RogueWave SOFTWARE
Accelerating Great Code

Evaluating TotalView® for HPC

1 Compiling Programs. Compile your programs using the –g option. For example:

gcc -g –o *my_prog my_prog.c*

2 Starting TotalView. Enter:

totalview *my_prog* **–a** *arguments*

Or, type **totalview** from the shell to open the Sessions Manager to:

- Start a new program or parallel program
- Attach to a running process
- Open a core file
- Manage your debug sessions

3 Toolbar Buttons Defined

- **Go**: starts execution.
- **Halt**: stops execution, but you can restart from where execution stopped.
- **Kill**: kills the executing program.
- **Restart**: does a **Delete**, then a **Go**.
- **Next**: executes all code on the current line; program counter (PC) will be at the next line.
- **Step**: executes line; if the line has a subroutine, PC moves into it.
- **Out**: executes remainder of current routine; PC is on the line that called this routine.
- **Run To**: After selecting a line (click on the line, not the line number), press this button to execute all instructions from the PC until this line.

4 Setting a Breakpoint

- **Line**: click on a line number.
- **Function**: select **Action Point > At Location**, and type a function name.
- **Function**: Use the **View > Lookup Function** command, then click the line number.
- **Search**: Use the **Edit > Find** command, then click on the line number.

5 Attaching to Already Running Programs

- Select the **File > Attach to a Running Program** command from the Root or Process windows to launch the Sessions Manager, and browse to the program.
- If you don't see the program, use the **ps** command to find its PID (Program ID), and then select the PID within the dialog box.

Always attach to a program's main thread.

6 Stopping at a Line When a Variable Equals (or Doesn't Equal) a Value

a Set a breakpoint within the loop.
b. Right click on the breakpoint icon and select **Properties**.
c. Select **Evaluate** in the dialog box.
d. Type a condition; for example:

```
if (my_variable == 0) $stop
```

7 Seeing Variable Values

- If it's a local variable, it'll be in the Stack Frame Pane. For a local or global variable, double-click it in the Source Pane to see the value in a Variable Window, or hover your cursor over it to see the value.
- If it is not a complex variable (that is, it is not an array or a structure), right-click on the variable and select **Add to Expression List**.
- For arrays and structures, double-click to see all values in a Variable Window.

8 Chasing Pointer Values. If a variable's type is a pointer, double-click to see the value being pointed to.

9 Seeing Many Variables at the Same Time. You can send as many variables as you want to the Expression List window. The values in this window update every time your program stops executing.

You can also send individual structure and array elements to this window.

10 Seeing Just Some of an Array's Elements. The **Slice** area within the Variable Window lets you tell TotalView which array elements it should display. For example, typing (31:60) in Fortran or [30:59] in C or C++ restricts the display to just 30 elements.

Type a condition within the **Filter** area to restrict the display to certain values. For example, typing **> 64000** restricts the display to array elements with a value greater than 64,000.

You can combine slices and filters.

11 Graphing Arrays. Seeing array data visually is an easy way to detect outliers and patterns. Display the data graphically by selecting the **Tools > Visualize** command within a Variable Window.

12 Casting. You can change the way TotalView interprets and displays variable data by editing the **Type** field of a variable window.

For example, if you have a pointer to an array, you'll want to change the datatype from something like **int *** to **int[100] *** to see array or pointer elements.

13 Changing Variable Values

- In the Expression List and Variable Windows, click a value and edit it.
- In the Stack Frame Pane, double-click a boldface number, then edit it.

14 STL Variables. TotalView provides automatic STL type transformations to more clearly display STL data without the underlying structure. This can be toggled in the preferences as preferred.

15 Searching For Variables. Select **View > Lookup Variable** from the Process Window. The variable displays in a Variable Window.

16 Stopping Execution When a Variable's Value Changes. Use the **Tools > Create Watchpoint** command.

If the Variable Window is displaying an array or a structure, you'll need to dive on an element so that only one of the variable's elements is displayed.

17 Seeing One Element in an Array of Structures as its own Array

a Select one element.
b. Right-click and select **Dive in All**.

The window now displays an array containing just those elements.

18 Seeing a Variable's Value in Multiple Threads or Processes. From the Variable Window menu, select:

- **View > Show Across > Thread** if the program is multi-threaded, or
- **View > Show Across > Process** if the program is multi-process.

In the Stack Frame or Source Pane, right-click on the variable and select **Across Processes** or **Across Threads**.

19 CLI Command Entry. Select the **Tools > Command Line** command. You can now type TotalView CLI commands within this window. Type **dhelp** for help.

20 Debugging with fork() and Execve() Programs. In most cases you must link your program with the libdbfork library that we provide. See our reference guide for more information

21 Debugging with ReplayEngine. ReplayEngine is an add-on for reverse debugging in Linux x86 and x86-64. Start it before a debugging session either from:

- the Debug Options dialog in the Sessions Manager, by selecting **Enable ReplayEngine**.
- the Process Window, by selecting menu option **Debug > Enable ReplayEngine.**

The ReplayEngine buttons on the toolbar are as follows:

- **Record:** Starts ReplayEngine before or after a process has started.
- **GoBack:** Runs backwards to the nearest stop event.
- **Prev:** Moves execution in reverse, over function calls; PC moves to previous line.
- **UnStep:** Moves execution in reverse, through functions; PC moves into function calls.
- **Caller:** PC returns to the point before the function was called.
- **Back To:** When a line is selected, moves execution in reverse to the most previous execution of the line.
- **Live:** Execution and the PC are returned to current live execution location.
- **Save**: Displays a dialog to save the current recording session to a file to be loaded in at a later time.