

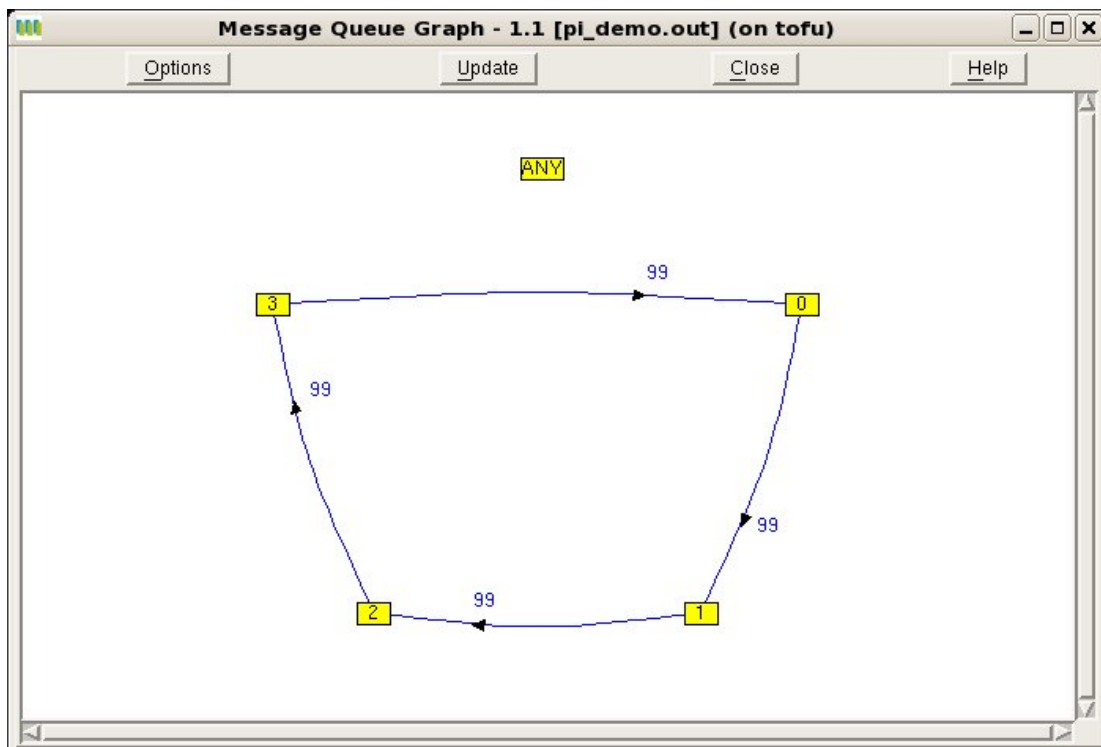
Additional useful features of Totalview Debugger in multi-core environments

Totalview Debugger has a many interesting features that make debugging in a multi-core environment easier. In the main document, we discussed how the memory leaks can be easily detected via using the built-in 'Memory Debugging' feature in Totalview. In this document we will discuss two other useful features, the visualization of the MPI message queues, and data generalization across several processes or threads.

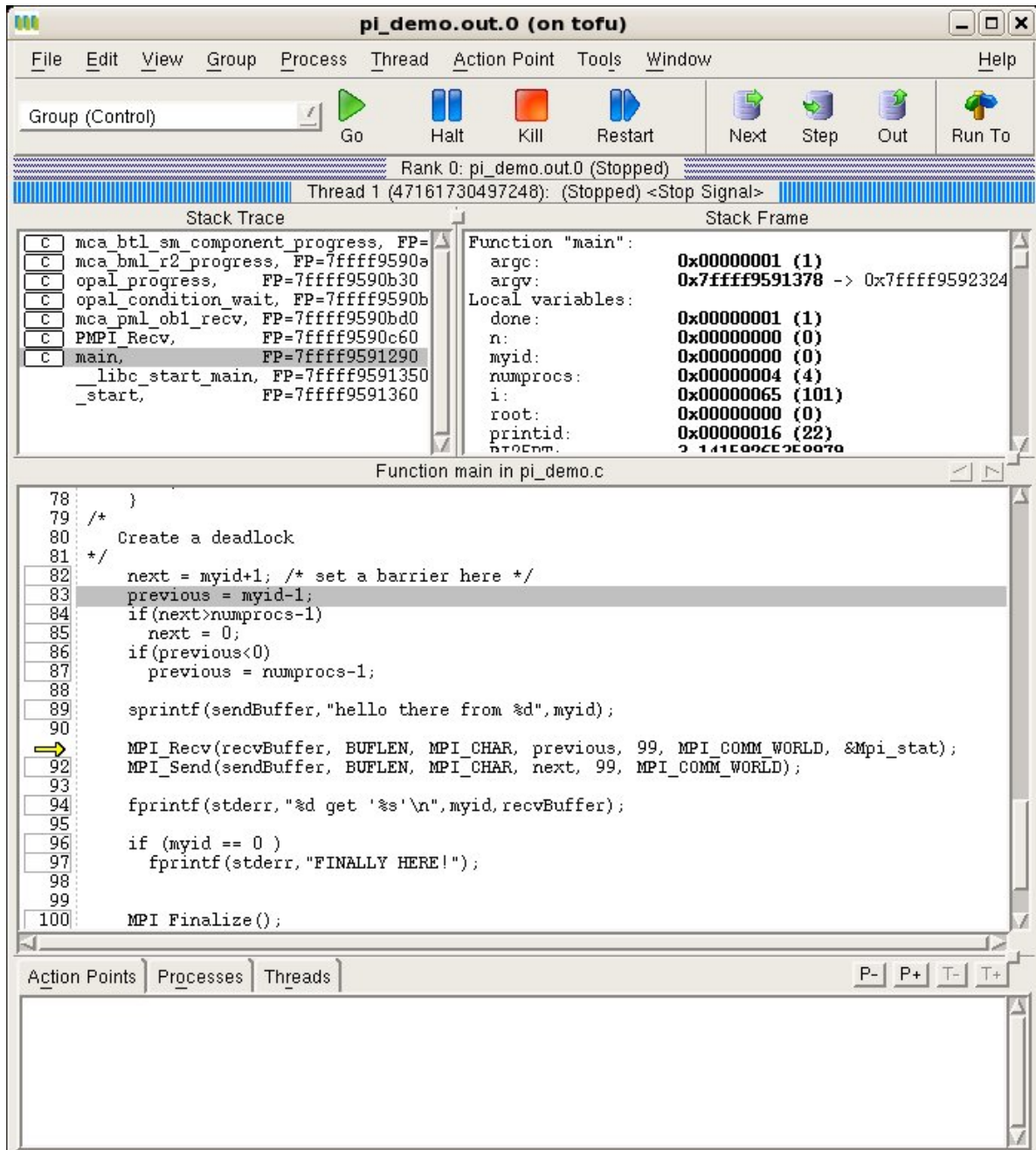
Message Queue Graph

The ability to monitor communication between different processes can help resolve many problems in parallel code, such as identifying deadlocks or performance sinks in the program. The 'Message Queue Graph' feature in Totalview allows the user to see all messages between tasks in a simple graphical form.

As an example of a 'Message Queue Graph' (MSQ Graph) we study the program `pi_demo.c`, that does not finish at all, rather than seems to stall. We launch the program under Totalview by typing `totalview ./pi_demo.out`. After 'Halting' the program, the MSQ Graph in Totalview is then available under the 'Tools' -> 'Message Queue Graph' menu. On the Figure 1 below we show the MSQ Graph for such a four-process job. The blue arrows indicate that all processes are expecting to receive a message, although none of the processes are actually sending any.

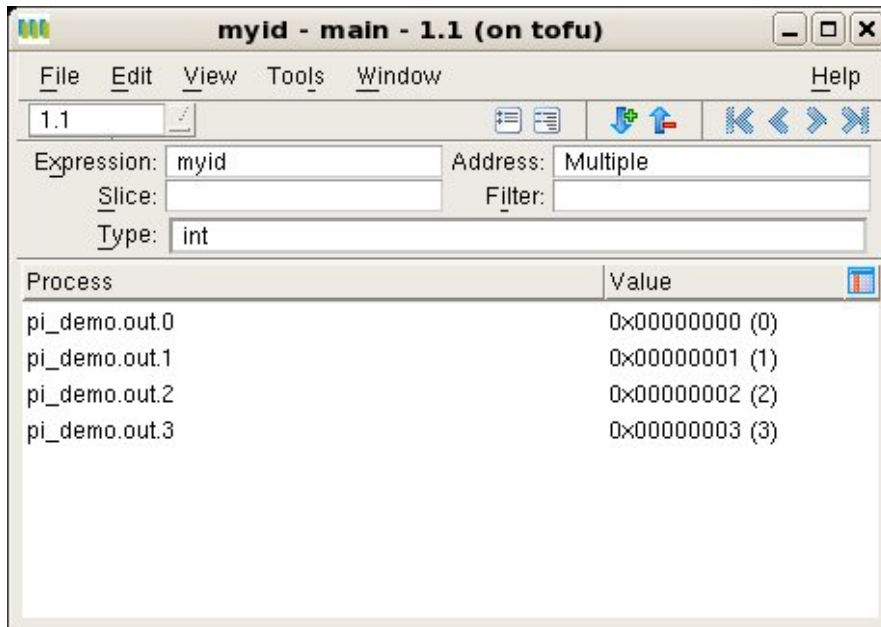


Looking at the source code in the 'Source Pane' window in Totalview (See Figure 2 below), the yellow arrow immediately indicates the current location in the program, and we see that indeed the MPI_Recv command is before the MPI_Send command. The easy fix is then to switch the order of these commands in the source code, recompile, and the program will execute as initially intended. For further details see the appendix to this document where the syntax for compilation and environment variables is discussed.



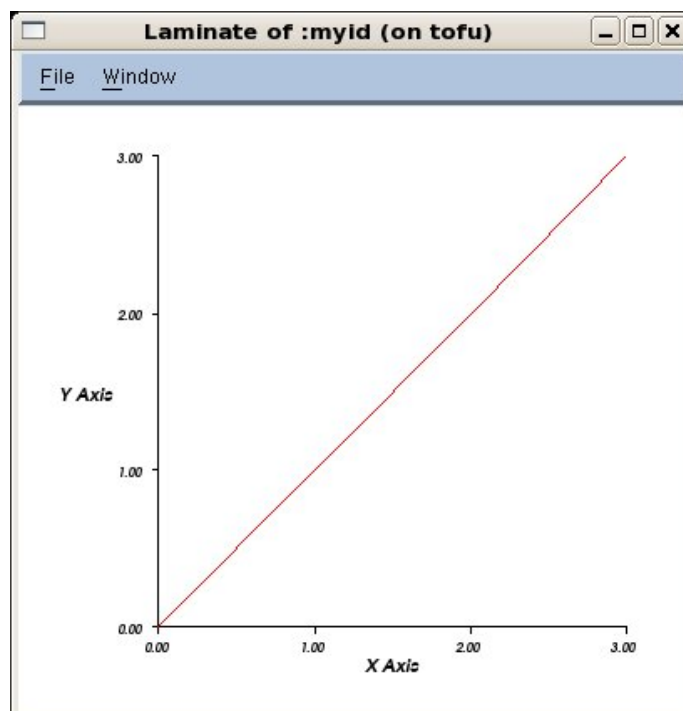
Data analysis across multiple processes and threads

Another prerequisite in a parallel debugging is to effectively analyze data across different processes or threads. Totalview allows the user access and view data across different processes or threads by simply right clicking on a variable in the 'Source Pane' and then choosing 'View Across' processes or threads. Figure 3 below shows the values of 'myid' variable for all processes in a single window for the pi_demo.c program (For this example, we have first set a breakpoint at line #39). In addition to simple numerical representation, Totalview also allows the user to visualize the data by choosing 'Tools' -> 'Visualize' in this window. In Figure 4 below we show the value of 'myid' as a function of process rank numbers, which is linear in this example.



The screenshot shows a TotalView window titled "myid - main - 1.1 (on tofu)". The window has a menu bar with File, Edit, View, Tools, Window, and Help. Below the menu bar is a toolbar with various icons. The main area contains a table with two columns: "Process" and "Value". The table lists four processes: pi_demo.out.0, pi_demo.out.1, pi_demo.out.2, and pi_demo.out.3. The values are 0x00000000 (0), 0x00000001 (1), 0x00000002 (2), and 0x00000003 (3) respectively. Above the table, there are fields for "Expression: myid", "Address: Multiple", "Slice:", "Filter:", and "Type: int".

Process	Value
pi_demo.out.0	0x00000000 (0)
pi_demo.out.1	0x00000001 (1)
pi_demo.out.2	0x00000002 (2)
pi_demo.out.3	0x00000003 (3)



COMPILATION AND LINKING:

In order to view the 'Message Queue Graph', the MPI libraries need to support Message Queues, as is the case with e.g. OpenMPI and MPICH2. The compilation and linking using OpenMPI can be done as follows:

```
> /home/mware/openmpi/openmpi/1.2.4/x86_64-linux/bin/mpicc -MD -ansi -g -c -o pi_demo.o pi_demo.c
```

```
> /home/mware/openmpi/openmpi/1.2.4/x86_64-linux/bin/mpicc -MD -g -o pi_demo.out pi_demo.o
```

```
> setenv PATH /home/mware/openmpi/openmpi/1.2.4/x86_64-linux/bin:$PATH
```

```
> setenv LD_LIBRARY_PATH /home/mware/openmpi/openmpi/1.2.4/x86_64-linux/lib/openmpi:/home/mware/openmpi/openmpi/1.2.4/x86_64-linux/lib:$LD_LIBRARY_PATH
```

```
> totalview ./pi_demo.out
```

In Totalview then choose 'Process' -> 'Startup Parameters' -> 'Parallel' and then 'OpenMPI' as the communication library and 4 processes. The program can then be started by simply clicking 'Go'.

pi_demo.c

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double f( double );
double f( double a )
{
    void* s;
    s = malloc(500);
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i, root, printid;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    int BUFLLEN=512, next, previous;
    char sendBuffer[BUFLLEN];
    char recvBuffer[BUFLLEN];
    MPI_Status Mpi_stat;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n",
            myid, processor_name);

    printid = 22;
    n = 0;
    while (!done)
    {
        if (myid == 0)
        {
            /*
                printf("Enter the number of intervals: (0 quits) ");
                scanf("%d",&n);
            */
            if (n==0) n=100; else n=0;

            startwtime = MPI_Wtime();
        }
        root=0;
        MPI_Bcast(&n, 1, MPI_INT, root, MPI_COMM_WORLD);
        if (n == 0)
```

```

        done = 1;
    else
    {
        h = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs)
        {
            x = h * ((double)i - 0.5);
            sum += f(x);
        }
        mypi = h * sum;

        root=0;
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, root,
MPI_COMM_WORLD);

        if (myid == printid)
        {
            printf("pi is approximately %.16f, Error is %.16f\n",
                pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("wall clock time = %f\n",
                endwtime-startwtime);
        }
    }
}

/*
Create a deadlock
*/
next = myid+1; /* set a barrier here */
previous = myid-1;
if(next>numprocs-1)
    next = 0;
if(previous<0)
    previous = numprocs-1;

sprintf(sendBuffer,"hello there from %d",myid);

MPI_Recv(recvBuffer, BUFLen, MPI_CHAR, previous, 99, MPI_COMM_WORLD,
&Mpi_stat);
MPI_Send(sendBuffer, BUFLen, MPI_CHAR, next, 99, MPI_COMM_WORLD);

fprintf(stderr,"%d get '%s'\n",myid,recvBuffer);

if (myid == 0 )
    fprintf(stderr,"FINALLY HERE!");

MPI_Finalize();

return 0;
}

```