DDT and Totalview on HPCx

Mike Ashworth and Andrew Sunderland
STFC Daresbury Laboratory, Warrington, WA4 4AD, UK
Email: m.ashworth@dl.ac.uk, a.g.sunderland@dl.ac.uk

Abstract

Debugging applications is an essential part of the cycle of the development and improvement of codes in computational science. Although a range of debugging techniques are available to the scientific programmer, a fully-featured, interactive, graphical, parallel debugger has become an essential tool and an essential part of the service provision on high-performance systems.

HPCx offers two interactive, parallel debuggers: Totalview from Totalview Technologies and Allinea Software’s DDT. This report gives an overview of the two tools and how they are used on HPCx, and compares the capabilities that each one offers.

Version 2.0 of the report is a full revision of HPCxTR0506 reflecting new capabilities available in the latest versions of both products.

Version 2.0
15th April 2008

This is a Technical Report from the HPCx Consortium.


© UoE HPCx Ltd 2008

Neither UoE HPCx Ltd nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.
1 Introduction

Debugging code is an essential part of the cycle of the development and improvement of scientific codes. Many techniques are applicable. In some circumstances it may still be appropriate to simply insert ‘write’ statements into the code. In order to deal with other problems we find that a modern graphical, parallel debugger is an essential tool.

On HPCx we currently offer two graphical, parallel debuggers: Totalview and DDT. This report introduces each debugger and highlights their individual strengths and weaknesses.

Both tools demonstrated in this report require the user to follow the usual procedure for debugging a program:

- Compile your code with –g in order to make symbol tables available to the debugger. We also recommend the –qfullpath option – this generates absolute path names for source files, which is especially useful if the source files are not in the same directory as the executable.
- Start your program using the runtv or ddt commands as described below.
- Set breakpoints.
- Examine data.
- Identify bug (hopefully).
- Close debugging session and edit source code accordingly.

2 Totalview Overview

Totalview is an interactive graphical debugger, developed and marketed by Totalview Technologies (http://www.totalviewtech.com) that enables the debugging, analysis and tuning of serial, multi-threaded and multi-process programs. On HPCx Totalview can be run either post-mortem or interactively for serial or parallel jobs. Access to the debugging session takes place through the Totalview GUI, which allows users to access various functions including setting breakpoints, switching between processors and diving on variables to display their current value. Full details of the use of Totalview are available in the User Guide [1].

TotalView also features an integrated, interactive, view-based memory debugger that helps you find and eliminate memory problems at the same time as you debug your code. Is built into the debugger and re-compilation is not necessary (providing the –g flag is used).

2.1 Running an Interactive Debugging Session on HPCx

On HPCx a script named runtv is provided for users to run interactive debugging sessions on the interactive nodes. Ensure that your program (e.g. myprog.exe) is compiled with the –g flag, and use a standard Loadleveler script (e.g. inter.ll) for interactive usage on HPCx (see the HPCx User Guide for this) and then issue the command:

```
runtv inter.ll myprog.exe
```
You will then be prompted for your password (unless you have ssh forwarding in place for your HPCx login session). A blank Totalview window will then appear, as shown below.

![Blank Totalview window](image)

Left click the Go button on the taskbar to start your parallel program running. A pop-up window then confirms that poe is running and asks ‘Do you want to stop the job now?’.

![Pop-up window](image)
Clicking “No” will cause the program to start running immediately, which is useful if the code fails and you want to run straight to the point of failure. Click the “Yes” button in order to start interactive debugging of your application, allowing you to browse the source code and set breakpoints before starting execution.

The GUI for a typical Totalview debugging session of a parallel program on HPCx is shown below:

---

Setting Source Code Paths and Startup Parameters in Totalview

Totalview will automatically search the current directory and your environment’s PATH variable for source code, executables and object files. If these reside in different directories, set search paths to these directories with the File => Search Path command.

Before the parallel program runs, users may want to set the program’s command-line arguments or environment variables or change the way standard input, output, and error. All this behaviour can be customized from the Process > Startup Parameters window.
DDT and Totalview on HPCx

3 DDT Overview

The Distributed Debugging Tool (DDT) is a graphical debugger specifically designed for debugging parallel codes written using MPI and/or OpenMP. DDT is a product of Allinea Software Ltd. (http://www.allinea.com). Like Totalview, DDT can be used for debugging serial, multi-threaded (OpenMP) or multi-process (MPI) programs. Full details of the use of DDT are available in the User Guide [2].

3.1 Using DDT on HPCx

Setting the environment for DDT

- Set an environment variable DDTPATH with the value /usr/local/packages/ddt
- Set an environment variable DDTCONFIG with the value /usr/local/packages/ddt/config.ddt
- Set an environment variable ACCOUNT_TAG with the name of your HPCx account e.g. z001
- Add /usr/local/packages/ddt/bin to your path

In csh this is achieved by adding the following to your .login script:

```csh
setenv DDTPATH /usr/local/packages/ddt
setenv DDTCONFIG $DDTPATH/config.ddt
setenv ACCOUNT_TAG z001
set path = ( $path $DDTPATH/bin )
```

In ksh this is achieved by adding the following to your .profile script:

```bash
export DDTPATH=/usr/local/packages/ddt
export DDTCONFIG=$DDTPATH/config.ddt
export ACCOUNT_TAG=z001
set path = ( $path $DDTPATH/bin )
```

Documentation

This technical report provides basic details of how to use DDT. For full details see the documentation in /usr/local/packages/ddt/doc/, which contains the following files:

- userguide.pdf - Userguide
- quickstart-c.pdf - Quickstart Guide (C programs)
- quickstart-f.pdf - Quickstart Guide (Fortran programs)

Interactive jobs

DDT is available to debug programs run under the interactive queue on HPCx. This is subject to the normal restrictions for this queue, currently a maximum of 32
processors and a runtime of 1 hour. To run DDT interactively with a serial or parallel program simply start DDT using the command alone or together with the name of your executable, thus:

user@l1f401> ddt

or

user@l1f401> ddt a.out

You do not need to supply a LoadLeveler script – DDT generates its own script, in fact this is why the ACCOUNT_TAG environment variable is required.

If everything is correctly set-up for you to display X-Windows on your local screen you should now see the main DDT session window and a welcome window. If not then follow the instructions in the TotalView FAQ entry (http://www.hpcx.ac.uk/support/FAQ/totalview/).

The welcome window, shown below, allows you to choose whether to launch the program for debugging or to carry out post-mortem debugging of an existing core file.

![Welcome to DDT](image)

Let’s click on “Run and Debug a Program”. This opens the run window, which allows you to type in or browse for the executable name, if you have not specified one on the command line. There is also a field for command line arguments to be passed to the executable and places to specify the number of processors (default 8) and/or OpenMP threads. An “Advanced” button reveals further options, including a field for the input file to be attached to stdin. A button which says “Pause when program reaches exit or _exit” can be very useful if the code is stopping without aborting, either at a user stop statement deep within the code or in the run-time libraries. This is also where you turn on memory debugging (see section 4.7).

When you are ready to run the program, click on the "Submit" button. A window will briefly appear showing the interactive job queue and you will see a progress bar showing that DDT is attaching to each of the processes. If the job can not run in the interactive queue it will wait, the "Job submitted" window will stay on the screen and you will have the opportunity to cancel the job and try again later if you wish.
DDT will wait at the MPI_Init call for you to press the "Play" button to start the execution of the code. If you have compiled with the -g and -qfullpath options (recommended) and the source files are available, you will see the source code. At this stage you can fix the MPI ranks if necessary. Step or run through the code until you are beyond your call to MPI_Comm_Rank. Right click on the variable containing the MPI rank and follow the menu options to “View Across Processes (CPC)”. Then click on “Use as MPI Rank” and you will see the process numbers change.

The GUI for a typical DDT debugging session of a parallel program on HPCx is shown below:

4 Comparison of Totalview and DDT

Both Totalview and DDT are sophisticated tools offering a graphical interface to a range of debugging capabilities. Both can operate with serial codes and with parallel codes built with OpenMP, MPI or mixed OpenMP/MPI. In this section we compare the operation of some of the major functions between the two tools.

4.1 Process Groups

In DDT each process is represented by a square containing its MPI rank (from zero) and is colour-coded according to its status: green for running, red for stopped. Selected processes are recognized by having a dashed border and a lighter shade. If you double-click on the square icon for a particular process, DDT jumps to that process and updates the code viewer, stack trace and variables accordingly.
DDT allows you to group processes so that actions can be applied to a subset of the total number of processes. This is especially useful when running on large numbers of processes. By default, when you start a debugging session DDT provides three process groups: All and Workers contain all processes; Root is empty but is intended to just contain the process with MPI rank 0 usually used as a master or root process in parallel programming. If we set the process numbers to be the MPI rank (as described earlier) we can drag the rank zero process into the Root group and delete it from Workers.

You can create your own groups. Right-click on the groups’ area and select “Add group” to create a user defined group. You can then drag processes from one group to another. The top part of the GUI is shown below where we have created two new groups for even and odd processes.

<table>
<thead>
<tr>
<th>Group</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>Root</td>
<td>0</td>
</tr>
<tr>
<td>Workers</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>Even</td>
<td>0 2 4 8 10 12 14</td>
</tr>
<tr>
<td>Odd</td>
<td>1 3 5 7 9 11 13 15</td>
</tr>
</tbody>
</table>

When looking at a single process it can be useful to create a new group with just that process in it.

Totalview does not have this feature.

### 4.2 Controlling program execution

DDT has video-type buttons to control program execution: a Play button to start or continue program execution and a Pause button to stop it. These buttons start and stop all processes in the currently selected group. There are three different ways to step through the program: Step into moves to the next source line, but if it is a subroutine or function enters the function and stops at its first line; Step through simply moves to the next source line; and Step out runs through the rest of the current function and stops at the next line in the calling function.

After starting a Totalview session for a parallel program on HPCx, a Root window summarising the started processes is created. An example of this window is shown below. Most importantly, this window tracks the Status of processes in your parallel job. The most common Status identifiers are B (at a breakpoint), T (stopped), E (error) and R (running). The figure below demonstrates the Process Window at the start of an interactive debugging session on HPCx for a program with 4 MPI processes. Process 1 is always reserved for poe, which remains under the debuggers control. Users should not attempt to start, stop or otherwise interact with this process.
You can switch between processes in the main debugging window either by left-clicking on listed tasks in the Process Window, or by using the Process Cycle (+/-) buttons on the main window shown below.

Processing can then be controlled interactively via the Totalview taskbar buttons shown below. These allow you to step through the program line-by-line, entering subroutines and functions and stopping and starting processing as desired.

### 4.3 Action points

DDT has breakpoints and watchpoints. Set a breakpoint in DDT by right-clicking on the source line and selecting **Add breakpoint for All**. Breakpoints may be limited to certain groups of processes by selecting a process group, say **Root**, and then the menu item when you right-click on the source line changes to **Add breakpoint for Root**. The **Breakpoints** pane in the bottom panel of the GUI shows the list of current
breakpoints. As well as being able to delete breakpoints, you can save and reload a list of breakpoints. Conditional breakpoints can be set by entering a conditional expression in the **Condition** column. Execution then continues past the breakpoint unless the condition is true.

A watchpoint is like a breakpoint but applies to a variable rather than a source line. Execution stops when the value of the variable changes. The User Guide warns that this may not work for variables on the stack and combined with the problems ‘seeing’ other variables (section 4.4) we have not been able to test this feature successfully.

**Totalview** supports four types of action point which are described in greater detail in the User Guide:

- A breakpoint stops execution of processes and threads that reach it.
- A barrier point synchronizes a set of threads or processes at a location.
- An evaluation point causes a code fragment to execute when it is reached.
- A watchpoint lets users monitor a location in memory and stop execution when it changes.

**Breakpoints**

Breakpoints can be set and unset at a specific line in the code by left-clicking on the line number. A red box containing the word STOP then appears over the line number. To enable most other types of action points, right-click the line number and select from the supplied menu. To clear the action point left-click on the line number once again.

When a process reaches a breakpoint, Totalview does the following:

- Suspends the process
- Displays a yellow arrow icon over the STOP sign to indicate progress through the program.
- Displays “At Breakpoint” in the Process Window title bar and other windows
- Updates the Stack Trace and Stack Frame Panes and all Variable Windows.

Breakpoints stop code execution on each processor just before the breakpointed line executes.

**Barrier points**
In contrast to Breakpoints, Barrier points are set to synchronize processes or threads rather than examine the state of the program. When any process hits the barrier, it stops executing (as in a breakpoint). However, unlike breakpoints, Totalview holds each thread at the barrier until all affected processes arrive at the barrier. Until this synchronized state is achieved, TV withholds responding to resume commands such as “Go”, “Next” or “Step”. In the figure below, processes 0 and 1 are held at the barrier whilst processes 2 and 4 remain unsynchronized. Therefore resume commands on these processes are currently suspended.

TotalView does not allow processes to execute past the barrier, thus synchronizing processes to the barrier's location. This means that you can allow your processes to run freely at other times because you can always gather them together when you need to execute them synchronously.

**Evaluation points**

Evaluation points are extremely useful tools when debugging large scientific codes on HPCx. Often, faults within programs occur within loops or arrays with large dimensions. In these cases to step through the loop manually until just before the iteration of interest would be tedious and time consuming.

To set an evaluation point in Totalview firstly set a breakpoint, then update this by right-clicking and selecting properties. From this Action Point Properties box, the action point can be converted to an Evaluation Point. By entering C, C++, Fortran or assembler code into the Expression window the user can control the behaviour of the evaluation point.
Watchpoints

To create watchpoints use the Tools > Watchpoint dialogue box (see below) after diving on a variable or an array. Watchpoints will automatically stop the program when the value at the associated memory location changes. Conditional watchpoints can also be set which act like evaluation points. For example these can be useful when determining when values become negative or positive or exceed a certain threshold.
4.4 Examining variables

With DDT you can not ‘dive’ on variables in the source listing, but by selecting a line or range of lines in the source window, DDT displays all the variables and their contents in the variable window on the right-hand side of the GUI. Variables are displayed in a tree format so that arrays and structures can be expanded to show their contents. Clicking on a variable displays its type at the bottom of the window. The window below shows the source and variables windows for a test program.

In addition to the variable window, you can view the value of any expression in the evaluate window. Right-click in this window, select Add Expression and type in the expression. See example below.
DDT has a special tool for viewing multi-dimensional arrays (MDA), available under the **View** menu. Alternatively, the MDA viewer can be invoked by right-clicking on a variable from the variable or evaluate windows. The MDA viewer can also be used to view expressions. Click on the **Evaluate** button and the viewer will initially present data in a table (note that this process is now much faster than with earlier versions of DDT). It also has a **Visualise** button which uses OpenGL to display a 3D plot of a 2D array section, as shown below.

![3D plot of a 2D array section](image)

Another very useful feature in DDT is the cross-process comparison (CPC). When selecting any variable in the variable window or the evaluation window, you can right-click and choose **View Across Processes (CPC)**. This shows you the value of a variable across all processes in the current process group, showing the actual values, statistics or a visualisation (see below for the MPI rank in a 32-processor run).
In Totalview the value of all types of variables, including the contents of structures and derived data types can be shown at any time that processing is stopped (e.g. whilst at a breakpoint or barrier). In addition the Stack Frame pane of the Process Window displays all of the routine’s parameters, its local variables, and the registers for the selected stack frame.
Right-clicking on a variable in the Source Pane and selecting **Dive** in the drop-down menu generates a pop-up window with the value currently assigned to the variable displayed. If the variable is an array, then the values assigned to the whole array are listed. From this window it is then possible to display the values graphically by choosing the **Visualize** option. An example of a visualised 2D square array of order 100 is shown above.

**Changing the value of variables in Totalview**

A new feature of Totalview is that variables can now be changed from within the debugging session itself. This may be particularly useful for situations where the developer wishes to alter the execution path within the program during runtime, for instance by switching a logical variable from `.true.` to `.false.`. The value of any variable or the contents of any memory location displayed in a Variable Window, Expression List Window, or Stack Frame Pane may be changed by simply selecting the value and overwriting with a new value. In addition to typing a value, you can also type an expression which may also include logical operators. Totalview distinguishes variables that have been edited with a yellow highlight (see example below).

![Variable Window Example](image)

**4.5 Program output**

**DDT** collects and displays the **stdout** output from all processes in the standard output pane at the bottom of the GUI. By default all output is displayed and each line is labelled with the process number from which it came, but the output can be filtered to show output from one process or a group of processes. By right-clicking in the standard output pane it is possible to save the contents to a file. Another pane displays **stderr** output in the same way as **stdout**.

**Totalview** does not trap the output to **stdout** or **stderr** and it is written to the terminal session as normal.

**4.6 Message queues**

The **DDT** User Guide describes the Message Queue window which shows information about outstanding MPI messages. However this feature requires a version of the MPI library with a special debugging interface, which is not currently available on HPCx.
The latest version of **Totalview** now allows access to the message queues on HPCx. No special versions of MPI libraries are required and the message queues can be checked at any time during execution. This feature is particularly useful when debugging codes with asynchronous and/or non-blocking communication models. The example shown here is taken from a program involving processes passing non-blocking messages in a ring arrangement. The message queues are inspected here just after a series of MPI_ISENDs and MPI_IRECVs have been issued by all the processes.

**Tools => Message Queues**

![Message Queue Interface](image1)

**Tools => Message Queues Graph**

![Message Queue Graph](image2)
4.7 Debugging OpenMP Programs on HPCx

Totalview supports source level debugging of OpenMP codes on HPCx. In common with MPI codes, breakpoints and synchronising barriers can be applied in both serial and parallel parts of the user program. Each OpenMP thread can be investigated independently of others and both private and shared variables can be accessed.

Preparing a debugging session for an OpenMP program on HPCx

Firstly, ensure that the code is compiled with `-g -qsmp,noauto` options. Then start an interactive Totalview session in the usual manner, as outlined in Section 2.1 of this document. Evidently for pure OpenMP jobs the `@# cpus` keyword in the job command script is set to the value 1.

On HPCx the number of OpenMP threads for a parallel job is specified by setting the environment variable `OMP_NUM_THREADS`. As Totalview does not inherit environment variables from the user’s login shell, this setting must instead be entered in the Startup Parameters window before the parallel job is started.

An example interactive Totalview debugging session for an OpenMP is shown in the windows below. When execution is stopped in a parallel region the root window now lists 2 processes – the user process (1) and the poe process (2). The user process is then subdivided into a set of threads (2.x where x is the thread ID).
The features of the main Totalview window remain largely the same as for an MPI debugging session except that individual threads can now be analysed, rather than processes (note that the Threads are now listed in the bottom pane and the Task Cycle +/- buttons are now enabled). Other differences with MPI debugging sessions are that execution cannot be stepped into (or out of) parallel regions; rather a breakpoint must be set inside the parallel region and execution must be allowed to run to it. HPCx users should refer to the Totalview Reference Guide for further information on debugging OpenMP codes. For example this details the different approaches required for viewing shared and private variables.

4.8 Memory debugging

DDT’s memory debugging features are accessed by re-linking with the dmalloc library which intercepts memory allocation and deallocation calls and performs heap and bounds checking.

For 32-bit executables:
-L$(DDTPATH)/lib/32 -ldmalloc

For 64-bit executables:
-L$(DDTPATH)/lib/64 -ldmalloc
Then when you start the DDT debugging session, before clicking the Submit button, click on Advanced>> to extend the range of options and check the box which says Enable memory debugging. Features of memory debugging include some error detection, such as freeing or referencing an unallocated variable or exceeding array bounds, checking that pointers are valid, viewing the source lines where pointers or dynamic arrays were allocated, and getting information and statistics about memory usage which can be useful in tracing memory leaks. Some of the error reporting features may duplicate features available in the Fortran run-time library.

To see current memory usage, stop anywhere in the program and select Current Memory Usage option from the View menu. This will show a graph of current memory usage for each process (shown below), which may be switched to a table view showing source line numbers where each allocatable array has been allocated and its size.

The Overall Memory Stats option shows in graphical and table form the total number of bytes allocated and freed on each process as well as the number of calls to allocate and free memory. Clearly if the difference between the numbers of allocate and free calls is growing as the code executes, this can indicate a memory leak.

Recent versions of Totalview also feature memory debugging facilities. This can be used to monitor memory usage in both Fortran and C codes and can be helpful in identifying memory leaks in users' programs. AIX applications differ from applications...
running on other platforms as AIX does not support interposition. This means that
HPCx users must re-link their codes in order to link in the Totalview heap
replacement library.

In order to enable memory debugging with Totalview, you must re-link your code with
the following files

For 32-bit executables:

-L/usr/local/packages/totalview/toolworks/totalview.8.4.0-0/rs6000/lib
/usr/local/packages/totalview/toolworks/totalview.8.4.0-0/rs6000/lib/aix_malloctype.o

For 64-bit executables:

-L/usr/local/packages/totalview/toolworks/totalview.8.4.0-0/rs6000/lib
/usr/local/packages/totalview/toolworks/totalview.8.4.0-0/rs6000/lib/aix_malloctype64_5.o

The interactive debugging session for a parallel program is then started in the usual
manner on the interactive node of HPCx.

1. Enter your password if required. The Totalview GUI should appear on your
display.

2. Click **Tools->Memory Debugging** to open the Memory Debugging Window of
Totalview. Do NOT click the **Enable memory debugging** button at this point
(this would attempt to debug the **poe** program that launches parallel programs
rather than the parallel program itself).

3. To connect to your own application you should click the **Go** button and
answer **Yes** when asked whether you wish to stop **poe**. Your source code
should now appear in the main Totalview Window and a list of your parallel
processes should now appear within the **Process Set** area of the Memory
Debugging Window.

4. In the main window, set a breakpoint (left mouse click) or barrier (right mouse
click) on one of the line numbers near the start of your source code. Click **Go**
to start the program running. The program should stop at the breakpoint. (Due
to the asynchronous nature of processes this may require more than one click
of **Go**).

5. In the **Process Set** list of the Memory Debugging Window left click on one of
your parallel processes. Memory debugging information is now available. For
example, to generate a view of the Heap Status click on the process you wish
to inspect, **Go** to an appropriate breakpoint, then click the **Heap Status** tab
and then click **Generate View**.

An example of the memory debugging analysis generated by Totalview is shown
below for the Heap:
Probably of most interest here is the **Memory Blocks Pane** where you can see a detailed breakdown of the memory used on the heap. Here, individual memory allocations are referenced back to your source code. You can switch between processes by left-clicking on the required process in the Process Set Pane of the window above. You should note that the Memory Debugging Window does not automatically refresh when switching between processes. To update the Memory Debugging window after switching processors click **Window→Update**.

A summary of all memory used by a program at a breakpoint or barrier can be generated by choosing the **Memory Usage** tab then **Generate View**. After a few seconds, a memory usage summary window is generated:

### 5 Summary

Only the basics of debugging parallel programs with DDT and Totalview on HPCx have been described in this document. Both tools contain a whole range of useful features that are described in more detail in their respective User Guides [1][2].

### References
