TotalView

Debugging Tool
Presentation

Josip Jakić
josipjakic@ipb.ac.rs

Agenda

- Introduction
- Getting started with TotalView
- Primary windows
- Basic functions
- Further functions
- Debugging parallel programs
- Topics not covered
- References and more information

Introduction [1/2]

- TotalView is a sophisticated software debugger product from Rogue Wave Software, Inc.
- Used for debugging and analyzing both serial and parallel programs
- Designed for use with complex, multi-process and/or multi-threaded applications
- The most popular HPC debugger to date

Introduction [2/2]

- Supported on most HPC platforms
- Provides both a GUI and command line interface
- Includes memory debugging features
- Supported languages include the usual HPC application languages:
  - C/C++
  - Fortran77/90
  - Assembler

Getting started with TotalView [1/3]

- **-g** flag enables generation of symbolic debug information for most compilers
- Programs compiled without the **-g** option are allowed to be debugged, however, only the assembler code can be viewed
- Programs should be compiled without optimization flags
- Parallel programs may require additional compiler flags

Getting started with TotalView [2/3]

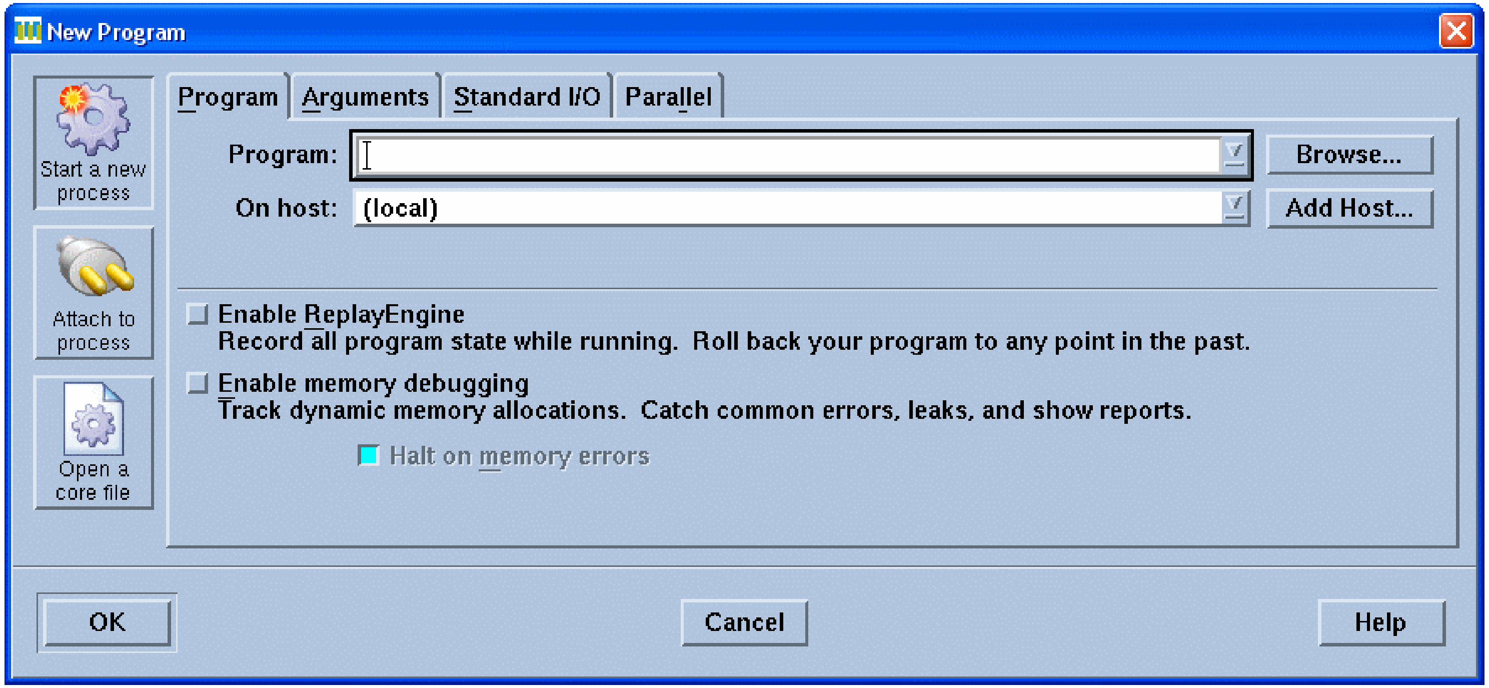
- A variety of ways to start the program
  - totalview *(invokes* New Program dialog box*)*
  - totalview *filename*
  - totalview *filename corefile*
  - totalview *filename* -a *args*
  - totalview *filename* -remote *hostname [:portnumber]*

WWW.SCL.RS

SCIENTIFIC
COMPUTING
LABORATORY

Getting started with TotalView [3/3]

- New Program dialog box



Program:

On host: (local)

□ Enable ReplayEngine
Record all program state while running. Roll back your program to any point in the past.

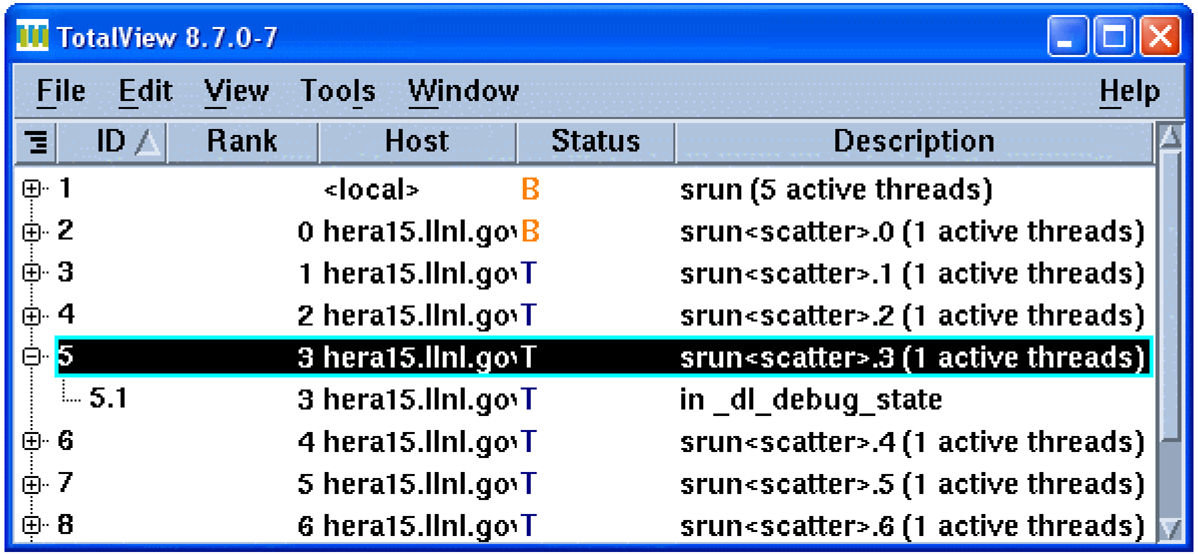
□ Enable memory debugging
Track dynamic memory allocations. Catch common errors, leaks, and show reports.

☑ Halt on memory errors

  – Numerous options for various means of selecting a program

Primary Windows [1/7]

- Root Window

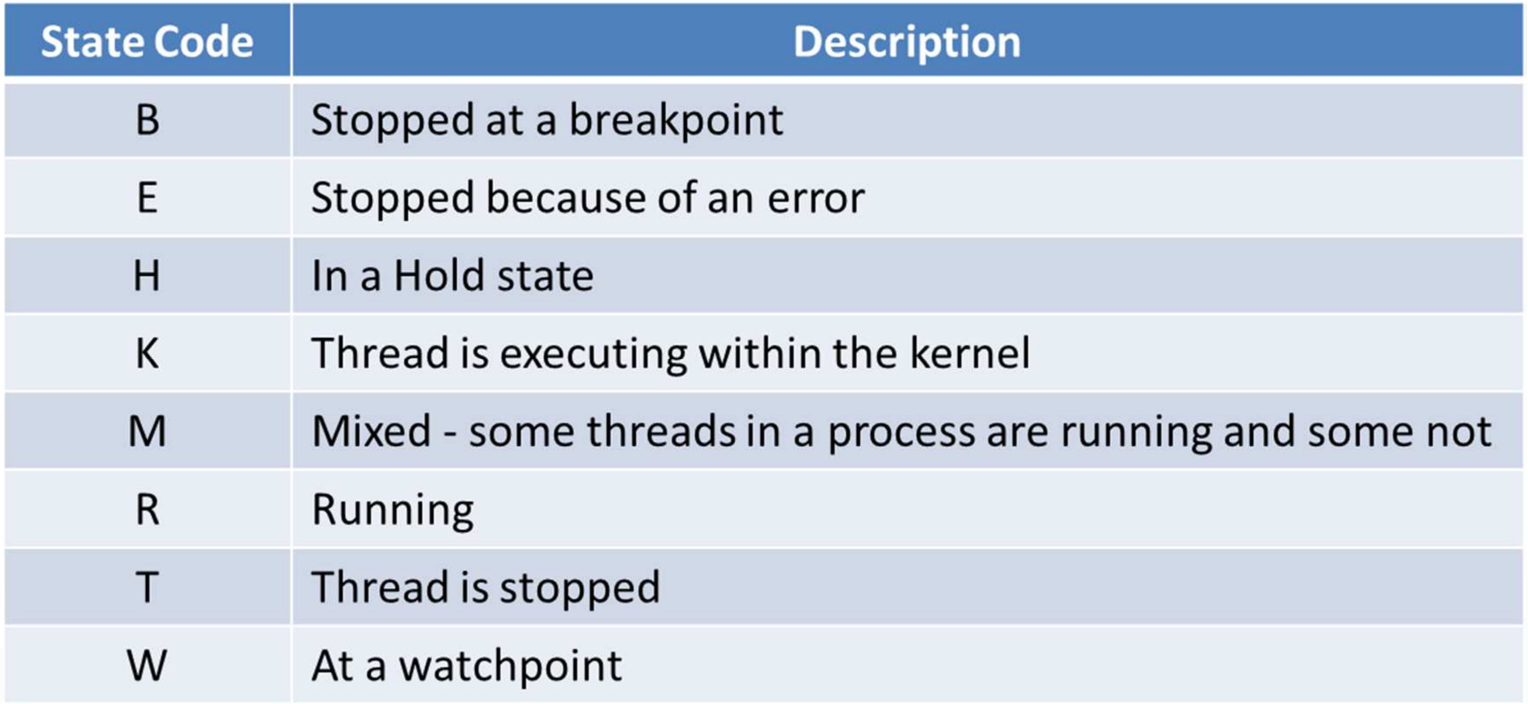


ID	Rank	Host	Status	Description
1		<local>	B	srun (5 active threads)
2	0	hera15.llnl.go	B	srun<scatter>.0 (1 active threads)
3	1	hera15.llnl.go	T	srun<scatter>.1 (1 active threads)
4	2	hera15.llnl.go	T	srun<scatter>.2 (1 active threads)
5	3	hera15.llnl.go	T	srun<scatter>.3 (1 active threads)
5.1	3	hera15.llnl.go	T	in _dl_debug_state
6	4	hera15.llnl.go	T	srun<scatter>.4 (1 active threads)
7	5	hera15.llnl.go	T	srun<scatter>.5 (1 active threads)
8	6	hera15.llnl.go	T	srun<scatter>.6 (1 active threads)

- Appears when the TotalView GUI is started
- Overview of all processes and threads, showing assigned ID, MPI rank, host, status and brief description/name for each

Primary Windows [2/7]

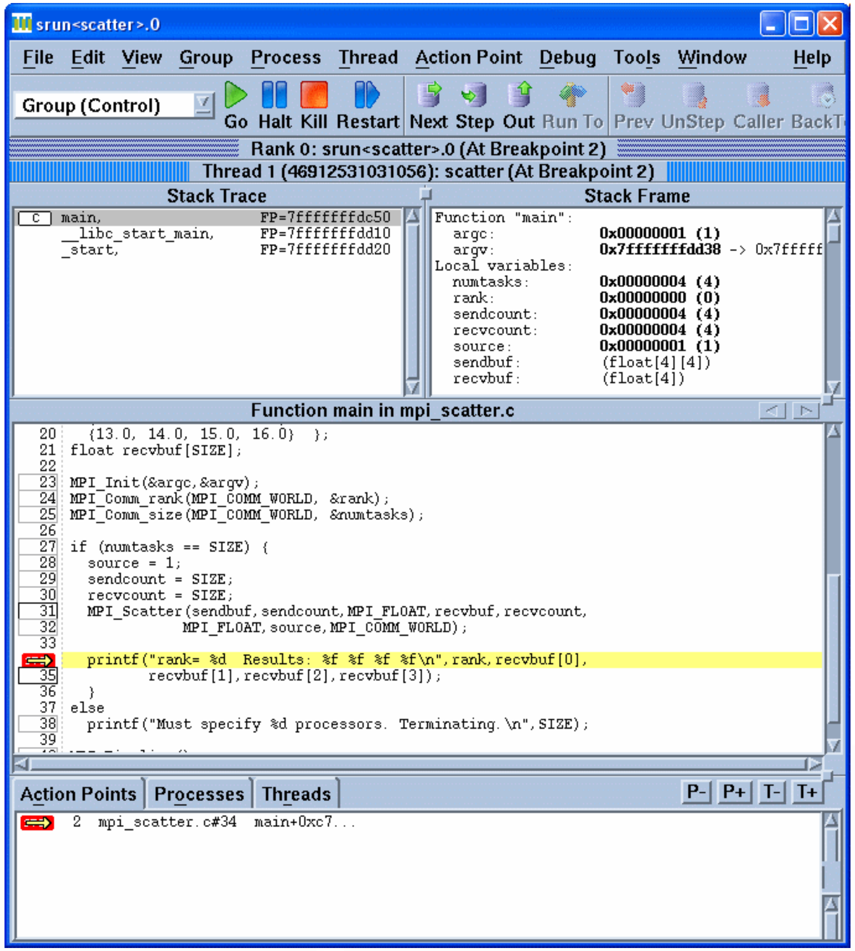
- Root Window
  - Process and Thread State Codes



State Code	Description
B	Stopped at a breakpoint
E	Stopped because of an error
H	In a Hold state
K	Thread is executing within the kernel
M	Mixed - some threads in a process are running and some not
R	Running
T	Thread is stopped
W	At a watchpoint

Primary Windows [3/7]

- Process Window



Rank 0: srun<scatter>.0 (At Breakpoint 2)

Thread 1 (46912531031056): scatter (At Breakpoint 2)

```
C  main,                  FP=7fffffffdc50
   __libc_start_main,     FP=7fffffffdd10
   _start,                FP=7fffffffdd20
```

```
Function "main":
  argc:                0x00000001 (1)
  argv:                0x7fffffffdd38 -> 0x7fffff
Local variables:
  numtasks:            0x00000004 (4)
  rank:                0x00000000 (0)
  sendcount:           0x00000004 (4)
  recvcount:           0x00000004 (4)
  source:              0x00000001 (1)
  sendbuf:             (float[4][4])
  recvbuf:             (float[4])
```

Function main in mpi_scatter.c

```
  {13.0, 14.0, 15.0, 16.0}  };
float recvbuf[SIZE];

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

if (numtasks == SIZE) {
  source = 1;
  sendcount = SIZE;
  recvcount = SIZE;
  MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
              MPI_FLOAT,source,MPI_COMM_WORLD);

  printf("rank= %d  Results: %f %f %f %f\n",rank,recvbuf[0],
         recvbuf[1],recvbuf[2],recvbuf[3]);
  }
else
  printf("Must specify %d processors. Terminating.\n",SIZE);

```

```
2  mpi_scatter.c#34  main+0xc7...
```

Primary Windows [4/7]

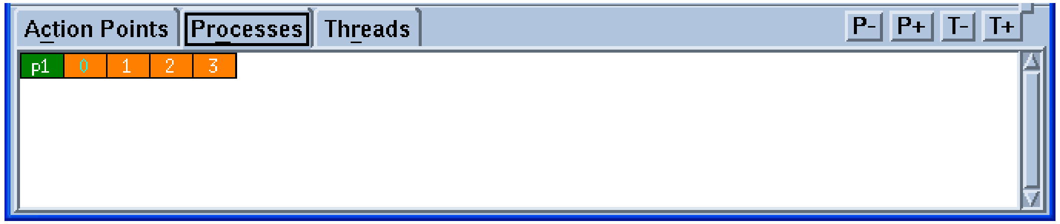
- Process Window
  - For multi-process/multi-threaded programs, every process and every thread may have its own Process Window if desired
  - Comprised of:
    - Pull-down menus
    - Execution control buttons
    - Navigation control buttons
    - Process and thread status bars
    - 4 "Panes"

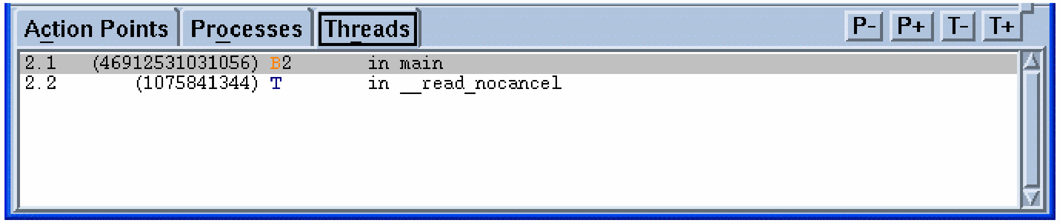
Primary Windows [5/7]

- Process Window
  - – Stack Trace Pane
    - Shows the call stack of routines the current executable is running
  - – Stack Frame Pane
    - Displays the local variables, registers and function parameters for the selected executable.
  - – Source Pane
    - Displays source for the currently selected program or function with program counter, line numbers and any associated action points

Primary Windows [6/7]

- Process Window
  - Action Points, Processes, Threads Pane
    - A multi-function pane. By default, it shows any action points that have been set
    - May also select Processes to show attached processes or Threads to show associated threads

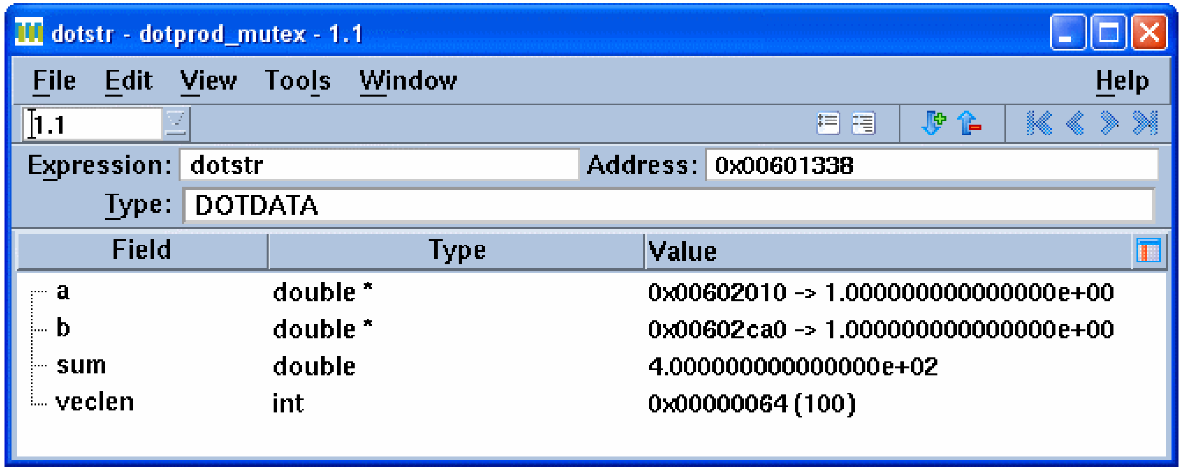




```
2.1   (46912531031056) B2        in main
2.2        (1075841344) T        in __read_nocancel
```

Primary Windows [7/7]

- Variable Window



dotstr - dotprod_mutex - 1.1
File Edit View Tools Window Help
1.1

Expression: dotstr Address: 0x00601338

Type: DOTDATA

Field	Type	Value
a	double *	0x00602010 -> 1.000000000000000e+00
b	double *	0x00602ca0 -> 1.000000000000000e+00
sum	double	4.000000000000000e+02
veclen	int	0x00000064 (100)

  – Displays detailed information about selected program variables
  – Permits editing, diving, filtering and sorting of variable data

Basic Functions [1/6]

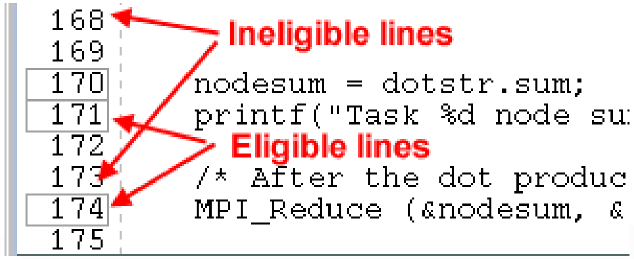
- Viewing Source Code
  - – Source, Assembler or Both
  - – To toggle between the different display modes:
    - Process Window > View Menu > Source As
- Displaying Function / File Source Code
  - – Finding and displaying the source code:
    - Process Window > View Menu > Lookup Function

WWW.SCL.RS

SCIENTIFIC COMPUTING LABORATORY

Basic Functions [2/6]

- Setting a Breakpoint
  - Most basic of TotalView’s action points used to control a program’s execution
  - Halts execution at a desired line before executing the line
  - ˝Boxed˝ lines are eligible for breakpoints

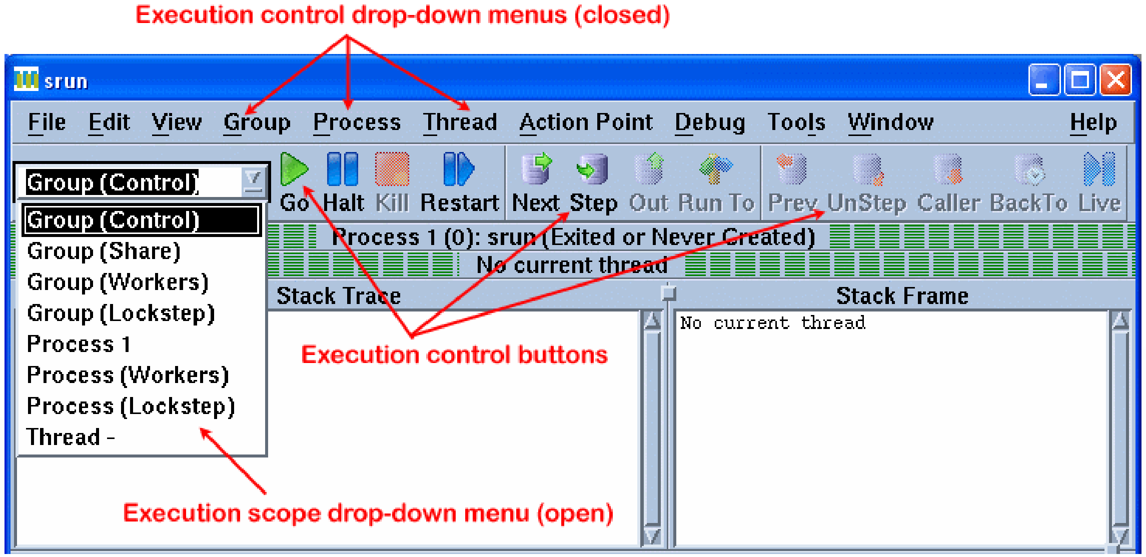


```
nodesum = dotstr.sum;
printf("Task %d node su
/* After the dot produc
MPI_Reduce (&nodesum, &
```

  - Setting and viewing breakpoints

Basic Functions [3/6]

- Controlling the execution of a program within TotalView involves two decisions:
  - Selecting the appropriate command
  - Deciding upon the scope of the chosen command



Execution control drop-down menus (closed)
srun
File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go Halt Kill Restart Next Step Out Run To Prev UnStep Caller BackTo Live
Group (Control)
Group (Share)
Group (Workers)
Group (Lockstep)
Process 1
Process (Workers)
Process (Lockstep)
Thread -
Process 1 (0): srun (Exited or Never Created)
No current thread
Stack Trace
Stack Frame
No current thread
Execution control buttons
Execution scope drop-down menu (open)

Basic Functions [4/6]

- Group, Process, Thread Command Scopes
  - For serial programs, execution scope is not an issue because there is only one execution stream
  - For parallel programs, execution scope is critical - you need to know which processes and/or threads your execution command will effect
    - Additional details about Group, Process and Thread Command Scopes are discussed later together with additional breakpoint options

Basic Functions [5/6]

- Viewing and Modifying Data
  - TotalView allows you to view variables, registers, areas of memory and machine instructions
  - Leaving a Variable Window open allows you to perform runtime monitoring of variables (updated each time program is stopped)
  - You can edit variables from within the Variable Window
  - The modified variable has effect when the program resumes execution

Basic Functions [6/6]

- Arrays
  - For array data, TotalView provides several additional features:
    - Displaying array slices
    - Data filtering
    - Data Sorting
    - Array statistics
  - Array Viewer
    - To view a multi-dimensional array in "spreadsheet" format:
      - Variable Window > Tools Menu > Array Viewer

Examples

- Demonstration on topics covered so far using simple serial code
  - Starting TotalView
  - Primary windows
  - Basic Functions

Further Functions [1/7]

- Viewing a Core File
  - – TotalView can be used to examine the core file from a crashed job and examining the state (variables, stack, etc.) of the program when it crashed
    - It is quite likely that your shell's core file size setting may limit the size of a core file so that it is inadequate for debugging
    - Check your shell's limit settings, use either the **limit** (csh/tcsh) or **ulimit -a** (sh/ksh/bash) command and override if neccessary

Further Functions [2/7]

- Code fragments
  - Code fragments can include a mixture of:C, Fortran or Assembler language
  - TotalView built-in variables ($tid, $pid, $systid ...)
  - TotalView built-in statements ($stop, $hold, $stopall ...)
  - Code fragments can be entered by two methods:
    - Evaluate Window
    - Evaluation Point

Further Functions [3/7]

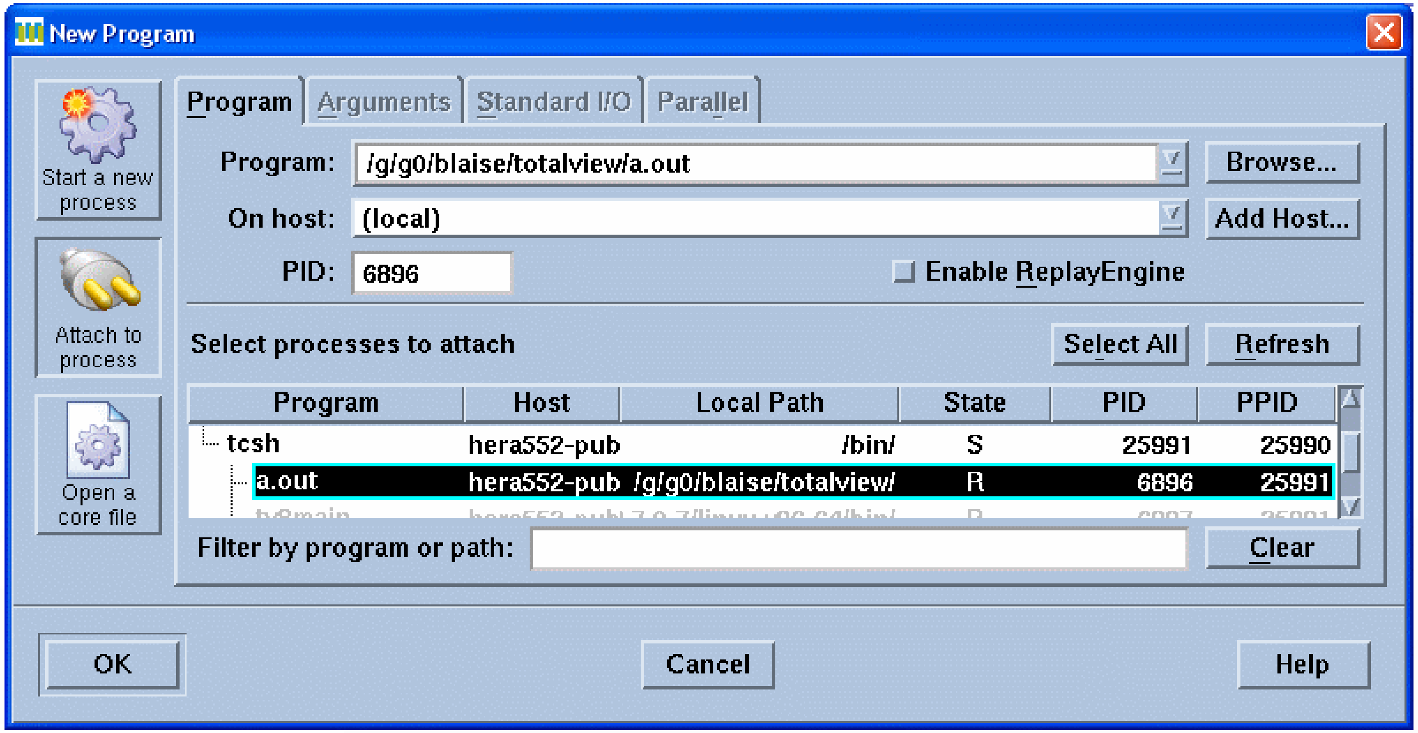
- TotalView supports four different types of action points:
  - Breakpoint
  - Process Barrier Point
    - holds each process when it reaches the barrier point until all processes have reached the barrier
  - Evaluation Point
    - causes a code fragment to execute when reached
  - Watchpoint
    - Monitors when the value stored in memory is modified and either stop execution or evaluates

Further Functions [4/7]

- Managing action points
  - Deleting Action Points
    - **Delete All**
  - Disabling / Enabling Action Points
    - **Suppress All**
  - Saving / Loading Action Points

Further Functions [5/7]

- **Attaching / Detaching Processes**
  - In the **New Program Dialog Box**, select the **Attach to process** button



Program: /g/g0/blaise/totalview/a.out

On host: (local)

PID: 6896

□ Enable ReplayEngine

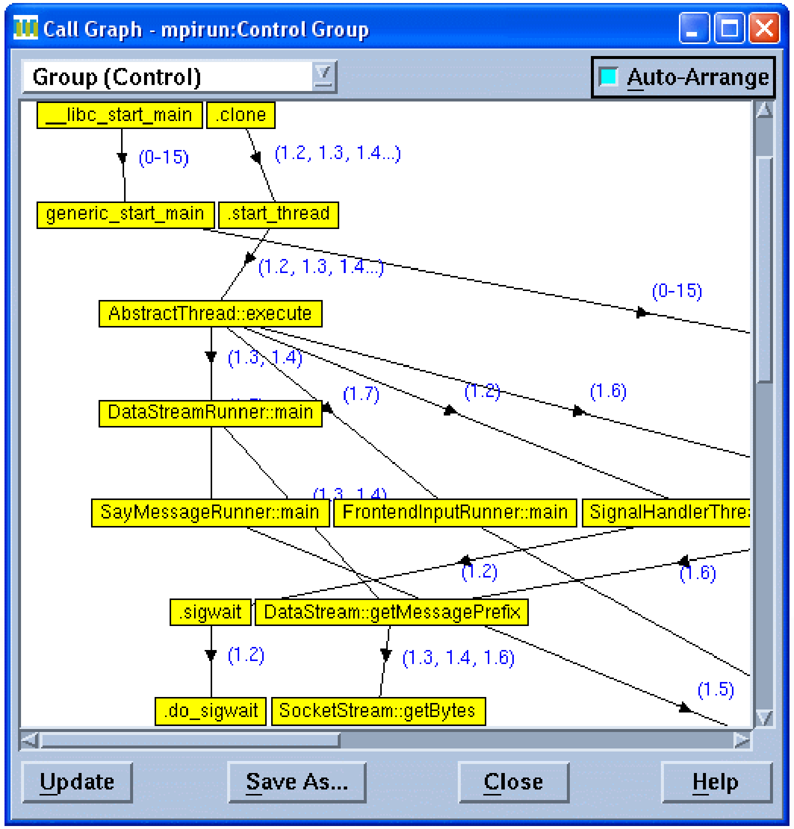
Select processes to attach

Program	Host	Local Path	State	PID	PPID
tcsh	hera552-pub	/bin/	S	25991	25990
a.out	hera552-pub	/g/g0/blaise/totalview/	R	6896	25991
tv8main	hera552-pub	7.0-7/linux-x86-64/bin/	R	6897	25991

Filter by program or path:

Further Functions [6/7]

- **Displaying Program's Call Graph**
  - Process Window > Tools Menu > Call Graph



Further Functions [7/7]

- Some other functions and settings
  - Setting Executable Command Arguments
  - Setting Source Code Search Paths
  - Setting stdin, stdout, and stderr
  - Setting Preferences
  - Signal Handling
  - Debugging Memory Problems
  - Visualizing Array Data
  - Command Line Interpreter (CLI)

Examples

- Start TotalView with the core file and determine why the program crashed
- Setting Evaluation Points
- Attach to a hung process
- Debugg the hung process

Debugging Parallel Programs [1/11]

- Process/Thread Groups
  - Types of P/T Groups:
    - Control Group:
      - Contains all processes and threads created by the program across all processors
    - Share Group:
      - Contains all of the processes and their threads, that are running the same executable
    - Workers Group:
      - Contains all threads that are executing user code
    - Lockstep Group:
      - Includes all threads in a Share Group that are at the same PC (program counter) address

Debugging Parallel Programs [2/11]

- Debugging Threaded Codes
  - Finding Thread Information
    - Root Window
    - Process Window
  - Selecting a Thread
    - Thread Navigation Buttons
  - Execution Control for Threaded Programs
    - Three Scopes of Influence
    - Synchronous vs. Asynchronous
    - Thread-specific Breakpoints

Debugging Parallel Programs [3/11]

- Viewing and Modifying Thread Data
  - Laminated Variables
    - In a parallel program, the same variable will usualy have multiple instances across threads and/or processes
    - Laminating a variable means to display all occurrences simultaneously in a Variable Window
    - Laminated variables can include scalars, arrays, structures and pointers
    - Variable Window > View Menu > Show Across > Thread

Debugging Parallel Programs [4/11]

- Debugging OpenMP Codes
  - Thread-based
  - Setting the number of threads
    - Default: usually equal to the number of cpus on the machine
    - OMP_NUM_THREADS environment variable at run time
    - OMP_SET_NUM_THREADS routine within the source code
  - Code transformation
  - Master thread vs. Worker threads

Debugging Parallel Programs [5/11]

- Debugging OpenMP Codes
  - Execution Control
    - You can not step into or out of a PARALLEL region
    - Set a breakpoint within the parallel region and allow the process to run to it
  - As with threaded codes, TotalView supports laminated variable displays for OpenMP
  - Manager Threads

Debugging Parallel Programs [6/11]

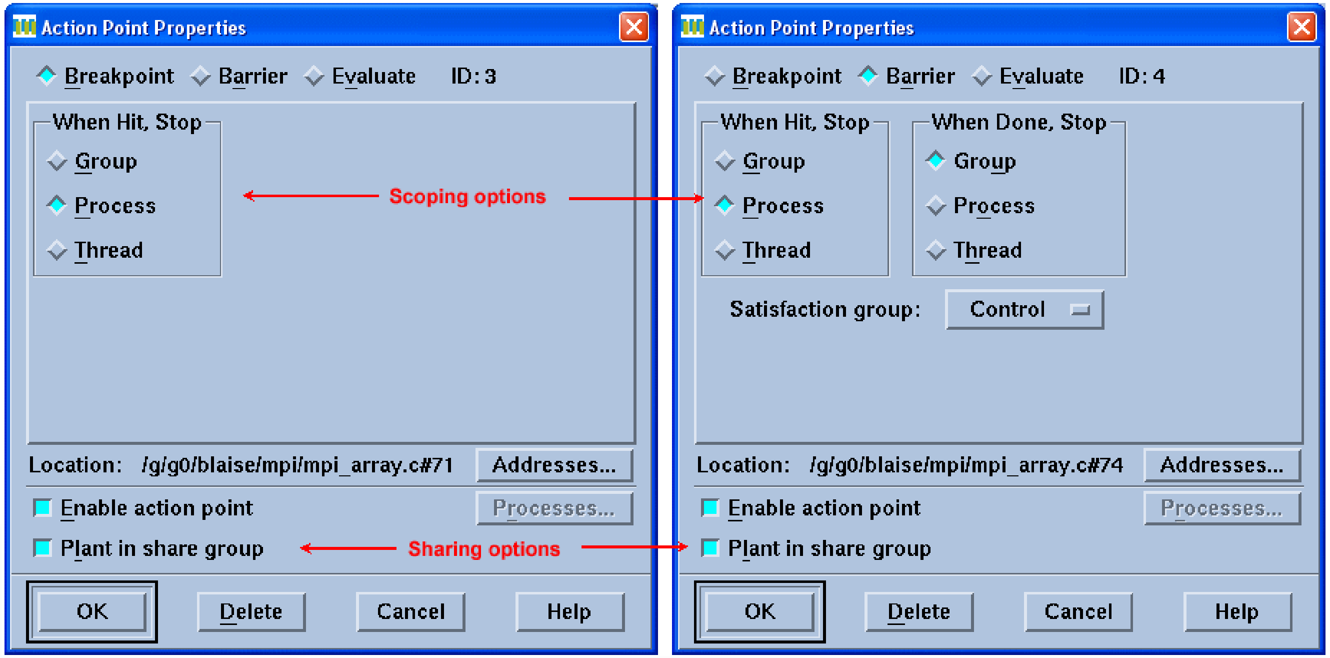
- Debugging MPI Codes
  - Multi-Process
  - MPI manager process
    - Typically, MPI programs run under a "manager" process, such as poe, srun, prun, mpirun, dmpirun, etc.
  - Automatic process acquisition

Debugging Parallel Programs [7/11]

- MPI features similar to OpenMP
  - Selecting an MPI Process
    - Process Navigation Buttons
  - Controlling MPI Process Execution
    - MPI task execution can be controlled at the individual process level, or collectively as a "group"
- Starting and Stopping Processes
- Holding and Releasing Processes

Debugging Parallel Programs [8/11]

- Breakpoints and Barrier Points
  - – Individual breakpoint and barrier point behavior can be customized via the Action Point Properties Dialog Box



Action Point Properties

● Breakpoint ○ Barrier ○ Evaluate ID: 3

When Hit, Stop
○ Group
● Process
○ Thread

Location: /g/g0/blaise/mpi/mpi_array.c#71

☑ Enable action point

☑ Plant in share group

Action Point Properties

○ Breakpoint ● Barrier ○ Evaluate ID: 4

When Hit, Stop
○ Group
● Process
○ Thread

When Done, Stop
● Group
○ Process
○ Thread

Satisfaction group: Control

Location: /g/g0/blaise/mpi/mpi_array.c#74

☑ Enable action point

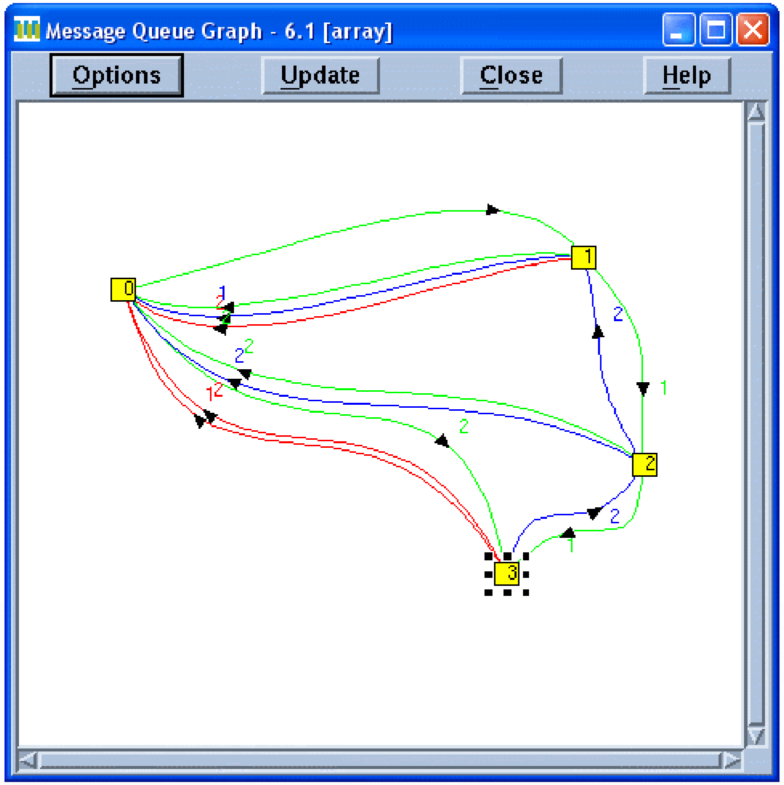
☑ Plant in share group

Debugging Parallel Programs [9/11]

- Displaying Message Queue State
  - Process Window > Tools Menu > Message Queue
  - The Message Queue Window
  - Types of Messages Displayed:
    - Pending receives - non-blocking and blocking.
    - Pending sends - non-blocking and blocking.
    - Unexpected messages - messages sent to this process which do not yet have a matching receive operation.

Debugging Parallel Programs [10/11]

- Message Queue Graph
  - Process Window > Tools Menu > Message Queue Graph



Debugging Parallel Programs [11/11]

- Debugging Hybrid Codes
  - – Hybrid codes are programs that use more than one type of parallelism
  - – Combines technics used in threaded, OpenMP and MPI debugging
- Attaching to a Running Batch Job
  - – If you have a batch job that is already running, you can start TotalView on one of the cluster's login nodes and then attach to it

Examples

- OpenMP example
  - Specify number of threads
  - Set breakpoint inside parallel region
  - Display a variable's value across all threads
- MPI example
  - Start TotalView using mpirun and executable
  - Set a barrier point
  - Display variables across processes

Topics not covered

- CLI
- Setting up remote debugging sessions
- Memory debugging
- Replay engine
- and more...

References and More Information

- The most useful documentation and reference material is from TotalView's vendor site: http://www.roguewave.com/
- Online tutorial: /https://computing.llnl.gov/tutorials/totalview/

WWW.SCL.RS

SCIENTIFIC COMPUTING LABORATORY