TotalView

Setting Up MPI Programs

version 8.6

TotalView Technologies

Contents

1 Setting Up MPI Debugging Sessions

chapter 1

Setting Up MPI Debugging Sessions

This chapter explains how to set up TotalView MPI debugging sessions.

If you are using TotalView Individual, all of your MPI processes must execute on the computer on which you installed TotalView. In addition, TotalView Individual limits you to no more than 16 processes and threads.

This chapter describes the following MPI systems:

This chapter also describes debugger features that you can use with most parallel models:

- If you're using a messaging system, TotalView displays this information visually as a message queue graph and textually in a Message Queue Window. For more information, see "*Displaying the Message Queue Graph Window*" on page 10 and "*Displaying the Message Queue*" on page 13.
- TotalView lets you decide which process you want it to attach to. See "*Attaching to Processes*" on page 27.
- See "*Debugging Parallel Applications Tips*" on page 27 for hints on how to approach debugging parallel programs.

Debugging MPI Programs

Starting MPI Programs

MPI programs use a starter program such as mpirun to start your program. TotalView lets you start these MPI programs in two ways. One requires that the starter program be under TotalView control, and the other does not. In the first case, you will enter the name of the starter program on the command the line. In the other, you will enter program information into the **File > New Program** or **Process > Startup Parameter** dialog boxes.

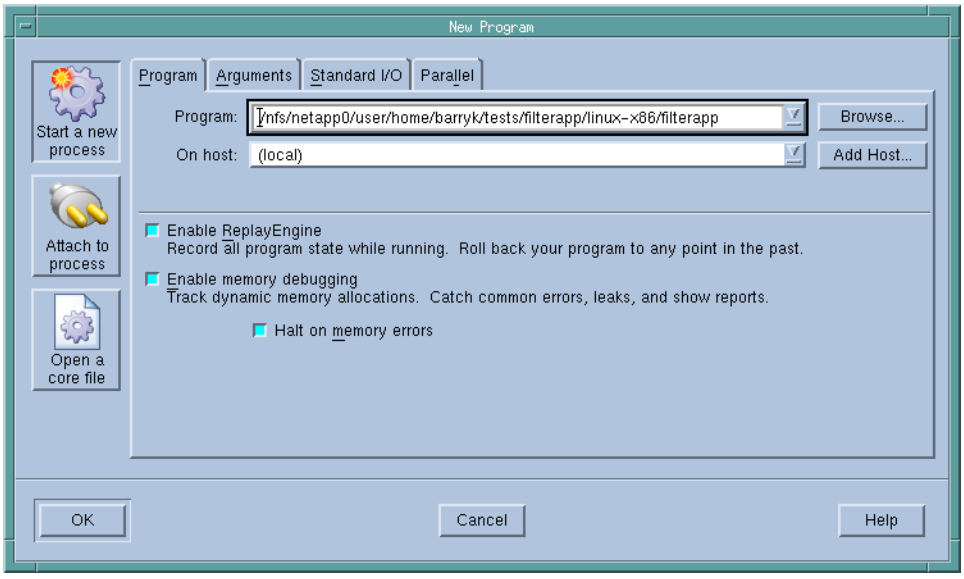
Programs started using these dialog boxes do not use the information you set for single-process and bulk server launching. In addition, you cannot use the Attach Subset command when entering information using these dialog boxes.

Starting MPI programs using the dialog boxes is the recommended method. This method is described in the next section. Starting using a started program is described in various places throughout this chapter.

Starting MPI Programs Using File > New Program

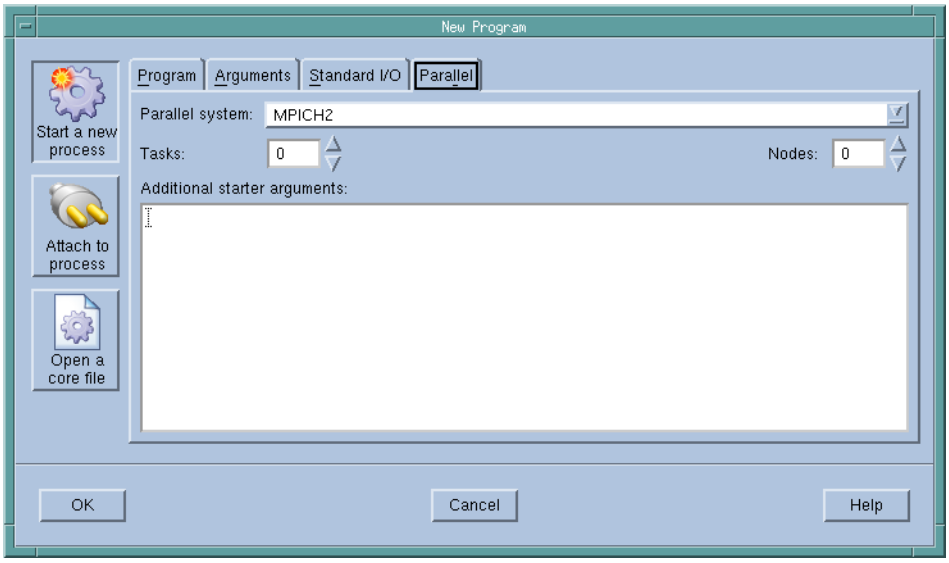
In many cases, the way in which you invoke an MPI program within TotalView control differs little from discipline to discipline. If you invoke TotalView from the command line without an argument, TotalView displays its **File > New Program** dialog box. (See Figure 1 on page 2.)

Figure 1: File > New Program Dialog Box



After entering program's name (**Start a new process** should be selected by default), select the Parallel tab. (See Figure 2 on page 3.)

Figure 2: File > New Program Dialog Box: Parallel Tab



You can now select the **Parallel** system, the number of **Tasks**, and **Nodes**. If there are additional arguments that need to the sent to the starter process, type them within the **Additional starter arguments** area. These arguments are ones that are sent to a starter process such as mpirun or poe. They are not arguments sent to your program.

If you need to add and initialize environment variables and command-line options, select the Arguments tab and enter them.

In most cases, TotalView will remember what you type between invocations of TotalView. For example, suppose you were debugging a program called **my_foo** and set it up using these controls. The next time you start TotalView, you can use the following command:

```
totalview my_foo
```

TotalView will remember what you entered so there is no need to respecify these options.

Debugging MPICH Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2.

To debug Message Passing Interface/Chameleon Standard (MPICH) applications, you must use MPICH version 1.2.3 or later on a homogenous collection of computers. If you need a copy of MPICH, you can obtain it at no cost from Argonne National Laboratory at **www.mcs.anl.gov/mpi**. (We strongly urge that you use a later version of MPICH. The *TotalView Platforms and Systems Requirements* document has information on versions that work with TotalView.)

The MPICH library should use the **ch_p4**, **ch_p4mpd**, **ch_shmem**, **ch_lfshmem**, or **ch_mpl** devices.

- For networks of workstations, the default MPICH library is **ch_p4**.
- For shared-memory SMP computers, use **ch_shmem**.
- On an IBM SP computer, use the **ch_mpl** device.

The MPICH source distribution includes all of these devices. Choose the device that best fits your environment when you configure and build MPICH.

When configuring MPICH, *you must ensure that the* MPICH *library maintains all of the information that TotalView requires. This means that you must use the* –**enable-debug** *option with the* MPICH **configure** *command. (Versions earlier than* 1.2 *used the* **--debug** *option.) In addition, the TotalView Release Notes contains information on patching your* MPICH *version* 1.2.3 *distribution.*

For more information, see:

- *"Starting TotalView on an* MPICH *Job"* on page 4
- *"Attaching to an* MPICH *Job"* on page 5
- *"Using* MPICH P4 *procgroup Files"* on page 7

Starting TotalView on an MPICH Job

Before you can bring an MPICH job under TotalView's control, both TotalView and the **tvdsvr** must be in your path. You can do this in a login or shell startup script.

For version 1.1.2, the following command-line syntax starts a job under TotalView control:

mpirun [MPICH-*arguments*] –**tv** *program* [*program-arguments*]

For example:

```
mpirun -np 4 -tv sendrecv
```

The MPICH **mpirun** command obtains information from the **TOTALVIEW** environment variable and then uses this information when it starts the first process in the parallel job.

For Version 1.2.4, the syntax changes to the following:

mpirun –dbg=totalview [*other_mpich-args*] *program* [*program-args*]

For example:

```
mpirun -dbg=totalview -np 4 sendrecv
```

In this case, **mpirun** obtains the information it needs from the **–dbg** command-line option.

In other contexts, setting this environment variable means that you can use TotalView different versions or pass command-line options to TotalView.

For example, the following is the C shell command that sets the **TOTALVIEW** environment variable so that **mpirun** passes the **–no_stop_all** option to TotalView:

```
setenv TOTALVIEW "totalview -no_stop_all"
```

TotalView begins by starting the first process of your job, the master process, under its control. You can then set breakpoints and begin debugging your code.

On the IBM SP computer with the **ch_mpl** device, the **mpirun** command uses the **poe** command to start an MPI job. While you still must use the MPICH **mpirun** (and its **–tv** option) command to start an MPICH job, the way you start MPICH differs. For details on using TotalView with **poe**, see "*Starting TotalView on a* PE *Program*" on page 20.

Starting TotalView using the **ch_p4mpd** device is similar to starting TotalView using **poe** on an IBM computer or other methods you might use on Sun and HP platforms. In general, you start TotalView using the **totalview** command, with the following syntax;

totalview mpirun [*totalview_args*] **–a** [*mpich-args*] *program* [*program-args*]

CLI: **totalviewcli mpirun** [*totalview_args*] \
–a [*mpich-args*] *program* [*program-args*]

As your program executes, TotalView automatically acquires the processes that are part of your parallel job as your program creates them. Before TotalView begins to acquire them, it asks if you want to stop the spawned processes. If you click **Yes**, you can stop processes as they are initialized. This lets you check their states or set breakpoints that are unique to the process. TotalView automatically copies breakpoints from the master process to the slave processes as it acquires them. Consequently, you don't have to stop them just to set these breakpoints.

If you're using the GUI, TotalView updates the Root Window to show these newly acquired processes. For more information, see "A*ttaching to* Processes" on page 27.

Attaching to an MPICH Job

TotalView lets you to attach to an MPICH application even if it was not started under TotalView control.

To attach to an MPICH application:

1 Start TotalView.

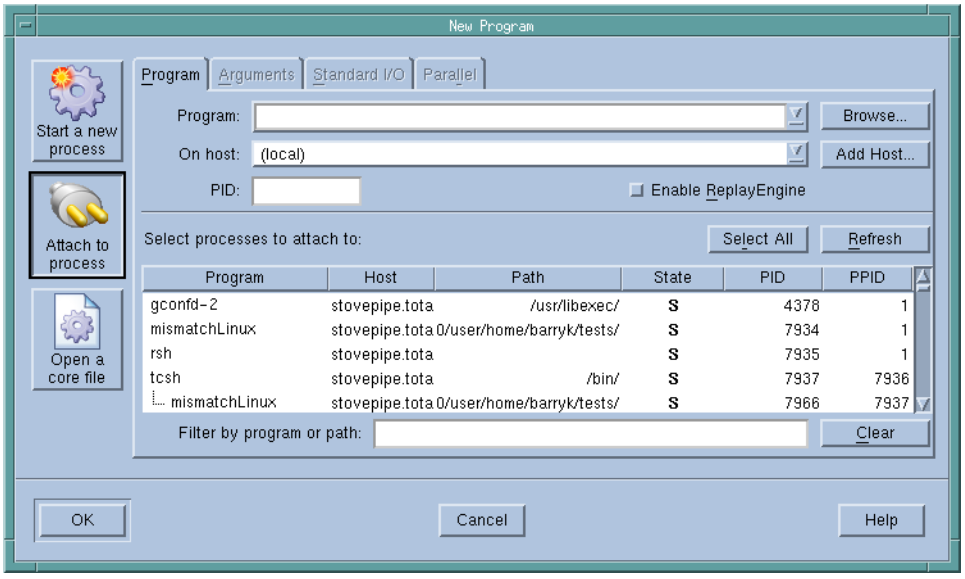
Select **Attach to an existing process** from within the **File > New Program** dialog box. TotalView updates the dialog box so that it displays the processes that are not yet owned.

2 Attach to the first MPICH process in your workstation cluster by diving into it.

CLI: **dattach** *executable pid*

3 On an IBM SP with the **ch_mpi** device, attach to the **poe** process that started your job. For details, see "*Starting TotalView on a* PE *Program*" on page 20. The following figureFigure 3 shows this information.

Figure 3: File > New Program: Attach to an Existing Process



Normally, the first MPICH process is the highest process with the correct program name in the process list. Other instances of the same executable can be:

- The **p4** listener processes if MPICH was configured with **ch_p4**.
- Additional slave processes if MPICH was configured with **ch_shmem** or **ch_lfshmem**.
- Additional slave processes if MPICH was configured with **ch_p4** and has a file that places multiple processes on the same computer.

4 After you attach to your program's processes, TotalView asks if you also want to attach to slave MPICH processes. If you do, press **Return** or choose **Yes**. If you do not, choose **No**.

If you choose **Yes**, TotalView starts the server processes and acquires all MPICH processes.

As an alternative, you can use the **Group > Attach Subset** command to predefine what TotalView should do. For more information, see "*Attaching to Processes*" on page 27.

If you are using TotalView Individual, all of your MPI *processes must execute on the computer on which you installed TotalView.*

In some situations, the processes you expect to see might not exist (for example, they may crash or exit). TotalView acquires all the processes it can and then warns you if it can not attach to some of them. If you attempt to dive into a process that no longer exists (for example, using a message queue display), TotalView tells you that the process no longer exists.

Using MPICH P4 procgroup Files

If you're using MPICH with a P4 **procgroup** file (by using the **–p4pg** option), you must use the *same* absolute path name in your **procgroup** file and on the **mpirun** command line. For example, if your **procgroup** file contains a different path name than that used in the **mpirun** command, even though this name resolves to the same executable, TotalView assumes that it is a different executable, which causes debugging problems.

The following example uses the same absolute path name on the TotalView command line and in the **procgroup** file:

```
% cat p4group
local 1 /users/smith/mympichexe
bigiron 2 /users/smith/mympichexe
% mpirun -p4pg p4group -tv /users/smith/mympichexe
```

In this example, TotalView does the following:

1. Reads the symbols from **mympichexe** only once.
2. Places MPICH processes in the same TotalView share group.
3. Names the processes **mypichexe.0**, **mympichexe.1**, **mympichexe.2**, and **mympichexe.3**.

If TotalView assigns names such as **mympichexe<mympichexe>.0**, a problem occurred and you need to compare the contents of your **procgroup** file and **mpirun** command line.

Debugging MPICH2 Applications

You should be using MPICH2 *version* 1.0.5p4 *or higher. Earlier versions had problems that prevented TotalView from attaching to all the processes or viewing message queue data.*

Downloading and Configuring MPICH2

You can download the current current MPICH2 version from:

http://www-unix.mcs.anl.gov/mpi/mpich/

If you wish to use all of the TotalView MPI features, you must configure MPICH2. Do this by adding the following to the **configure** script that is within the downloaded information:

- -enable-debuginfo - -enable-totalview

The **configure** script looks for the following file:

python2.x/config/Makefile

It fails if the file is not there.

The next steps are:

1. Run **make**
2. Run **make install**
   This places the binaries and libraries in the directory specified by the optional **- -prefix** option.
3. Set the PATH and LD_LIBRARY_PATH to point to the MPICH2 **bin** and **lib** directories.

Starting the mpd Daemon

You must start the **mpd** daemon using the **mpdboot** command. For example:

```
mpdboot -n 4 -f hostfile
```

where:

–n 4	Indicates the number of hosts upon which you wish to run the daemon. In this example, the daemen runs of four housrs
–f hostfile	Is a list of the hosts upon which the application will run. In this example, a file named **hostfile** contains this list.

You are now ready to start debugging your application.

TotalView only supports jobs run using MPD. *Using other daemons such as* SMPD *is not supported*.

Starting TotalView Debugging on an MPICH2 Job

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI *Programs" on page* 2.

TotalView lets you start an MPICH2 job in one of the following ways:

mpiexec *mpi-args* **–tv** *program* **–a** *program-args*
This command tells MPI to start TotalView. You will need to set the TOTALVIEW environment variable to where TotalView is located in your file system when you start a program using **mpiexec**. For example:

```
setenv TOTALVIEW \
     /opt/totalview/bin/totalview
```

This method of starting TotalView does not let you restart your program without exiting TotalView and you will not be able to attach to a running MPI job.

**totalview python -a `which mpiexec` **
–tvsu *mpiexec-args program program-args*

This command lets you restart your MPICH2 job. It also lets you attach to a running MPICH2 job by using the **Attach to process** options within the **File > New Program** dialog box. You need to be careful that you attach to the right instance of python as it is likely that a few instances are running. The one to which you want to attach has no attached children—child processes are indented with a line showing the connection to the parent.

You may not see sources to your program at first. If you do see the program, you can set breakpoints. In either case, press the **Go** button. Your process will start and TotalView displays a dialog box when your program goes paralell that allows you to stop execution. (This is the default behavior. You can change it using the options within **File >Preferences >Parallel** page.

You will also need to set the TOTALVIEW environment variable as indicated in the previous method.

Starting MPI Issues

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2.

If you can't successfully start TotalView on MPI programs, check the following:

- Can you successfully start MPICH programs without TotalView?
  The MPICH code contains some useful scripts that let you verify that you can start remote processes on all of the computers in your computers file. (See **tstmachines** in **mpich/util**.)
- You won't get a message queue display if you get the following warning:
  `The symbols and types in the MPICH library used by TotalView to extract the message queues are not as expected in the image <your image name>. This is probably an MPICH version or configuration problem.`
  You need to check that you are using MPICH Version 1.1.0 or later and that you have configured it with the **–debug** option. (You can check this by looking in the **config.status** file at the root of the MPICH directory tree.)
- Does the TotalView Server (**tvdsvr**) fail to start?

tvdsvr must be in your **PATH** when you log in. Remember that TotalView uses **rsh** to start the server, and that this command doesn't pass your current environment to remotely started processes.

- Make sure you have the correct MPI version and have applied all required patches. See the *TotalView Release Notes* for up-to-date information.
- Under some circumstances, MPICH kills TotalView with the **SIGINT** signal. You can see this behavior when you use the **Group > Kill** command as the first step in restarting an MPICH job.

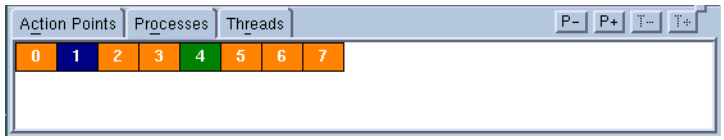
CLI: dfocus g ddelete

If TotalView exits and terminates abnormally with a **Killed** message, try setting the **TV::ignore_control_c** variable to true.

MPI Rank Display

The Processes/Ranks Tab at the bottom of the Process Window contains a grid that displays the status of each rank. For example, in Figure 4 on page 10the following figure, six ranks are at a breakpoint, one is running, and one is stopped.

Figure 4: Ranks Tab



Displaying the Message Queue Graph Window

TotalView can graphically display your MPI program's message queue state. If you select the Process Window **Tools > Message Queue Graph** command, TotalView displays a window that shows a graph of the current message queue state. (See Figure 5.)

If you want to restrict what TotalView displays, you can select the **Options** button. This is shown in Figure 6 on page 11the following figure.

Figure 5: Tools > Message Queue Graph Window

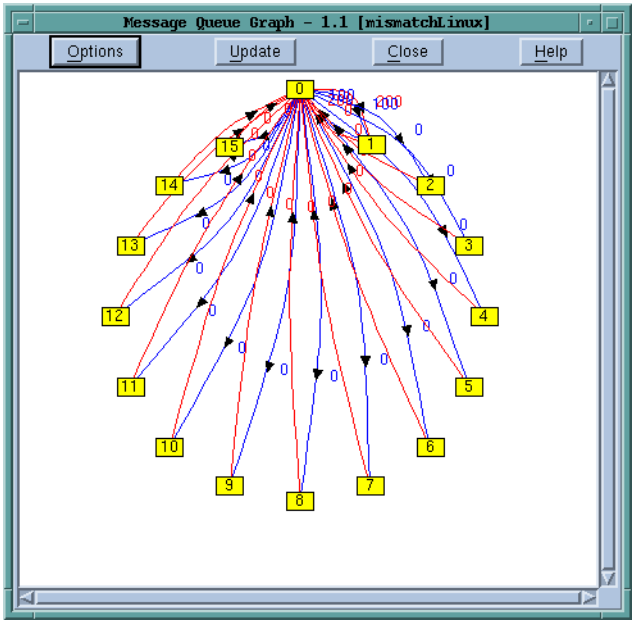
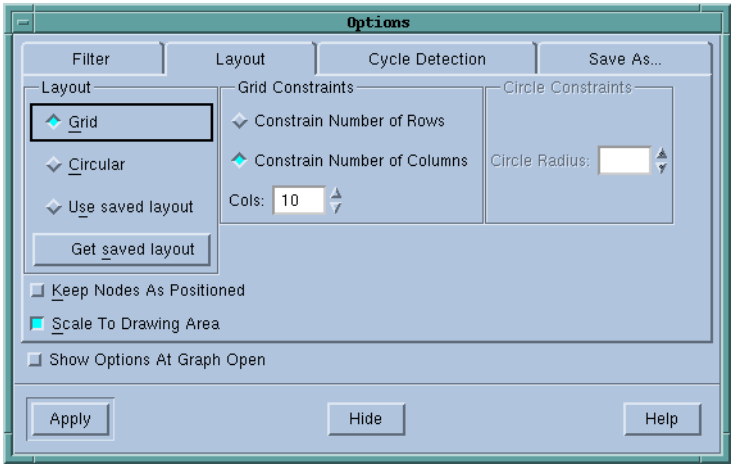


Figure 6: Tools > Message Queue Graph Options Window



Using commands and controls within this window, you can either alter the way in which TotalView displays ranks within this window—for example, as a grid or in a circle.

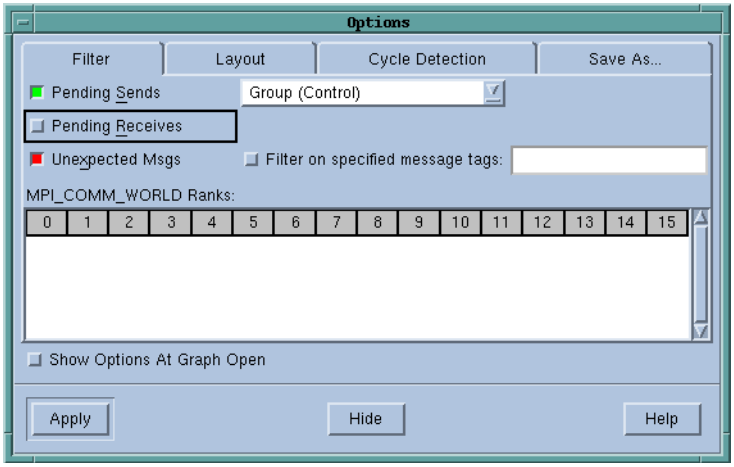
Using the commands within the **Cycle Detection** tab tells TotalView to lets you know about cycles in your messages. This is a quick and efficient way to detect when messages are blocking one another and causing deadlocks.

Perhaps the most used of these tabs is **Filter**. (See Figure 7 on page 12.)

The button colors used for selecting messages are the same as those used to draw the lines and arrows in the **Message Queue Graph** Window, as follows:

- Green: Pending Sends
- Blue: Pending Receives
- Red: Unexpected Messages

Figure 7: Tools > Message Queue Graph Options. Filter Tab



You can directly select which ranks you want displayed in the lower part of the window. The **Filter on specified message tags** area lets you name which tags should be used as filters. Finally, you can select a group or a communicator in the group pulldown, If you have created your own communicators and groups, they will appear here.

Changes made within the **Options** dialog box do not occur until after you click the **Apply** button. The graph window will then change to reflect your changes.

The message queue graph shows your program's state at a particular instant. Selecting the **Update** button tells TotalView to fetch new information and redraw the graph.

The numbers in the boxes within the **Message Queue Graph** Window indicate the MPI message source or destination process rank. Diving on a box tells TotalView to open a Process Window for that process.

The numbers next to the arrows indicate the MPI message tags that existed when TotalView created the graph. Diving on an arrow tells TotalView to display its **Tools > Message Queue** Window, which has detailed information about the messages. If TotalView has not attached to a process, it displays this information in a grey box.

You can use the **Message Queue Graph** Window in many ways, including the following:

- Pending messages often indicate that a process can't keep up with the amount of work it is expected to perform. These messages indicate places where you may be able to improve your program's efficiency.
- Unexpected messages can indicate that something is wrong with your program because the receiving process doesn't know how to process the message. The red lines indicate unexpected messages.
- After a while, the shape of the graph tends to tell you something about how your program is executing. If something doesn't look right, you might want to determine why.

- You can change the shape of the graph by dragging nodes or arrows. This is often useful when you're comparing sets of nodes and their messages with one another. Ordinarily, TotalView doesn't remember the places to which you have dragged the nodes and arrows. This means that if you select the **Update** button after you arrange the graph, your changes are lost. However, if you select **Keep nodes as positioned** from with the **Options** dialog box, updating the window does not change node positioning.

Displaying the Message Queue

The **Tools > Message Queue** Window displays your MPI program's message queue state textually. This can be useful when you need to find out why a deadlock occurred.

The MPI versions for which we display the message queue are described in our platforms guide. This document is contained within the online help and is also available on our web site at http://www.totalviewtech.com/Documentation/

For more information, see:

- *"About the Message Queue Display"* on page 13
- *"Using Message Operations"* on page 13

About the Message Queue Display

After an MPI process returns from the call to **MPI_Init()**, you can display the internal state of the MPI library by selecting the **Tools > Message Queue** command.

This window displays the state of the process's MPI communicators. If user-visible communicators are implemented as two internal communicator structures, TotalView displays both of them. One is used for point-to-point operations and the other is used for collective operations.

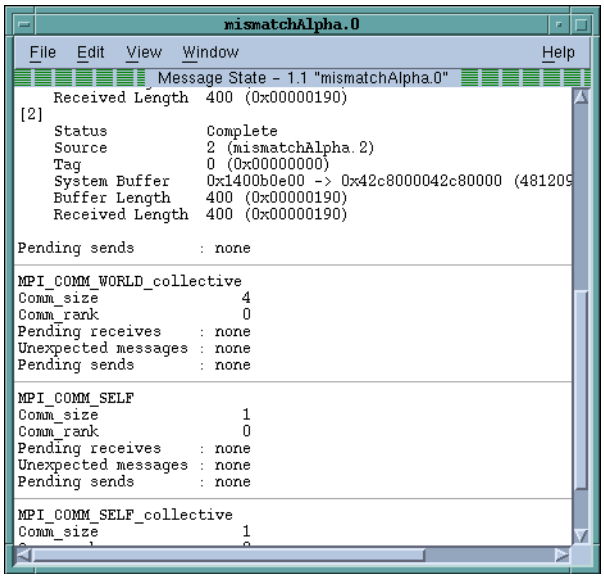
You cannot edit any of the fields in the Message Queue Window.

The contents of the Message Queue Window are only valid when a process is stopped.

Using Message Operations

For each communicator, TotalView displays a list of pending receive operations, pending unexpected messages, and pending send operations. Each

Figure 8: Message Queue Window



operation has an index value displayed in brackets ([*n*]). The online Help for this window contains a description of the fields that you can display.

For more information, see:

- "*Diving on* MPI *Processes*" on page 14
- "*Diving on* MPI *Buffers*" on page 14
- "*About Pending Receive Operations*" on page 15
- "*About Unexpected Messages*" on page 15
- "*About Pending Send Operations*" on page 15

Diving on MPI Processes

To display more detail, you can dive into fields in the Message Queue Window. When you dive into a process field, TotalView does one of the following:

- Raises its Process Window if it exists.
- Sets the focus to an existing Process Window on the requested process.
- Creates a new Process Window for the process if a Process Window doesn't exist.

Diving on MPI Buffers

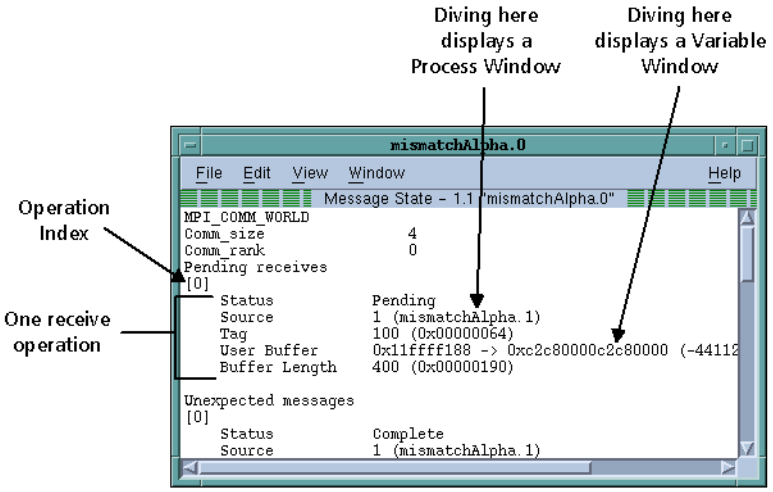
When you dive into the buffer fields, TotalView opens a Variable Window. It also guesses what the correct format for the data should be based on the buffer length and the data alignment. You can edit the **Type** field within the Variable Window, if necessary.

TotalView doesn't use the MPI *data type to set the buffer type.*

About Pending Receive Operations

TotalView displays each pending receive operation in the **Pending receives** list. Figure 9The following figure shows an example of an MPICH pending receive operation.

Figure 9: Message Queue Window Showing Pending Receive Operation



TotalView displays all receive operations maintained by the IBM MPI *library. Set the environment variable* **MP_EUIDEVELOP** *to the value* **DEBUG** *if you want blocking operations to be visible; otherwise, the library only maintains nonblocking operations. For more details on the* **MP_EUIDEVELOP** *environment variable, see the* IBM *Parallel Environment Operations and Use manual.*

About Unexpected Messages

The **Unexpected messages** portion of the **Message Queue** Window shows information for retrieved and enqueued messages that are not yet matched with a receive operation.

Some MPI libraries, such as MPICH, only retrieve messages that have already been received as a side effect of calls to functions such as **MPI_Recv()** or **MPI_Iprobe()**. (In other words, while some versions of MPI may know about the message, the message may not yet be in a queue.) This means that TotalView can't list a message until after the destination process makes a call that retrieves it.

About Pending Send Operations

TotalView displays each pending send operation in the **Pending sends** list.

MPICH does not normally keep information about pending send operations. If you want to see them, start your program under TotalView control and use the **mpirun –ksq** or **–KeepSendQueue** command.

Depending on the device for which MPICH was configured, blocking send operations may or may not be visible. However, if TotalView doesn't display them, you can see that these operations occurred because the call is in the stack backtrace.

If you attach to an MPI program that isn't maintaining send queue information, TotalView displays the following message:

```
Pending sends : no information available
```

Debugging Cray MPI Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2

Specific information on debugging Cray MPI applications is located in our discussion of running TotalView on Cray platforms.

Debugging HP Tru64 Alpha MPI Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2.

To use TotalView with HP Tru64 Alpha MPI applications, you must use HP Tru64 Alpha MPI version 1.7 or later.

Starting TotalView on an HP Alpha MPI Job

In most cases, you start an HP Alpha MPI program by using the **dmpirun** command. The command for starting an MPI program under TotalView control is similar; it uses the following syntax:

{ totalview | totalviewcli } dmpirun –a *dmpirun-command-line*

This command invokes TotalView and tells it to show you the code for the main program in **dmpirun**. Since you're not usually interested in debugging this code, use the **Process > Go** command to let the program run.

CLI: dfocus p dgo

The **dmpirun** command runs and starts all MPI processes. After TotalView acquires them, it asks if you want to stop them.

Problems can occur if you rerun HP Alpha MPI *programs that are under TotalView control because resource allocation issues exist within* HP Alpha MPI. *The* HP Alpha MPI *documentation contains information on using* **mpiclean** *to clean up the* MPI *system state*.

Attaching to an HP Alpha MPI Job

To attach to a running HP Alpha MPI job, attach to the **dmpirun** process that started the job. The procedure for attaching to a **dmpirun** process is the same as the procedure for attaching to other processes. You can also use the **Group > Attach Subset** command which is discussed in *"Attaching to Processes"* on page 27.

After you attach to the **dmpirun** process, TotalView asks if you also want to attach to slave MPICH processes. If you do, press **Return** or choose **Yes**. If you do not, choose **No**.

If you choose **Yes**, TotalView starts the server processes and acquires all MPICH processes.

Debugging HP MPI Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI *Programs" on page* 2.

You can debug HP MPI applications on a PA-RISC 1.1 or 2.0 processor. To use TotalView with HP MPI applications, you must use HP MPI versions 1.6 or 1.7.

Starting TotalView on an HP MPI Job

TotalView lets you start an MPI program in one of the following ways:

{ totalview | totalviewcli } *program* **–a** *mpi-arguments*

This command tells TotalView to start the MPI process. TotalView then shows you the machine code for the HP MPI **mpirun** executable.

CLI: dfocus p dgo

mpirun *mpi-arguments* **–tv** *program*

This command tells MPI to start TotalView. You will need to set the TOTALVIEW environment variable to where TotalView is located in your file system when you start a program using **mpirun**. For example:

```
setenv TOTALVIEW \
     /opt/totalview/bin/totalview
```

mpirun *mpi-arguments* **–tv –f** *startup_file*

This command tells MPI to start TotalView and then start the MPI processes as they are defined in the *startup_file* script. This file names the processes that MPI starts. Typically, this file has contents that are similar to:

```
-h aurora –np 8/path/to/program
-h borealis –np 8 /path/to/program1
```

Your HP MPI documentation describes the contents of this startup file. These contents include the remote host name, environment variables, number of processes, programs, and so on. As is described in the previous example, you must set the TOTALVIEW environment variable.

Just before **mpirun** starts your MPI processes, TotalView acquires them and asks if you want to stop the processes before they start executing. If you answer **yes**, TotalView halts them before they enter the **main()** routine. You can then create breakpoints.

Attaching to an HP MPI Job

To attach to a running HP MPI job, attach to the HP MPI **mpirun** process that started the job. The procedure for attaching to an **mpirun** process is the same as the procedure for attaching to any other process.

After TotalView attaches to the HP MPI **mpirun** process, it displays the same dialog box as it does with MPICH. (See step 4 on page 6 of *"Attaching to an* MPICH *Job"* on page 5.)

Debugging IBM MPI Paralle Environment (PE) Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI *Programs" on page* 2.

You can debug IBM MPI Parallel Environment (PE) applications on the IBM RS/6000 and SP platforms.

To take advantage of TotalView's ability to automatically acquire processes, you must be using release 3,1 or later of the Parallel Environment for AIX.

Topics in this section are:

Preparing to Debug a PE Application

The following sections describe what you must do before TotalView can debug a PE application.

Using Switch-Based Communications

If you're using switch-based communications (either IP *over the switch* or *user space*) on an SP computer, you must configure your PE debugging session so that TotalView can use IP *over the switch* for communicating with the TotalView Server (**tvdsvr**). Do this by setting the **–adapter_use** option to **shared** and the **–cpu_use** option to **multiple**, as follows:

- If you're using a PE host file, add **shared multiple** after all host names or pool IDs in the host file.
- Always use the following arguments on the **poe** command line:

```
-adapter_use shared -cpu_use multiple
```

If you don't want to set these arguments on the **poe** command line, set the following environment variables before starting **poe**:

```
setenv MP_ADAPTER_USE shared
setenv MP_CPU_USE multiple
```

When using IP *over the switch*, the default is usually **shared adapter use** and **multiple cpu use**; we recommend that you set them explicitly using one of these techniques. You must run TotalView on an SP or SP2 node. Since TotalView will be using IP *over the switch* in this case, you cannot run TotalView on an RS/6000 workstation.

Performing a Remote Login

You must be able to perform a remote login using the **rsh** command. You also need to enable remote logins by adding the host name of the remote node to the **/etc/hosts.equiv** file or to your **.rhosts** file.

When the program is using switch-based communications, TotalView tries to start the TotalView Server by using the **rsh** command with the switch host name of the node.

Setting Timeouts

If you receive communications timeouts, you can set the value of the MP_TIMEOUT environment variable; for example:

```
setenv MP_TIMEOUT 1200
```

If this variable isn't set, TotalView uses a **timeout** value of 600 seconds.

Starting TotalView on a PE Program

The following is the syntax for running Parallel Environment (PE) programs from the command line:

program [*arguments*] [*pe_arguments*]

You can also use the **poe** command to run programs as follows:

poe *program* [*arguments*] [*pe_arguments*]

If, however, you start TotalView on a PE application, you must start **poe** as TotalView's target using the following syntax:

{ totalview | totalviewcli } poe –a *program* [*arguments*] [PE_*arguments*]

For example:

```
totalview poe -a sendrecv 500 -rmpool 1
```

Setting Breakpoints

After TotalView is running, start the **poe** process using the **Process > Go** command.

CLI: dfocus p dgo

TotalView responds by displaying a dialog box—in the CLI, it prints a question—that asks if you want to stop the parallel tasks.

If you want to set breakpoints in your code before they begin executing, answer **Yes**. TotalView initially stops the parallel tasks, which also allows you to set breakpoints. You can now set breakpoints and control parallel tasks in the same way as any process controlled by TotalView.

If you have already set and saved breakpoints with the **Action Point > Save All** command, and you want to reload the file, answer **No**. After TotalView loads these saved breakpoints, the parallel tasks begin executing.

CLI: dactions –save *filename*
dactions –load *filename*

Starting Parallel Tasks

After you set breakpoints, you can start all of the parallel tasks with the Process Window **Group > Go** command.

CLI: dfocus G dgo
Abbreviation: G

No parallel tasks reach the first line of code in your main routine until all parallel tasks start.

Be very cautious in placing breakpoints at or before a line that calls **MPI_Init()** or **MPL_Init()** because timeouts can occur while your program is

being initialized. After you allow the parallel processes to proceed into the MPI_Init() or MPL_Init() call, allow all of the parallel processes to proceed through it within a short time. For more information on this, see *"Avoid unwanted timeouts"* on page 32.

Attaching to a PE Job

To take full advantage of TotalView's **poe**-specific automation, you need to attach to **poe** itself, and let TotalView automatically acquire the **poe** processes on all of its nodes. In this way, TotalView acquires the processes you want to debug.

Attaching from a Node Running poe

To attach TotalView to **poe** from the node running **poe**:

1. Start TotalView in the directory of the debug target.

   If you can't start TotalView in the debug target directory, you can start TotalView by editing the **tvdsvr** command line before attaching to **poe**.
2. In the **File > New Program** dialog box, select **Attach to an existing process**, then find the **poe** process list, and attach to it by diving into it. When necessary, TotalView launches **tvdsvr**s. TotalView also updates the Root Window and opens a Process Window for the **poe** process.

   CLI: dattach poe *pid*
3. Locate the process you want to debug and dive on it. TotalView responds by opening a Process Window for it. If your source code files are not displayed in the Source Pane, you might not have told TotalView where these files reside. You can fix this by invoking the **File > Search Path** command to add directories to your search path.

Attaching from a Node Not Running poe

The procedure for attaching TotalView to **poe** from a node that is not running **poe** is essentially the same as the procedure for attaching from a node that is running **poe**. Since you did not run TotalView from the node running **poe** (the startup node), you won't be able to see **poe** on the process list in the Root Window and you won't be able to start it by diving into it.

To place **poe** in this list:

1. Connect TotalView to the startup node.
2. Select the **File > New Program** dialog box, and select **Attach to an existing process**.
3. Look for the process named **poe** and continue as if attaching from a node that is running **poe**.

CLI: dattach -r *hostname* poe *poe-pid*

Debugging IBM Blue Gene Applications

While the way in which you debug IBM Blue Gene MPI programs is identical to the way in which you debug these programs on other platforms, starting TotalView on your program differs slightly. Unfortunately, each machine is configured differently so you'll need to find information in IBM's documentation or in documentation created at your site.

Nevertheless, the remainder of this section will present some hints.

In general, you will either launch **mpirun** under debugger control or start TotalView and attach to an already running **mpirun**. For example:

{ totalview | totalviewcli } mpirun –a *mpirun-command-line*

TotalView tells **mpirun** to launch TotalView Debug Servers on each Blue Gene I/O nodes.

Because I/O nodes cannot resolve network names, TotalView must pass the address of the front-end node interface to the servers on the I/O nodes. This is usually not the same interface that is generally used to connect to the front-end node. TotalView assumes that the address can be resolved by using a name that is:

front-end-hostname-**io**.

For example, if the hostname of the front-end is **fred**, the servers will connect to **fred-io**.

The systems at the IBM *Blue Gene Capacity on Demand follow this convention. If you are executing programs there, you will not need to set the TotalView variables described in the rest of this section.*

If the front-end cannot resolve this name, you must supply the name of the interface using the **–local_interface** command-line option or by setting the **bluegene_io_interface** TotalView variable. (This variable is described in the Chapter 4 of the *TotalView Reference Guide*.)

Because the same version of TotalView must be able to debug both Power-Linux programs (for example, **mpirun**) and Blue Gene programs, TotalView uses a Blue Gene-specific server launch string. You can define this launch string by setting the **bluegene_server_launch_string** TotalView variable or command-line option.

You must set this variable in a tvdrc file. This differs from other TotalView launch strings, which you can set using the File > Preferences Dialog Box.

The default value for the **bluegene_server_launch_string** variable is:

```
–callback %L –set_pw %P -verbosity %V %F
```

In this string, **%L** is the address of the front-end node interface used by the servers. The other substitution arguments have the same meaning as they do in a normal server launch string. These substitution arguments are discussed in Chapter 7 of the *TotalView Reference Guide*.

Debugging LAM/MPI Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI *Programs" on page* 2.

The following is a description of the LAM/MPI implementation of the MPI standard. Here are the first two paragraphs of Chapter 2 of the "LAM/MPI *User's Guide*". You can find You can obtain this document by going to the LAM documentation page, which is: **http://www.lam-mpi.org/using/docs/**.

> "LAM/MPI is a high-performance, freely available, open source implementation of the MPI standard that is researched, developed, and maintained at the Open Systems Lab at Indiana University. LAM/MPI supports all of the MPI-1 Standard and much of the MPI-2 standard. More information about LAM/MPI, including all the source code and documentation, is available from the main LAM/MPI web site.
>
> "LAM/MPI is not only a library that implements the mandated MPI API, but also the LAM run-time environment: a user-level, daemon-based run-time environment that provides many of the services required by MPI programs. Both major components of the LAM/MPI package are designed as component frameworks—extensible with small modules that are selectable (and configurable) at run-time. ...

You debug a LAM/MPI program in a similar way to how you debug most MPI programs. Use the following syntax if TotalView is in your path:

```
mpirun –tv args prog prog_args
```

As an alternative, you can invoke TotalView on **mpirun**:

```
totalview mpirun –a prog prog_args
```

The LAM/MPI *User's Guide* discusses how to use TotalView to debug LAM/MPI programs.

Debugging QSW RMS Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2.

TotalView supports automataic process acquistion on AlphaServer SC systems and 32-bit Red Hat Linux systems that use Quadrics RMS resource management system with the QSW switch technology.

Message queue display is only supported if you are running version 1, patch 2 or later, of AlphaServer SC.

Starting TotalView on an RMS Job

To start a parallel job under TotalView control, use TotalView as if you were debugging **prun**:

{ totalview | totalviewcli } prun –a *prun-command-line*

TotalView starts and shows you the machine code for RMS **prun**. Since you're not usually interested in debugging this code, use the **Process > Go** command to let the program run.

CLI: dfocus p dgo

The RMS **prun** command executes and starts all MPI processes. After TotalView acquires them, it asks if you want to stop them at startup. If you answer **yes**, TotalView halts them before they enter the main program. You can then create breakpoints.

Attaching to an RMS Job

To attach to a running RMS job, attach to the RMS **prun** process that started the job. You attach to the **prun** process the same way you attach to other processes.

After you attach to the RMS **prun** process, TotalView asks if you also want to attach to slave MPICH processes. If you do, press **Return** or choose **Yes**. If you do not, choose **No**.

If you choose **Yes**, TotalView starts the server processes and acquires all MPI processes.

As an alternative, you can use the **Group > Attach Subset** command to predefine what TotalView should do. For more information, see *"Attaching to Processes"* on page 27.

Debugging SiCortex MPI Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2

Specific information on debugging SiCortex MPI applications is located in our discussion of running TotalView on SiCortex platforms.

Debugging SGI MPI Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2.

TotalView can acquire processes started by SGI MPI applications. This MPI is part of the Message Passing Toolkit (MPT) 1.3 and 1.4 packages. TotalView can display the Message Queue Graph Window for these releases. See "*Displaying the Message Queue Graph Window*" on page 10 for message queue display.

Starting TotalView on an SGI MPI Job

You normally start SGI MPI programs by using the **mpirun** command. You use a similar command to start an MPI program under debugger control, as follows:

{ totalview | totalviewcli } mpirun –**a** *mpirun-command-line*

This invokes TotalView and tells it to show you the machine code for **mpirun**. Since you're not usually interested in debugging this code, use the **Process > Go** command to let the program run.

CLI: dfocus p dgo

The SGI MPI **mpirun** command runs and starts all MPI processes. After TotalView acquires them, it asks if you want to stop them at startup. If you answer **Yes**, TotalView halts them before they enter the main program. You can then create breakpoints.

If you set a verbosity level that allows informational messages, TotalView also prints a message that shows the name of the array and the value of the array services handle (**ash**) to which it is attaching.

Attaching to an SGI MPI Job

To attach to a running SGI MPI program, attach to the SGI MPI **mpirun** process that started the program. The procedure for attaching to an **mpirun** process is the same as the procedure for attaching to any other process.

After you attach to the **mpirun** process, TotalView asks if you also want to attach to slave MPICH processes. If you do, press **Return** or choose **Yes**. If you do not, choose **No**.

If you choose **Yes**, TotalView starts the server processes and acquires all MPICH processes.

As an alternative, you can use the **Group > Attach Subset** command to predefine what TotalView will do. For more information, see *"Attaching to Processes"* on page 27.

Debugging Sun MPI Applications

In many cases, you can bypass the procedure described in this section. For more information, see "Debugging MPI Programs" on page 2.

TotalView can debug a Sun MPI program and can display Sun MPI message queues. This section describes how to perform *job startup* and *job attach* operations.

To start a Sun MPI application, use the following procedure.

1 Type the following command:

totalview mprun [*totalview_args*] **–a** [*mpi_args*]

For example:

```
totalview mprun –g blue –a –np 4 /usr/bin/mpi/conn.x
```

CLI: **totalviewcli mprun** [*totalview_args*] **–a** [*mpi_args*]

When the TotalView Process Window appears, select the **Go** button.

CLI: **dfocus p dgo**

TotalView may display a dialog box with the following text:

```
Process mprun is a parallel job. Do you want to stop
the job now?
```

2 If you compiled using the **–g** option, click **Yes** to tell TotalView to open a Process Window that shows your source. All processes are halted.

Attaching to a Sun MPI Job

To attach to an already running **mprun** job:

1 Find the host name and process identifier (PID) of the **mprun** job by typing **mpps –b**. For more information, see the **mpps(1M)** manual page.

The following is sample output from this command:

```
JOBNAME      MPRUN_PID    MPRUN_HOST
cre.99       12345        hpc-u2-9
cre.100      12601        hpc-u2-8
```

2 After selecting **File > New Program**, type **mprun** in the **Executable** field and type the PID in the **Process ID** field.

CLI: dattach *mprun mprun-pid*
For example:
dattach mprun 12601

3 If TotalView is running on a different node than the **mprun** job, enter the host name in the **Remote Host** field.

CLI: dattach –r *host-name* mprun *mprun-pid*

Debugging Parallel Applications Tips

This section contains information about debugging parallel programs:

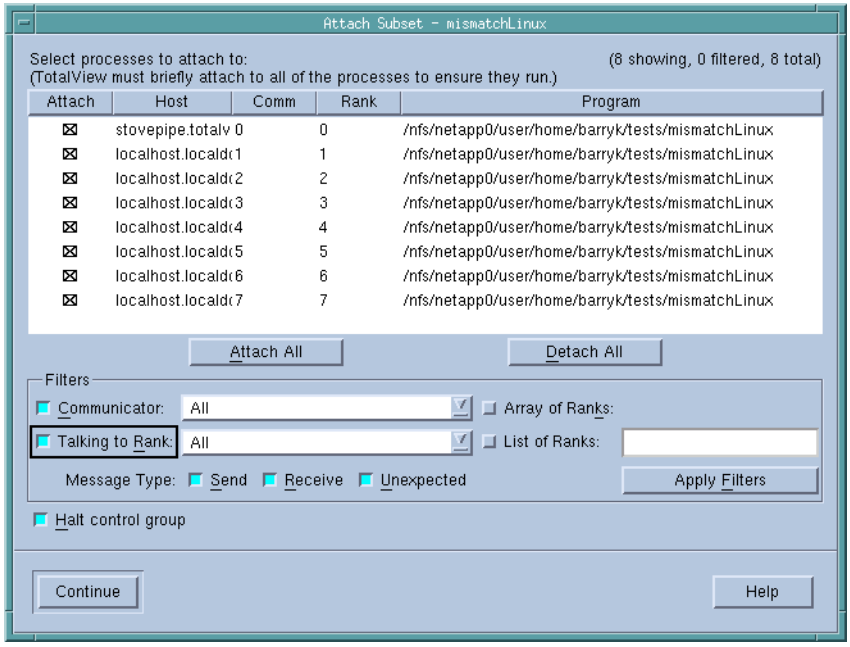
Attaching to Processes

In a typical multi-process job, you're interested in what's occurring in some of your processes and not as much interested in others. By default, TotalView tries to attach to all of the processes that your program starts. If there are a lot of processes, there can be considerable overhead involved in opening and communicating with the jobs.

You can minimize this overhead by using the **Group > Attach Subset** command, which displays the dialog box shown in the folliwng figureFigure 10.

TotalView lets you start MPI *jobs in two ways. One requires that the starter program be under TotalView control and have special instrumentation for TotalView while the other does not. In the first case, you will enter the name of the starter program on the com-*

Figure 10: Group > Attach Subset Dialog Box



mand line. The other requires that you enter information into the File > New Program or Process > Startup Parameters dialog boxes. The Attach Subset command is only available if you directly name a starter program on the command line.

Selecting boxes on the left side of the list tells TotalView which processes it should attach to. Although your program will launch all of these processes, TotalView only attaches to the processes that you have selected.

The controls under the **All** and the **None** buttons let you limit which processes TotalView automatically attaches to, as follows:

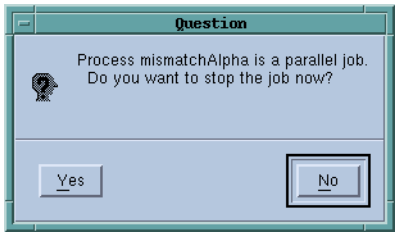
- The **Communicator** control specifies that the processes must be involved with the communicators that you select. For example, if something goes wrong that involves a communicator, selecting it from the list tells TotalView to only attach to the processes that use that communicator.
- The **Talking to Rank** control further limits the processes to those that you name here. Most of the entries in this list are just the process numbers. In most cases, you would select **All** or **MPI_ANY_SOURCE**.
- The three checkboxes in the **Message Type** area add yet another qualifier. Checking a box tells TotalView to only display communicators that are involved with a **Send**, **Receive**, or **Unexpected** message.

After you find the problem, you can detach from these nodes by selecting **None**. In most cases, use the **All** button to select all the check boxes, then clear the ones that you're not interested in.

Many applications place values that indicate the rank in a variable so that the program can refer to them as they are needed. If you do this, you can display the variable in a Variable Window and then select the **Tools > Attach Subset (Array of Ranks)** command to display this dialog box.

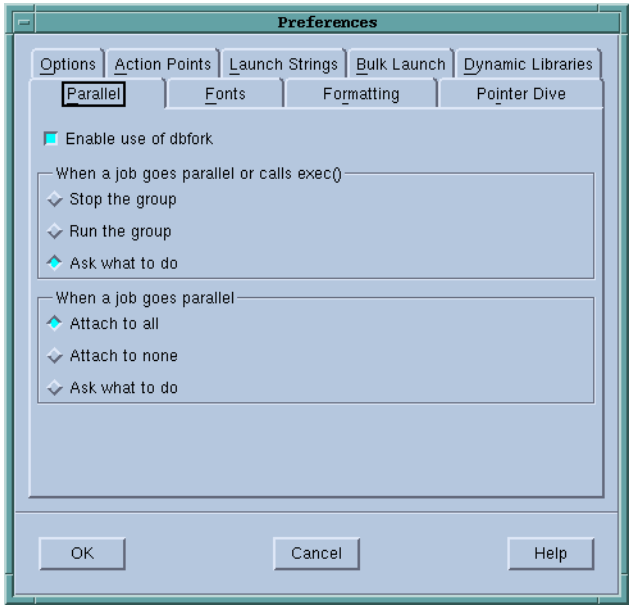
You can use the **Group > Attach Subset** command at any time, but you would probably use it immediately before TotalView launches processes. Unless you have set preferences otherwise, TotalView stops and asks if you want it to stop your processes. When selected, the **Halt control group** check box also tells TotalView to stop a process just before it begins executing. (See Figure 11 on page 29.)

Figure 11: Stop Before Going Parallel Question Box



The commands on the Parallel Page in the **File > Preferences** Dialog Box let you control what TotalView does when your program goes parallel. (See Figure 12.)

Figure 12: File > Preferences: Parallel Page



TotalView only displays the preceding question box when you directly name a starter program on the command line.

The radio button in the **When a job goes parallel or calls exec()** area lets TotalView:

- **Stop the group**: Stop the control group immediately after the processes are created.
- **Run the group**: Allow all newly created processes in the control group to run freely.

- **Ask what to do**: Ask what should occur. If you select this option, TotalView asks if it should start the created processes.

CLI: dset TV::parallel_stop

The radio buttons in the **When a job goes parallel** area let TotalView:

- **Attach to all**: Automatically attach to all processes when they begin executing.
- **Attach to none**: Does not attach to any created process when it begins executing.
- **Ask what to do**: Asks what should occur. If you select this option, TotalView opens the same dialog box that is displayed when you select **Group > Attach Subset**. TotalView then attaches to the processes that you have selected. This dialog box isn't displayed when you set the preference. Instead, it controls what happens when your program creates parallel processes.

CLI: dset TV::parallel_attach

Parallel Debugging Tips

The following tips are useful for debugging most parallel programs:

- **Setting Breakpoint behavior**

  When you're debugging message-passing and other multi-process programs, it is usually easier to understand the program's behavior if you change the default stopping action of breakpoints and barrier breakpoints. By default, when one process in a multi-process program hits a breakpoint, TotalView stops all the other processes.

  To change the default stopping action of breakpoints and barrier breakpoints, you can set debugger preferences. The online Help contains information on these preference. These preferences tell TotalView whether to continue to run when a process or thread hits the breakpoint.

  These options only affect the default behavior. You can choose a behavior for a breakpoint by setting the breakpoint properties in the **File > Preferences** Action Points Page.

- **Synchronizing Processes**

  TotalView has two features that make it easier to get all of the processes in a multi-process program synchronized and executing a line of code. Process barrier breakpoints and the process hold/release features work together to help you control the execution of your processes.

  The Process Window **Group > Run To** command is a special stepping command. It lets you run a group of processes to a selected source line or instruction.

- **Using group commands**

  Group commands are often more useful than process commands.

It is often more useful to use the **Group** > **Go** command to restart the whole application instead of the **Process** > **Go** command.

CLI: dfocus g dgo
Abbreviation: G

You would then use the **Group** > **Halt** command instead of **Process** > **Halt** to stop execution.

CLI: dfocus g dhalt
Abbreviation: H

The group-level single-stepping commands such as **Group** > **Step** and **Group** > **Next** let you single-step a group of processes in a parallel.

CLI: dfocus g dstep
Abbreviation: S

dfocus g dnext
Abbreviation: N

- **Stepping at Process-level**

  If you use a process-level single-stepping command in a multi-process program, TotalView may appear to hang (it continuously displays the watch cursor). If you single-step a process over a statement that can't complete without allowing another process to run, and that process is stopped, the stepping process appears to hang. This can occur, for example, when you try to single-step a process over a communication operation that cannot complete without the participation of another process. When this happens, you can abort the single-step operation by selecting **Cancel** in the **Waiting for Command to Complete** Window that TotalView displays. As an alternative, consider using a group-level single-step command.

CLI: Type Ctrl+C

TotalView Technologies receives many bug reports about processes being hung. In almost all cases, the reason is that one process is waiting for another. Using the **Group** *debugging commands almost always solves this problem.*

- **Determining which processes and threads are executing**

  The Root Window helps you determine where various processes and threads are executing. When you select a line of code in the Process Window, the Root Window updates to show which processes and threads are executing that line.

- **Viewing variable values**

  You can view the value of a variable that is replicated across multiple processes or multiple threads in a single Variable Window.

- **Restarting from within TotalView**

  You can restart a parallel program at any time. If your program runs past the point you want to examine, you can kill the program by selecting the

Group > Kill command. This command kills the master process and all the slave processes. Restarting the master process (for example, **mpirun** or **poe**) recreates all of the slave processes. Start up is faster when you do this because TotalView doesn't need to reread the symbol tables or restart its **tvdsvr** processes, since they are already running.

CLI: dfocus g dkill

MPICH Debugging Tips

The following debugging tips apply only to MPICH:

- **Passing options to mpirun**

  You can pass options to TotalView using the MPICH **mpirun** command.

  To pass options to TotalView when running **mpirun**, you can use the **TOTALVIEW** environment variable. For example, you can cause **mpirun** to invoke TotalView with the **–no_stop_all** option, as in the following C shell example:

  ```
  setenv TOTALVIEW "totalview -no_stop_all"
  ```

- **Using ch_p4**

  If you start remote processes with MPICH/**ch_p4**, you may need to change the way TotalView starts its servers.

  By default, TotalView uses **rsh** to start its remote server processes. This is the same behavior as **ch_p4** uses. If you configure **ch_p4** to use a different start-up mechanism from another process, you probably also need to change the way that TotalView starts the servers.

IBM PE Debugging Tips

The following debugging tips apply only to IBM MPI (PE):

- **Avoid unwanted timeouts**

  Timeouts can occur if you place breakpoints that stop other processes too soon after calling **MPI_Init()** or **MPL_Init()**. If you create "stop all" breakpoints, the first process that gets to the breakpoint stops all the other parallel processes that have not yet arrived at the breakpoint. This can cause a timeout.

  To turn the option off, select the Process Window **Action Point > Properties** command while the line with the stop symbol is selected. After the **Properties** Dialog Box appears, select the **Process** button in the **When Hit, Stop** area, and also select the **Plant in share group** button.

  CLI: dbarrier *location* –stop_when_hit process

- **Control the poe process**

  Even though the **poe** process continues under debugger control, do not attempt to start, stop, or otherwise interact with it. Your parallel tasks require that **poe** continues to run. For this reason, if **poe** is stopped, TotalView automatically continues it when you continue any parallel task.

- **Avoid slow processes due to node saturation**

If you try to debug a PE program in which more than three parallel tasks run on a single node, the parallel tasks on each node can run noticeably slower than they would run if you were not debugging them.

In general, the number of processes running on a node should be the same as the number of processors in the node.

This becomes more noticeable as the number of tasks increases, and, in some cases, the parallel tasks does not progress. This is because PE uses the SIGALRM signal to implement communications operations, and AIX requires that debuggers must intercept all signals. As the number of parallel tasks on a node increases, TotalView becomes saturated and can't keep up with the SIGALRM signals being sent, thus slowing the tasks.

MPI Startup

chapter 2

Overview

TotalView Technologies products know about different MPI implementations. Because so many implementations are standard, our products usually do the right thing. Unfortunately, subtle differences in your environment or an implementation can cause difficulties that prevent our products from automatically starting your program. In these cases, you must declare what needs to be done.

The following explanation is for TotalView and MemoryScape.

The only way MemoryScape users can alter the way an MPI program stars up is by altering the **parallel_support.tvd** file, which is contained within the **totalview/lib** installation directory area. TotalView users can also alter this file and they can create a local definition.

If you are using a locally-installed MPI implementation, you should add it to your PATH variable. By default, our products use the information in PATH to find the parallel launcher (for example, **mpirun**, **mpiexec**, **poe**, **srun**, **prun**, **dmpirun**, and so on). Generally, if you can run your parallel job from a command line, TotalView and MemoryScape can alo run it.

If you have multiple installed MPI systems—for example, multiple versions of MPICH installed on a common file server—only one can be in your path. In this case, you would need to specify an absolute path to launch it, which means you will need to customize the **TV::parallel_configs** list variable or the **parallel_support.tvd** file contained within your installation directory so that it does not rely on your PATH variable.

The easiest way to create your own startup configuration is to copy a similar configuration from the **TV::private::parallel_configs_base** variable to the **TV::parallel_configs** variable, then make changes.

When you add configurations, they are simply added to a list. This means that if TotalView Technologies supplies a definition named **foo** and you create a definition named **foo**, both exist and your product chooses the first one in the list. Because both are displayed, you must be careful to give each new definition a unique names.

Customizing Your Parallel Configuration

The **File > New** dialog box (TotalView) or the **Add parallel program** screen (MemoryScape) lets you select a parallel configuration. If the default configurations that TotalView Technologies provides do not meet your needs, you can either overwrite these configurations or create new ones by setting the **TV::parallel_configs** variable. Here are three examples:

```
dset TV::parallel_configs {
        #Argonne MPICH
        name:           MPICH;
        description:    Argonne MPICH;
        starter:        mpirun -tv -ksq %s %p %a;
        style:          setup_script;
        tasks_option:   -np;
        nodes_option:   -nodes;
        env_style:      force;
        pretest:        mpichversion;

        #Argonne MPICH2
        name:           MPICH2;
        description:    Argonne MPICH2;
        starter:        $mpiexec -tvsu %s %p %a;
        style:          manager_process;
        tasks_option:   -n;
        env_option:     -env;
        env_style:      assign_space_repeat;
        comm_world:     0x44000000;
        pretest:        mpich2version

        # AIX POE
        name:            poe - AIX;
        description:     IBM PE - AIX;
        tasks_option:    -procs;
        tasks_env:       MP_PROCS;
        nodes_option:    -nodes;
        starter:         /bin/poe %p %a %s;
        style:           bootstrap;
        env:          NLSPATH=/usr/lib/nls/msg/%L/%N/: \
                        /usr/lib/nls/msg/%L/%N.cat;
        service_tids:   2 3 4;
```

```
    comm_world:     0;
    pretest:        test -x /bin/poe
    msq_lib:        /usr/lpp/ppe.poe/lib/%m
}
```

All lines (except for comments) end with a semi-colon (;). Add spaces freely to make what you enter readable as TotalView and MemoryScape ignore them.

Notice that the MPICH2 definition contains the **$mpiexec** variable. This variable is defined elsewhere in the **parallel_support.tvd** file as follows:

```
set mpiexec mpiexec;
```

There is no limit to how many definitions you can place within the **parallel_support.tvd** file or within a variable. The definitions you create will appear in the **Parallel** system pulldown list in the **File > New** dialog box (TotalView) or the **Add parallel program** screen (MemoryScape) and can be used an argument to the **–mpi** option of the CLI's **dload** command.

When running TotalView, you can set this variable in two places:

- Your system's **.tvdrc** file. If you set this variable here, everyone using this TotalView version will see the definition.
- Your **.totalview/tvdrc** file. You will be the only person to see this definition when you start TotalView.

The fields that you can set are as follows:

comm_world — Only use this option when **style** is set to **bootstrap**. This variable is the definition of MPI_COMM_WORLD in C and C++. MPI_COMM_WORLD is usually a **#define** or **enum** to a special number or a pointer value. If you do not include this field, TotalView and MemoryScape cannot acquire the rank for each MPI process.

description — (optional) A string describing what the configuration is used for. There is no length limit.

env — (optional) Defines environment variables that are placed in the starter program's environment. (Depending on how the starter works, these variables may not make their way into the actual ranked processes.) If you are defining more than one environment variable, define each in its own **env** clause.

The format to use is:

```
variable_name=value
```

env_option — (optional) Names the command-line option that exports environment variables to the tasks started by the launcher program. Use this option along with the **env_style** field.

env_style — (optional) Contains a list of environment variables that are passed to tasks.

assign: The argument to be inserted to the command-line option named in **env_option** is a comma-separated list of environment variable **name=value** pairs; that is,

```
NAME1=VALUE1,NAME2=VALUE2,NAME3=VALUE3
```

This option is ignored if you do not use an **env_option** clause.

assign_space_repeat: The argument after **env_option** is a space-separated name/value pair that is assigned to an environment variable. The command within **env_option** is repeated for each environment variable; that is, suppose you enter:

```
-env NAME1 VALUE1 -env NAME2 VALUE2
-env NAME3 VALUE3
```

This mode is primarily used for the **mpiexec**.py MPICH2 starter program.

excenv: ???

export: The argument to be inserted after the command named in **env_option**. This is a comma-separated list of environment variable names; that is,

```
NAME1,NAME2,NAME3
```

This option is ignored if you do not use the **env_option** clause.

force: Environment variables are forced into the ranked processes using a shell script. TotalView or MemoryScape will generate a script that launches the target program. The script also tells the starter to run that script. This clause requires that your home directory be visible on all remote nodes. In most cases, you will use this option when you need to dynamically link memory debugging into the target. While this option does not work with all MPI implementations, it is the most reliable method for MPICH1.

none: No argument is inserted after **env_option**.

msq_lib — (optional) Names the dynamically loaded library that TotalView and MemoryScape use when it needs to locate message queue information. You can name this file using either a relative or full pathname.

name — A short name describing the configuration. This name shows up in such places as the **File > New** dialog box and in the **Process > Startup Parameter**'s Parallel tab in TotalView and the **Add parallel program** screen in MemoryScape. TotalView and MemoryScape remember which configuration you use when starting a program so that they can automatically reapply the configuration when you restart the program.

Because the configuration is associated with a program's name, renaming or moving the program destroys this association.

nodes_option — Names the command-line option (usually **–nodes**) that sets the number of nodes upon which your program runs.

This statement does not define the value that is the argument to this command-line option.

Only omit this statement if your system doesn't allow you to control the number of nodes from the command line. If you set this value to zero ("0"), this statement is omitted.

pretest (optional) Names a shell command that is run before the parallel job is launched. This command must run quickly, produce a timely response, and have no side-effects. This is a test, not a setup hook.

TotalView or MemoryScape may kill the test if it takes too long. It may call it more than once to be sure if everything is OK. If the shell command exit is not as expected, TotalView or Memoryscape complains and asks for permission before continuing,

pretext_exit The expected error code of the pretest command. The default is zero.

service_tids (optional) The list of thread IDs that TotalView and MemoryScape marks as service threads. When using TotalView, you can use the **View > Display Managers** command to tell TotalView to display them.

A service thread differs from a system manager thread in that it is created by the parallel runtime and are not created by your program. POE for example, often creates three service threads.

starter Defines a template that TotalView and MemoryScape use to create the command line that starts your program. In most cases, this template describes the relative position of the arguments. However, you can also use it to add extra parameters, commands, or environment variables. Here are the three substation parameters:

%a: Replaced with the command-line arguments passed to rank processes.

%p: Replaced with the absolute pathname of the target program.

%s: Replaced with additional startup arguments. These are parameters to the starter process, not the rank processes.

For example:

```
starter: mpirun -tv -all-local %s %p %a;
```

When the user selects a value for the option indicated by the **nodes_option** and **tasks_options**, the argument and the value are placed within the **%s** parameter. If you enter a value of 0 for either of these, MemoryScape and TotalView omit the parameter. In MemoryScape, 0 is the default.

style MPI programs are launched in two ways: either by a manager process or by a script. Use this option to name the method, as follows:

manager_process: The parallel system uses a binary manager process to oversee process creation and process lifetime. Our products attach to this process and communicate with it using its debug interface. For example, IBM's poe uses this style.

```
style: manager_process;
```

setup_script: The parallel system uses a script—which is often **mpirun**—to set up the arguments, environment, and temporary files. However, the script does not run as part of the parallel job. This script must understand the **–tv** command-line option and the TOTALVIEW environment variable.

bootstrap: The parallel system attempts to launch an uninstruemented MPI by interposing TotalView or MemoryScape inside the parallel launch sequence in place of the target program. This does not work for MPICH and SGI MPT.

tasks_env The name of an environment variable whose value is the expected number of parallel tasks. This is consulted when the user does not explicitly specify a task count.

tasks_option (sometimes required) Lets you define the option (usually **-np** or **-procs**) that controls the total number of tasks or processes.

Only omit this statement if your system doesn't allow you to control the number of tasks from the command line. If you set this to 0, this statement is omitted.

Index

E

F

G

H

I

K

L

M

N

P

Q

R

S

T

U

V

W