups as described above are used to control most of the operations you will wish to perform on your processes. If you wish to make all processes stop at a point in your code, or all processes move a single step, the “All” process group would be selected. Alternatively, if none of the predefined groups are exactly right for you, by right clicking in the process group area, you can create your own using drag and drop moves.

The next step in our exercise will be to advance all the processes through the code. We will be using the process operation buttons found in the toolbar just above the process groups (see Fig.3). These buttons are (from left to right): Play, Pause, Step In, Step Over, and Step Out. The last four change the current stack frame and do not cause your processes to execute any code at all. From left to right, they move down the stack, up the stack, to the bottom of the stack and set all processes in the group to the current stack frame. Navigating the stack is described in more detail in the main userguide.

![Process operation buttons](Fig.3 Process operation buttons)

**Advancing Processes Forward By A Line**

Select the “All” process group using the mouse, and then press the `Step Over` button. The red line in your code will advance one line, your processes may also briefly turn green whilst they perform the step, returning to red when they have finished working.

In the example, press step over again - until the processes are at the line: “CALL SUB1()”.

**Stepping Into A Function or Subroutine**

Suppose we now wish to see what happens inside FUNC1(). Press the `Step Into` button, and the code window leaps to the relevant part of the source code. The processes have now moved into SUB1().

Do this again to enter FUNC1().

**Stepping Out Of A Function or Subroutine**

You should now be seeing the definition of FUNC1() and it looks uninteresting so lets go back up to SUB1() and see what happens there. Click the `Step Out` button. The processes will now continue until they reach the end of FUNC1() and then return to SUB1().

After this the processes are still shown as being on the line of SUB1() that calls FUNC1(); this is because it hasn’t finished executing the last part of this line, which is where the return value of FUNC1() is assigned to test.
**Following If Branches**

Probably the most important part of SUB1() is the if statement and we’d like to watch what happens in there; which path is taken. By stepping over we can watch what happens as the program proceeds. Use the “Step Over” button to do this until you leave SUB1().

**Pausing Processes At A Breakpoint**

You could keep using “Step Over” until you have reached a point in your code that is important to you, but this would be time consuming. It’s also possible that some processes may take more instructions to reach a point of interest than others, leading to different processes being on different lines of code. A better idea is to cause your processes to stop at a breakpoint.

We’d like to see what is happening during an MPI operation - where process 0 is receiving data from all of the other processes. Select the process group “Workers” and then click on the line containing MPI_Send; line 56. Click the right mouse button and choose “Add breakpoint”, this will add a breakpoint for the currently selected group.

Now select the “All” process group, and press play to resume execution. After a short period of computation, you will see that process 0 is Green, still running, and processes 1, 2 and 3 have reached the breakpoint.

You can have as many breakpoints as you like, each breakpoint you set will be active for the currently selected process group. You can see all the breakpoints that are set by clicking on the breakpoints tab at the bottom of the window.

**Manually Pausing Processes**

Process 0 is still running, but we can guess that it will be waiting to receive data and is not actually “working”. By selecting either the “Root” or the “All” group if neither is selected, and then pressing “Pause”, process 0 can now be stopped.

Select process 0 and you will see the stack, located to the right side of the code. This can be viewed by clicking the mouse over the stack display window, which will currently be displaying the top of the stack.

If you double click on process 0 the code window will jump to the point in the code where process 0 has paused.

All of the process operation buttons, and the setting of breakpoints work on the currently selected process group. The ability to configure and save your own groups is a very powerful feature making it much easier to debug your programs using DDT.
4. Viewing Data

Now that you know how to control processes in DDT, we are now ready to look at the data or contents of the processes.

Current Line(s)

By clicking on any of the paused processes, we can examine the data on the current line of code. If we choose any of the processes that are about to execute a send procedure we can view the message that will be sent to process 0. First select process 1 (using a double click) and then click on the “Current Line(s)” tab, to the right of the window.

The current line(s) panel shows the variables that used on the current line of code for the currently selected process. In this case we can see the “message” variable, and the dest and tag variables. We can see that the dest and tag variables are integers with a value of 0. We can also see from looking at the “message” variable that it is a scalar character variable.

You can see more than the currently selected line by dragging the mouse around the lines of interest in the code window.

![Current line for process 1](Fig.4_Current_line_for_process_1)

Local Data

The variables known as “locals”, those in scope in your current routine, are displayed in this tab. Some of the information here is dependent on your compiler information, so the precise list of variables displayed may differ from machine to machine.

Keeping An Expression In View

It would be useful to keep a particular eye on the values in “ierr”, in order to know if our mpi calls are throwing errors or not; we’d like to watch this as we proceed. We can do this by putting expressions into the evaluate window. This is found below the current line(s)/locals window.
You can drag “ierr” from the locals window into the evaluate window using mouse drop and drop techniques, or you can type it in directly by right clicking in the evaluate window and selecting “Add expression”.

We’ll now make processes 1, 2 and 3 send the data. Press `Step Over` whilst the “Workers” group is selected. Now examine the value of ierr for each of the 3 processes and it should still be 0. On some MPIs the buffering may not be sufficient to let all three communications occur without first setting process 0 running, you will be able to use the process operation buttons talked about in the previous section to find this out if your processes do not stop and instead are running but waiting on I/O.

**Finding And Fixing An Error**

The “hello” program was designed to break. If the program is started with exactly 8 processes it will abort with an error. Let’s try to find and fix this error.

We can restart “hello” by selecting `Session` and then the `Session Control` option from the tool bar. Then click on `End Session`.

In the session control dialog, change the number of processes to 8. Now commence the program as before. When DDT is ready, press play to start everything going.

![Fig.5 Error output in DDT](image)

Process 5 will immediately stop, and may print output to stderr (depending on your MPI implementation and other system settings). Double-click on process 5 and look at its stack trace – you can clearly see that a call to MPI_Send resulted in MPI_Error being called.

Click on the stack and choose the frame containing the `main` function – for gnu-compiled codes this may actually be called main__. You’ll know you have the right one when the current line variables appear below. Looking at these it is now obvious to see that you are trying to call an MPI_Send without first initializing the variable “dest”.

In this way it is possible to find errors and faults in your code and locate where they are and what the problem is.
Hint: If using MPICH standard p4 mode, you will often have noticed a p4_error output in the `stdout` window, this is often a sign that some memory allocation problem has occurred and is a good first guide to the fact that you have a problem in your program. See Fig.6 for an example of this.

<table>
<thead>
<tr>
<th>Stdout</th>
<th>Stderr</th>
<th>Breakpoints</th>
<th>Watches</th>
<th>Stdin (to current group)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Currently Displaying:</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 5: p5_12090: p4_error: 8262</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 0: My rank is 0!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 0: waiting for message from 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 0: Message received: HelloFromMexxxxxxxxxx!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 0: waiting for message from 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 4: My rank is 4!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 4: Greetings from process 4!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 4: Sending message from 4!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig.6 Errors in the Stdout window
5. Conclusion

You have now mastered the basic operations of DDT for debugging a fortran program. For a more in depth look at the possibilities with DDT you can read supplied userguide. Alternatively take a look at the quick start guide and tutorials for C, to learn how to use DDT's powerful features on standard C code.