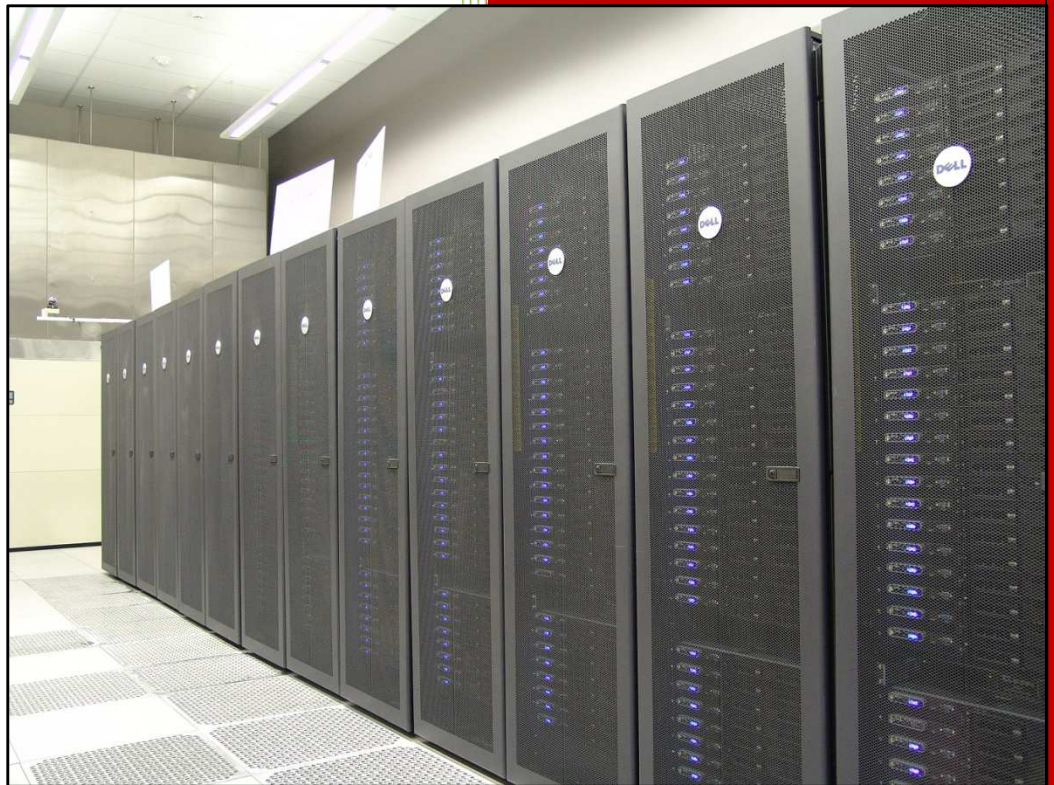


# HPCC - Hrothgar

## Getting Started User Guide – TotalView



High Performance Computing Center  
Texas Tech University

## Table of Contents

*This user guide is under development.....	3
1. Introduction .....	3
2. Setting up the environment.....	3
3. TotalView example.....	5

# User Guide

**\*This user guide is under development**

## 1. Introduction

Intro to TotalView ILOG

## 2. Setting up the environment

Hrothgar is equipped with SoftEnv to set up the environment with minimum work by users. The use of SoftEnv is not required but highly recommended by HPCC staff.

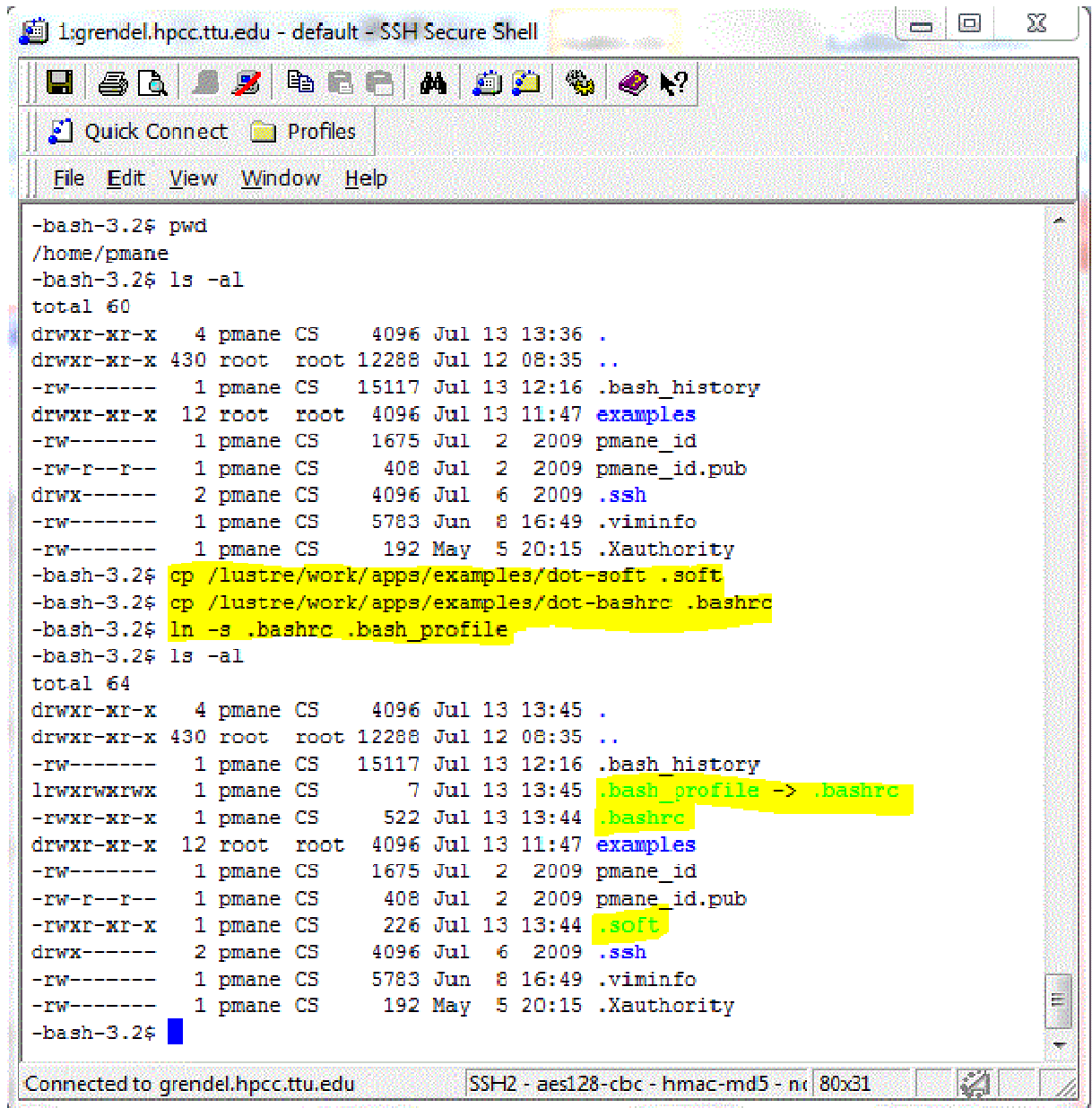
### Step 1: setting up user environment

If the user environment is already set up, please skip this step.

At the first use, the user should copy two sample dot-files: dot-bashrc is the start up script which evokes SoftEnv; dot-soft contains a list of software whose specific environment variables will be set up for the user.

```
cp /lustre/work/apps/examples/dot-bashrc .bashrc  
cp /lustre/work/apps/examples/dot-soft .soft  
ln -s .bashrc .bash_profile
```

Log out and log in again.



The screenshot shows an SSH terminal window titled "1:grendel.hpcc.ttu.edu - default - SSH Secure Shell". The terminal displays the following commands and output:

```
-bash-3.2$ pwd
/home/pmane
-bash-3.2$ ls -al
total 60
drwxr-xr-x  4 pmane CS   4096 Jul 13 13:36 .
drwxr-xr-x 430 root  root 12288 Jul 12 08:35 ..
-rw-----  1 pmane CS  15117 Jul 13 12:16 .bash_history
drwxr-xr-x 12 root  root  4096 Jul 13 11:47 examples
-rw-----  1 pmane CS   1675 Jul  2 2009 pmane_id
-rw-r--r--  1 pmane CS    408 Jul  2 2009 pmane_id.pub
drwx-----  2 pmane CS   4096 Jul  6 2009 .ssh
-rw-----  1 pmane CS   5783 Jun  8 16:49 .viminfo
-rw-----  1 pmane CS    192 May  5 20:15 .Xauthority
-bash-3.2$ cp /lustre/work/apps/examples/dot-soft .soft
-bash-3.2$ cp /lustre/work/apps/examples/dot-bashrc .bashrc
-bash-3.2$ ln -s .bashrc .bash_profile
-bash-3.2$ ls -al
total 64
drwxr-xr-x  4 pmane CS   4096 Jul 13 13:45 .
drwxr-xr-x 430 root  root 12288 Jul 12 08:35 ..
-rw-----  1 pmane CS  15117 Jul 13 12:16 .bash_history
lrwxrwxrwx  1 pmane CS      7 Jul 13 13:45 .bash_profile -> .bashrc
-rwxr-xr-x  1 pmane CS    522 Jul 13 13:44 .bashrc
drwxr-xr-x 12 root  root  4096 Jul 13 11:47 examples
-rw-----  1 pmane CS   1675 Jul  2 2009 pmane_id
-rw-r--r--  1 pmane CS    408 Jul  2 2009 pmane_id.pub
-rwxr-xr-x  1 pmane CS    226 Jul 13 13:44 .soft
drwx-----  2 pmane CS   4096 Jul  6 2009 .ssh
-rw-----  1 pmane CS   5783 Jun  8 16:49 .viminfo
-rw-----  1 pmane CS    192 May  5 20:15 .Xauthority
-bash-3.2$
```

The status bar at the bottom indicates the connection details: "Connected to grendel.hpcc.ttu.edu" and "SSH2 - aes128-cbc - hmac-md5 - n".

## Step 2: setting up TotalView environment

To take advantage of TotalView's graphical interface, one can submit interactive jobs. The following steps submit an interactive job to a node, and the user can invoke totalview's graphical interface on that node.

First, set up X-win32 for X window forwarding. Refer to the user guide Installing X-Windows server on the user guides page <http://www.hpcc.ttu.edu/php/NewUser.php>.

### Adding TotalView to the environment

*soft add +totalview*

### Override the LC limit on core file size

LC sets a minimum core file size, which is useless for debugging. It has to be overridden in order to produce a useful core file. After logging in, use the command below:

*limit coredumpsize unlimited*

## 3. TotalView example

### Copy examples

*cp -r /lustre/work/apps/examples/TotalView-Demo TotalView-Demo*

### List the contents of your TotalView subdirectory

You should have the following files:

C Files	Fortran Files	Description
<a href="#">tvEx1.c</a>	<a href="#">tvEx1.f</a>	Exercise 1 example file for demonstrating TotalView basics. Serial code.
<a href="#">tvEx2.c</a>	<a href="#">tvEx2.f</a>	Exercise 2 example file for showing additional TotalView functions and features. Serial code with a bug.
<a href="#">tvEx2.dat</a>	<a href="#">tvEx2.dat</a>	Exercise 2 input file
<a href="#">tvEx2Hang.c</a>	<a href="#">tvEx2Hang.f</a>	Exercise 2 example file demonstrating how to attach to a running code and "fix" it. Serial code with a bug.
<a href="#">tvEx3omp.c</a>	<a href="#">tvEx3omp.f</a>	Exercise 3 example file demonstrating TotalView use with parallel OpenMP codes. Shared memory threads-based parallelism.

[tvEx3mpi.c](#)[tvEx3mpi.f](#)

Exercise 3 example file demonstrating TotalView use with parallel MPI codes. Distributed memory parallelism.

### Compile the Exercise 1 example code

To produce an executable file that can be used with TotalView:

For C code: ***icc -g tvEx1.c -o ex1***

For Fortan Code: ***ifort -g tvEx1.f -o ex1***

**\*Warning Note: Do not debug on the head node**

### Submit an interactive job:

***qsub -q normal -pe fill 12 -P hrothgar***

### Start the TotalView debugger with your executable

***totalview ex1 &***

If everything is setup and working correctly, including your XWindows environment, you should then see TotalView's Root and Process windows appear, loaded with your Exercise 1 program.

### Run the program

Use any of the following methods to start running the program (remember at least one method for later):

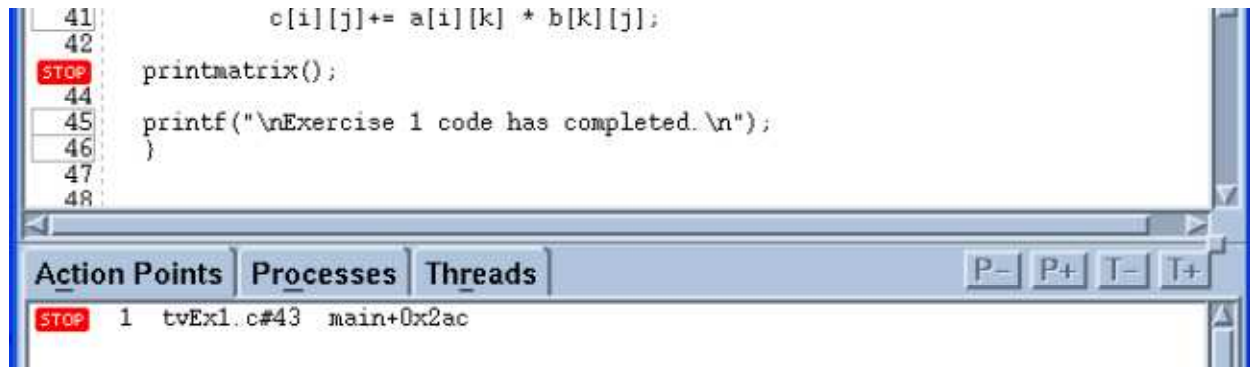
1. Accelerator key: Type **g** in the **Process Window**
2. Go Button: Selected from the Process Window's **execution control button** panel.
3. Process/Group Menus: Select **Go** from any of these three pull down menus in the Process Window.

Note that since no breakpoints were set, the program simply runs to completion. Note also that the program's output is displayed in the window where you started totalview.

### Set a breakpoint

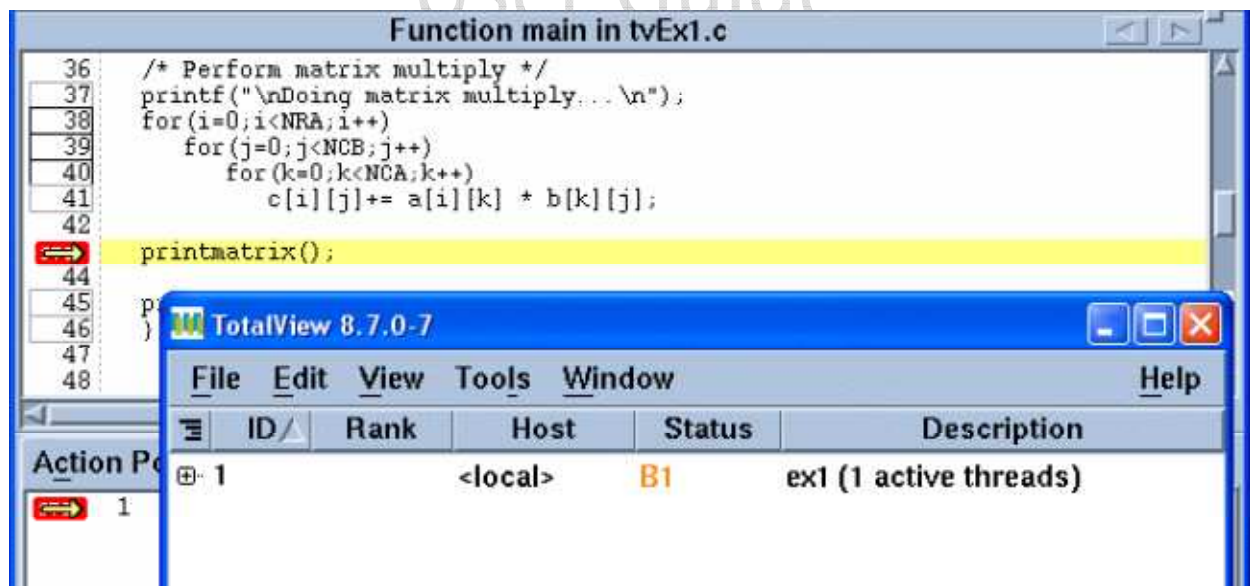
In the Process Window's Source Pane, left-click on the box for line 43. A STOP icon will appear here and also in the Action Points Pane, indicating that the breakpoint has been set (shown below).

Note that this is just one of several ways to set a breakpoint - it is probably the easiest and quickest however.



### Run the program again

1. Use any of several methods to run the program.
2. When the program hits the breakpoint and stops, notice the Program Counter (yellow arrow) on line 43.
3. Check the Root Window and note the **B** breakpoint status code.





**Dive on a routine to view its source code**

This can be done several ways. Only one is described here.

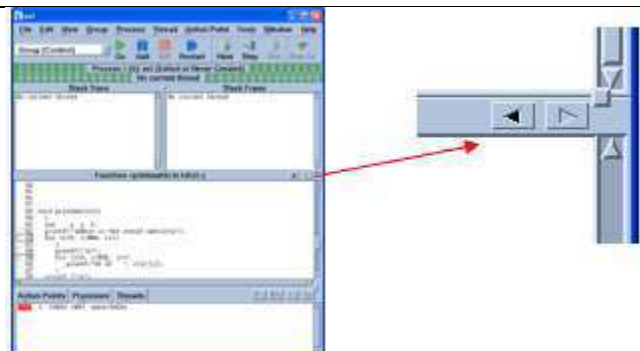
1. Find the routine call to **printmatrix** on line 43. Then, right-click (and hold) on the actual word for **printmatrix**
2. When the pop-up window (shown at right) appears, select the **Dive** option.
3. The routine should now appear in the Source Code Pane.

**Undive from a routine**

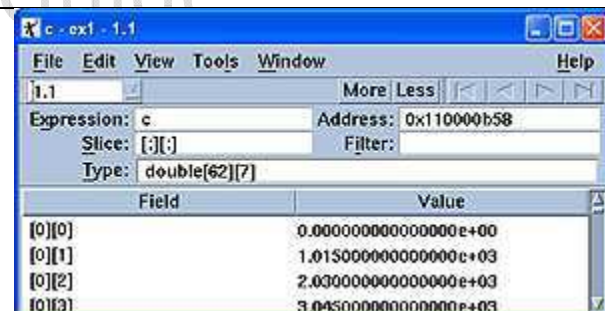
As with diving, this can be done several ways.

Only one is described here.

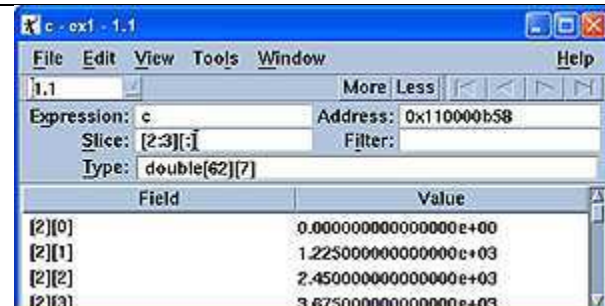
1. Find the undive button located in the upper right corner of the Source Code Pane (shown at right)
2. Left-click on the undive button
3. This will return you to the source code for the main program.

**Dive on an array variable**

1. Find the array variable **c** - several occurrences of it appear between lines 35-42. Double left-click (another way of diving) on any occurrence.
2. This will cause a new Variable Window to open, displaying the contents of the array (shown at right).
3. Try diving on other arrays (or any variable). Each will open a new Variable Window.
4. Leave the array **c** Variable Window open for the next step.

**Display an array slice**

1. In the Variable Window for array **c**, find the box that says: **Slice**:
2. Left-click on the array dimension brackets. For C these will look like **[::]** and for Fortran they will look like **(::)**.
3. This will invoke the line editor, allowing you to type in a range of array elements, called a "slice".
4. Try typing in a slice as shown at right.

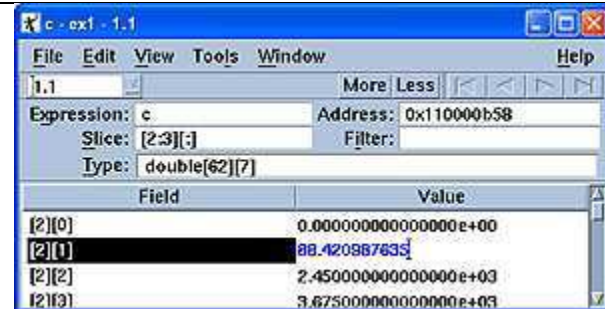




5. When you hit return, the array slice you specified will appear in the Variable Window. Scroll through the window contents to prove this.

#### Modify a variable value

1. Scroll to any of the array elements shown in the Variable Window for array **c**.
2. Left-mouse click on the element's value field to invoke the line editor, allowing you to modify the value (shown at right).
3. Hit return for the modification to take effect.
4. If you want to confirm that the modification took effect, close the Variable Window. Then, dive on array **c** again to open a new Variable Window. Find the array element you modified and verify that it was changed.



#### Stepping

1. Try stepping through the code's execution. This can be done several ways:

Accelerator key: Type **s** in the [Process Window](#)

Step Button: Selected from the Process Window's [execution control button](#) panel.

Process/Group/Thread

Menus:

Select **Step** from any of these three pull down menus in the Process Window.

2. Continue to step through the program several times, noticing what occurs in the Source Code Pane.

3. When you are finished with stepping, **Go** the program so that it completes execution.

#### Get Help

1. In either the Root Window or the Process Window, pull down the Help Menu.

2. Select "Documentation"

3. Then, wait (and wait) until a new web browser window appears with the TotalView online help.

4. Try viewing the Users Guide and/or other online docs.

5. Now try bringing up context-sensitive help. This is done by left-clicking on any window, pane, etc. and then selecting **Help** or hitting the **F1** key.

6. Then wait (and wait) until the context-sensitive help information appears in the web browser window previously opened above.

7. Close the web browser help window when you are done.

#### Quit TotalView

Use either of the following methods to quit the debugger:

Accelerator key: Type **CTRL-Q** in the Root Window

Menu: Select **Exit** from the Root Window's [File Menu](#)

#### Compile the Exercise 2 example code

This exercise assumes that you have completed Exercise 1, and that your environment is still setup to run TotalView.

For C code: `icc -g tvEx2.c -o ex2`

For Fortran: `ifort -g tvEx2.f -o ex2`

### Run the executable

In the same window where you just compiled, run your executable by simply entering the command:

`ex2`

What happens? It should crash and dump core

Check for a core file - note its name - it should be *somelongstring.core*)

### Start TotalView with the core file and determine why the program crashed

Enter the command: `totalview ex2 somelongstring.core &`

When the Process Window appears, the yellow PC should be pointing to the statement that caused the program to crash. Examples below.

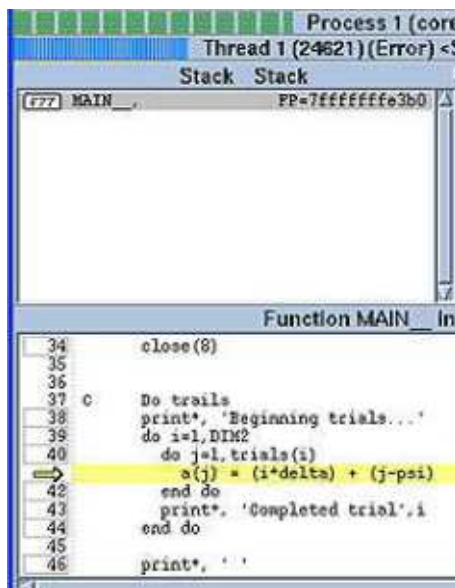


Figure 1 Fortran

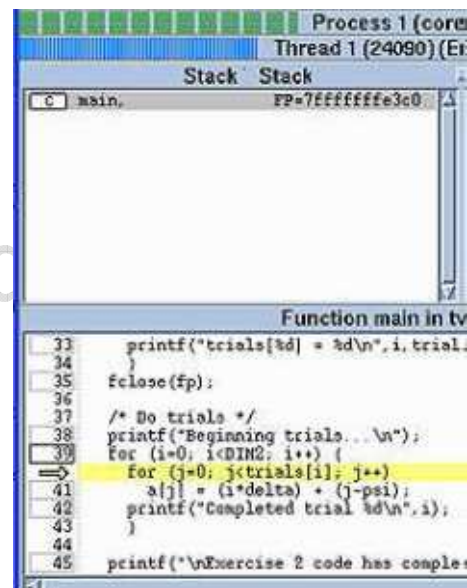


Figure 2 C

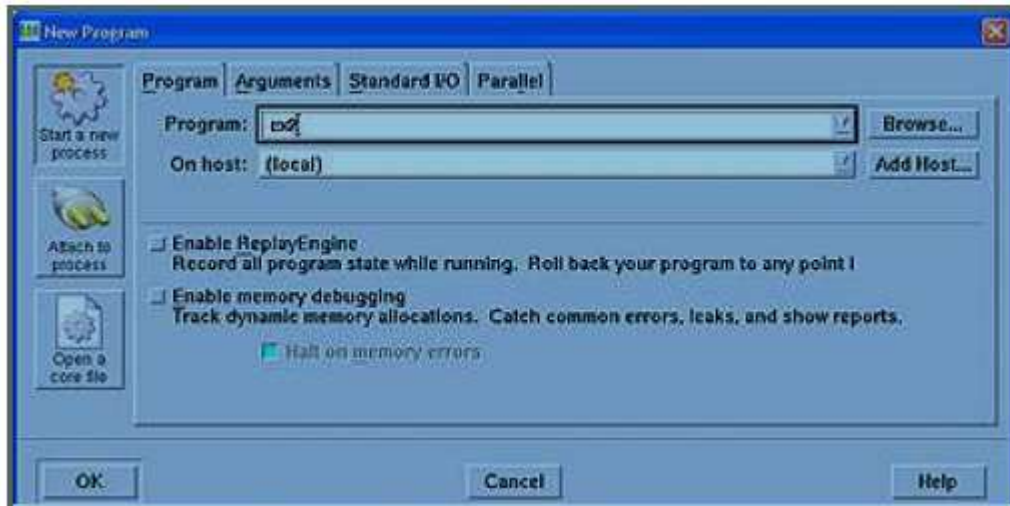
3. Figure out why this program crashed. Hint: dive on the variables/array indices that appear in line 41.

4. After you are satisfied that you know why the program crashes, **CLOSE THE PROCESS WINDOW** so you don't confuse yourself with the next step.

### Begin debugging the crashed program by loading the executable

In order to perform further debugging the actual executable, not the core file, must be loaded. One way of doing this is shown below:

1. Select the **New Program** command from the Root Window's **File Menu**.

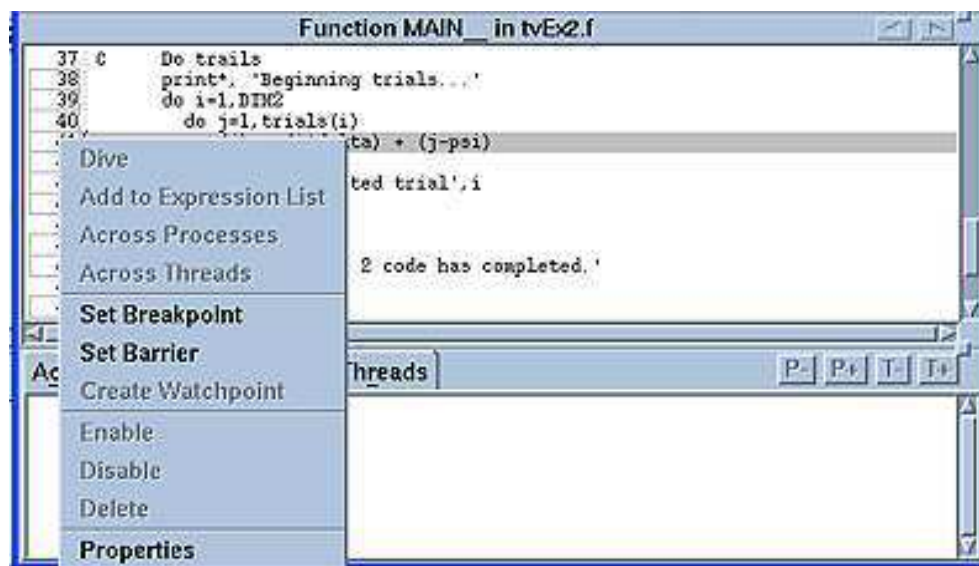


2. The **New Program Dialog Box** will appear (shown at right). Enter the program name **ex2** in the **Program** box.
3. Click the **OK** button. A new Process Window will appear and display the source code for the **ex2** program.

### Set an Evaluation Point to trap the bug

Assuming that you reached the conclusion that the program crashes due to an array boundary condition, setup a test using an Evaluation Point to prove your hypothesis. One way of doing this is shown below.

1. Open a pop-up menu by right-clicking (and holding) on the statement in line 41.
2. When the pop-up menu appears, select the **Properties** command (below).



3. An **Action Point Properties Dialog Box** will then appear.
4. Select the **Evaluate** button and make sure the correct language is also selected.
5. Enter an expression (C or Fortran) to trap the array boundary problem, as shown below.

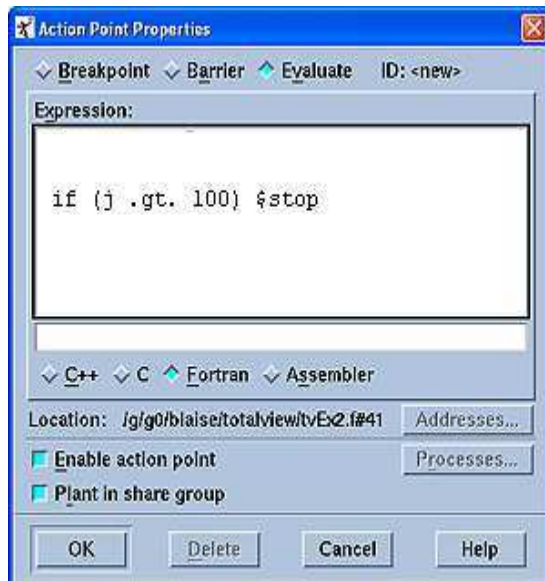


Figure 3 Fortran

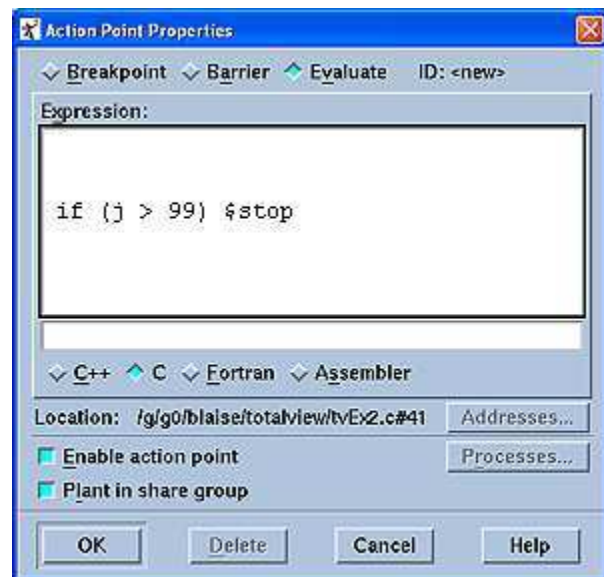
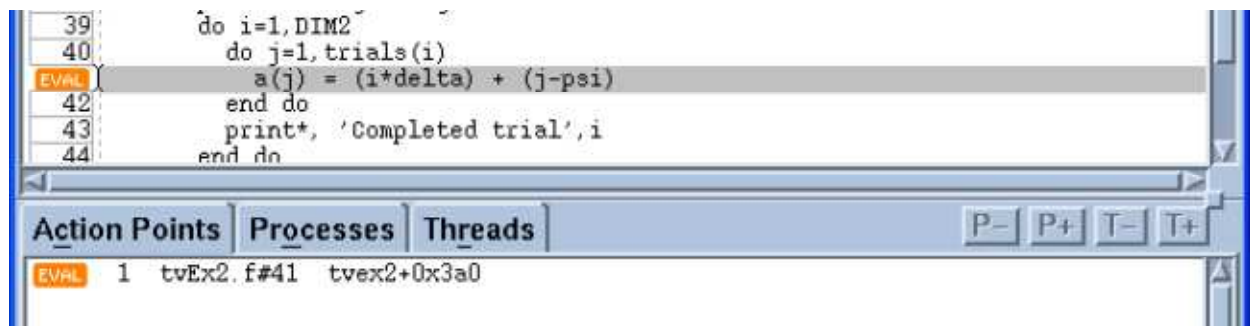


Figure 4 C

6. Click the **OK** button when completed. The source line and the Process Window's Action Points Pane should now display an EVAL icon (below).



### Run the program and catch the bug

1. Go to the program by any of the methods you already know. The program should now run until it triggers the Evaluation Point condition you coded in the previous step.
2. When the Evaluation Point condition is triggered, the program will stop. The PC will point to the problem.
3. On the line where your Evaluation Point occurs, dive on the *j* index for the array *a*. When the new Variable Window opens, inspect its value. Is it out of bounds as an index on array *a*?

### Modify your Evaluation Point to patch around the bug and finish execution

1. Right-mouse click (and hold) on the source line where your Evaluation Point occurs, to open a pop-up window. Then select **Properties** from the pop-up menu.
2. An **Action Point Properties Dialog Box** will open. Note that it is displaying your previously entered Evaluation Point. Modify the code as shown below, to "patch" around the problem. The "patch" simply skips the program out of the crash causing loop.

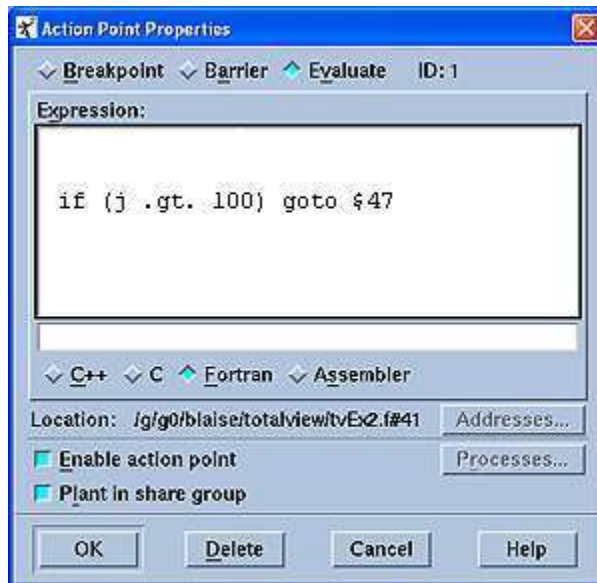


Figure 5 Fortran

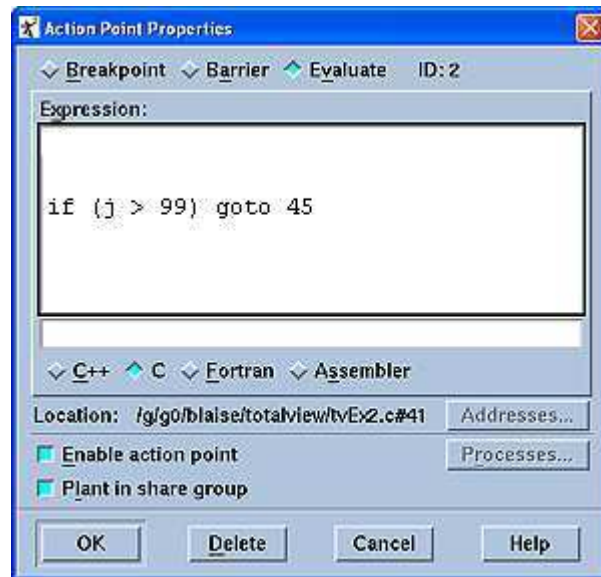


Figure 6 C

3. Click the **OK** button when done.
4. Resume (**Go**) execution by any of the methods you already know. The program should now complete without crashing. Note that in the real world, you would now want to go back and fix your source code.
4. **Quit TotalView when you are done.**

## User Guide

### Attach to a hung process

This part of the exercise will be **very CPU intensive** on the machine where it executes.

**Please make sure that it is terminated before quitting!!!**

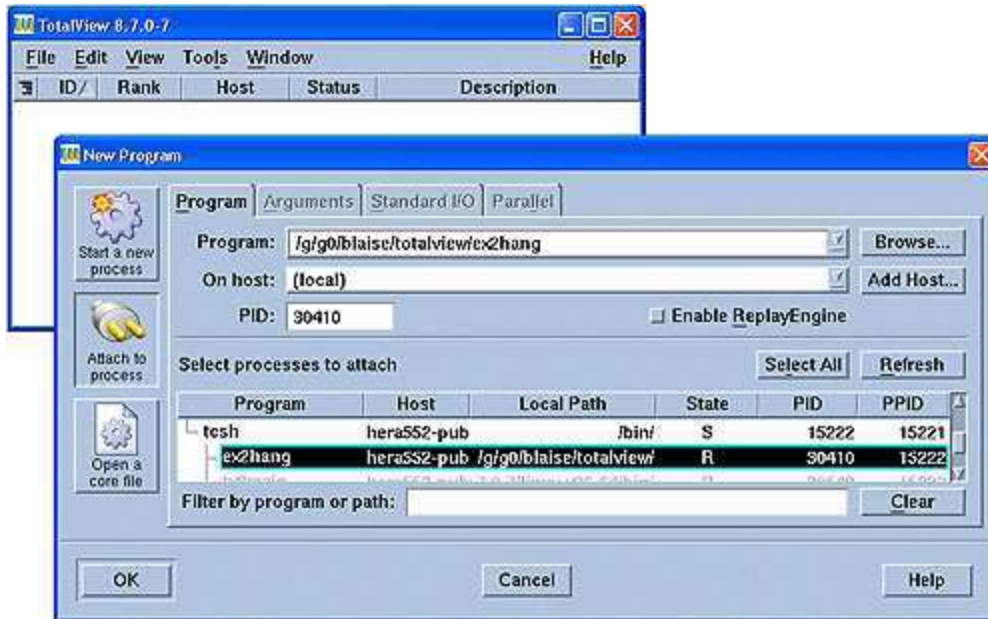
1. Compile the example program.

**C:** `icc -g tvEx2Hang.c -o ex2hang`

**Fortran:** `ifort -g tvEx2Hang.f -o ex2hang`

2. Start the program that will hang: On the same machine where you are running TotalView, start the program, and then verify that it is running (and consuming lots of cpu cycles). At the Unix prompt:  
`ex2hang &`  
`ps uf`
3. Start TotalView by itself: `totalview &`
4. The Root Window should appear along with the New Program Dialog Box.
5. In the **New Program Dialog Box** select **Attach to an existing process**. A list of attachable processes will then display (shown below).
6. Select **ex2hang** and click **OK**.
7. A new Process Window will appear containing the running ex2hang process.





### Debug the hung process

1. First, Halt the hung process by using any of the following methods:

Accelerator key: Type **h** in the [Process Window](#)

Halt Button: Selected from the Process Window's [execution control](#) button panel.

Process Menu: Select **Halt** from the Process Window's [Process Menu](#)

Group Menu: Select **Halt** from the Process Window's [Group Menu](#)

2. Examine the source code and determine the problem. The reason why this trivial program is hung is rather obvious.

3. Dive (by any means you choose) on the index variable *i*. A new Variable Window will open.

4. In the new Variable Window, leftclick on the variable *i* value to invoke the field editor.

5. Modify the variable's value so that the condition causing it to hang is resolved. Simply make *i* greater than 100 as shown at the right.

6. Hit return for the modification to take effect.



**Resume execution of the hung process**

Resume (**Go**) execution of the hung process now that you've "debugged" it. Use any of the methods you already know.

The hung process should now complete execution.

**Make sure the hung process is gone**

At the Unix prompt, issue the `ps` command to verify that the process successfully terminated. If not, then use the `kill pid` command to kill the process, where *pid* is the process ID number as shown by the `ps` command.

**Quit TotalView**

Use either of the following methods to quit the debugger:

Accelerator key: Type **CTRL-Q** in the Root Window

Menu: Select **Exit** from the Root Window's [File Menu](#)

**TotalView Exercise for Part 3: Debugging Parallel Codes**

This exercise assumes that you have completed [Exercise 1](#), [Exercise 2](#), that you are still logged into the workshop machine, and that your environment is still setup to run TotalView.

**Debugging OpenMP Programs****Compile the Exercise 3 OpenMP example code**

**C:** `icc -mp -g tvEx3omp.c -o ex3omp`

**Fortran:** `ifort -mp -g tvEx3omp.f -o ex3omp`

**Specify 4 threads and start TotalView with your executable**

`setenv OMP_NUM_THREADS 4`

`totalview ex3omp &`

**Review the source code**

In this simple example, the master thread first initializes two vectors A and B, and then spawns a parallel region. Inside the parallel region, threads share the work of summing A and B into a third vector, C, by using the OpenMP `DO` (Fortran) or `for` (C) directive. Note the scoping of the variables used in this program.

**Set two breakpoints**

Set breakpoints on lines 48 and 53. The first breakpoint occurs inside the parallel region, and will affect all threads. The second one occurs outside the parallel region and will only affect the master thread.

**Run the program**

**Go** the program. After the first thread hits the line 48 breakpoint, check to see if all 4 threads have been created. This can be done by expanding the toggle in the Root Window, or selecting the Threads tab in the

bottom pane of the Process Window.

If you don't see 4 threads, **Go** the program again...until all 4 threads appear.

**Find where thread information is displayed**

1. Process Window: click on the Threads Pane
2. Process Window: note the status bar that shows the process and thread ids, and also a unique color/pattern
4. Root Window: click the process toggle to show the thread list



**Cycle through all threads**

1. Use the **T-** and **T+** buttons
2. Left-click on any thread in the Threads Pane
3. Double left-click (Dive) on any thread in the Root Window list
4. As you cycle through each thread, notice what changes in the Process Window (status bar info and colors/patterns, info in the various panes, etc.)
5. It is possible that some threads will be in library or system calls, in which case you won't see source code.

**Open a new Process Window for at least one other thread**

This can be done by selecting any thread (other than the current thread) in the Root Window thread list, and then selecting **Dive in a New Window** from the Root Window's **View Menu**

**View SHARED and PRIVATE variables**

1. Dive on any of the variables scoped as SHARED and/or PRIVATE.
2. Cycle through the threads doing this.
3. You may notice that some variables in some threads have "garbage" values because the thread may not be far along enough yet.

**Display a variable's value across all threads**

1. Source Pane: Find an occurrence of the `tid` variable.
2. Dive on it (double left-click or right click menu) - a new Variable Window will appear
3. In the Variable Window, select the **View Menu**, and then **Show Across -> Thread**
4. The Variable Window display will toggle into a laminated display for the `tid` variable. Note the different values for this PRIVATE variable in each thread.
5. Note that if a thread has not yet reached the point where it has obtained its `tid`, the laminate window will say "Has no matching call frame" for that thread.

**Disable the first breakpoint**

1. In the **Action Points Pane** (lower right corner of the Process Window), left-click on the first breakpoint icon.
2. Notice that the red **STOP** icon is now dimmed in both the Source Pane and the Action Points Pane. This means that it is disabled (not deleted).

**Finish program and quit TotalView**

1. **Go** the program again and observe the program's output in the window where you started totalview.
2. Continue to **Go** the program and observe the output until the program completes.
3. Quit TotalView

**Debugging MPI Programs****1. Compile the Exercise 3 MPI example code**

**C:** `mpicc -g tvEx3mpi.c -o ex3mpi`

**Fortran:** `mpif77 -g tvEx3mpi.f -o ex3mpi`

**2. Start TotalView using srun and your executable**

1. Issue the command:  
`totalview srun -a -n4 -ppReserved ex3mpi &`
2. The `srun` process will appear in the Root and Process startup windows.
3. **Go** the process so that your parallel job gets launched
4. Eventually, a dialog box will appear asking if your parallel job should be stopped now. Select **Yes**
5. Your MPI task 0 should now appear in the Process Window and a list of all MPI tasks plus the

srunk process should appear in the Root Window.

### 3. Review the source code

The header comments explain what's happening with this program. It follows the SPMD (Single Program Multiple Data) programming model, which means the same program is executed by all MPI tasks. Note however, that there are sections of code that are executed by the master task (0) only, by non-master tasks only, and by all tasks.

### 4. Find where MPI task information is displayed

Root Window: list of MPI tasks

Process Window: status bar information and colors/patterns

### 5. Experiment with breakpoints

The whole point of this section is to familiarize you with the behavior and options associated with action points...using breakpoints (the simplest) as an example. The default behaviors may or may not be what you think or want.

1. First, set a breakpoint on line 53 by left-clicking the line box. Note that this line occurs in the master (task 0) section of code. Other tasks can not execute it.

2. **Go** the Group. What happens when the master task hits the breakpoint? Notice that the other MPI tasks keeps running when the master task hits the breakpoint.

3. Now, delete the breakpoint on line 53 (simply left-click the red stop icon again) and set a new one on line 73, which is still in the master only section of code - **however** this time right-click (and hold) until the pop-up menu (at right) appears and then select **Properties**.

4. When the **Action Point Properties Dialog Box** appears you will see the current properties for this breakpoint. Override this behavior to stop all processes by clicking on the **Group** toggle. Then click **OK**.

5. **Go** the Group. What happens this time when task 0 hits the breakpoint? Notice that the non-master tasks now stop even though they don't hit the breakpoint.

6. Moral of the story: know how you want your breakpoints to behave.

### 6. Notice that your MPI processes are multi-threaded

You may already have noticed at this point that each MPI task is actually multi-threaded. Note that only one of these threads is of interest - the one which is executing your code. The others are created by the system or MPI library and are ordinarily not of interest to you.

### 7. Cycle through all MPI tasks

1. Use the **P-** and **P+** buttons of the Process Window

2. Double left-click (dive) on any MPI process in the Root Window list

3. As you cycle through each process, notice what changes in the Process Window (status bar info and colors/patterns, info in the various panes, etc.)

Ignore the srunk process, by the way.

4. **Note:** if you don't see source code as you cycle through the MPI processes, click on the main program name in the Stack Trace Pane.

### Set a barrier point accepting its default properties

1. Find the call to the MPI\_Finalize routine - at line 122.

2. Right-click (and hold) to raise a pop-up menu. Then select **Set Barrier**.

3. A blue Barrier icon will then appear on the line number and in the Action Points Pane.

### Run the program

1. **Go** the Group. All MPI tasks will now execute until they hit the barrier point. The Root Window will indicate this by giving each task a **B** status code.

2. Cycle through all of the MPI tasks - they should all be at the barrier point. Ignore the srunk task, by the way.

### 10. Open a new Process Window for a different MPI task

This can be done by selecting an MPI task in the Root Window's process list, and then selecting **Dive Anew** from the Root Window's **View Menu**.

**11. Examine variables**

Experiment. Dive on any variables of your choice. Compare between tasks. Dive from the Source Pane or the Stack Frame Pane.

**12. Display variables across processes (lamine)**

1. Remember that laminated variables only have meaning if they exist between multiple tasks (or threads, as seen previously).
2. Two suggested variables to laminate include **offset** and **mysum**. They are unique for each MPI task.
3. Dive on these variables. Each will open a new Variable Window.
4. Pull-down the **View** menu and then select **View Across -> Process**
5. Note the different values between tasks.

**13. Finish execution and quit TotalView**

1. **Go** the Group. The program should complete.
2. Quit TotalView.
3. We're done. Pfew.

# User Guide

**Last updated: 07/20/2012**

**For Additional Assistance Contact: [hpcsupport@ttu.edu](mailto:hpcsupport@ttu.edu)**

**For Comments/Suggestions on user guide [hpc@ttu.edu](mailto:hpc@ttu.edu)**

# User Guide