RogueWave by Perforce

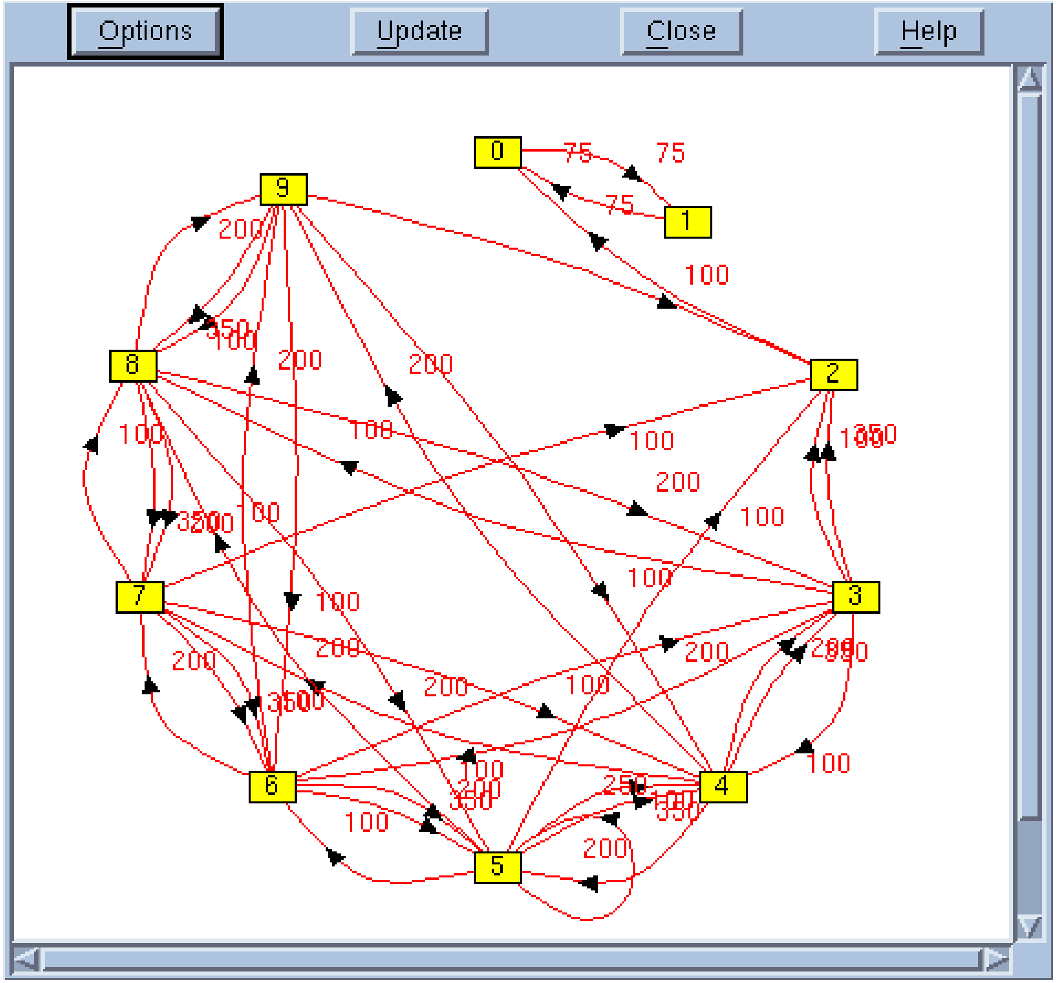
# Techniques for Debugging HPC Applications

NIKOLAY PISKUN , DIRECTOR OF CONTINUING ENGINEERING, TOTALVIEW PRODUCTS

AUGUST 7 2019, ATRESC 2019

# Agenda

- What is debugging and why TotalView?
- Overview of TotalView  and TotalView's new UI
- Advanced C++ and Data debugging
- MPI and OpenMP parallel debugging
- Reverse debugging
- Memory debugging
- GPU debugging
- Python/C++  debugging
- Using TotalView on ANL
- TotalView resources and documentation
- Questions/Comments



# What is Debugging  and Why do you need TotalView?

# What is Debugging?

- Debugging is the process of finding and resolving defects or problems within a computer program or a system.
  - Algorithm correctness
  - Data correctness
  - Scaling/Porting correctness

# TotalView debugger enables you to do:

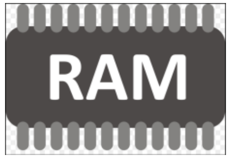
- **Interactive debugging**
  - Live control of an executing program

- **Remote debugging**

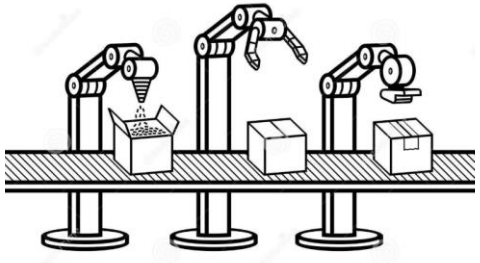
  - Debug a program running on another computer

- **Post-mortem debugging (core files and reverse debugging)**
  - Debugging a program after it has crashed or exited

- **Memory debugging**



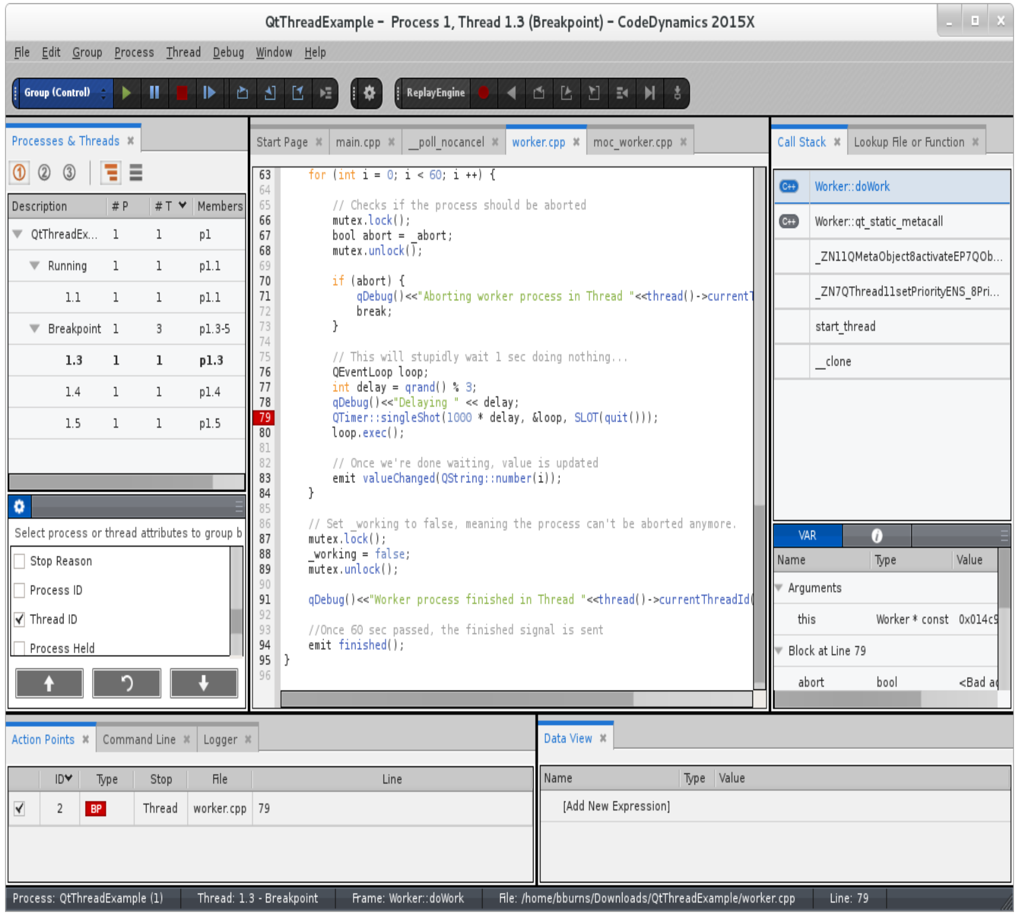
  - Find memory management problems (leaks, corruption …)
  - Comparing results between executions



- **Batch debugging (tvscript, CI environments)**
  - Unattended debugging

# TotalView for HPC and for All

- Leading debug environment for HPC users
  - Active development for 30+ years
  - Thread specific breakpoints
  - Control individual thread execution
  - View complex data types easily
  - From **MacBook** to **Top500** Supercomputers
- Track memory leaks in running applications
- Supports C/C++ and Fortran on Linux/Unix/Mac
- Support debugging mixed Python/C++
- Integrated Reverse debugging
- Batch non-interactive debugging.



| Description | # P | # T | Members |
|---|---|---|---|
| QtThreadEx... | 1 | 1 | p1 |
| Running | 1 | 1 | p1.1 |
| 1.1 | 1 | 1 | p1.1 |
| Breakpoint | 1 | 3 | p1.3-5 |
| **1.3** | **1** | **1** | **p1.3** |
| 1.4 | 1 | 1 | p1.4 |
| 1.5 | 1 | 1 | p1.5 |

Select process or thread attributes to group b

- ☐ Stop Reason
- ☐ Process ID
- ☑ Thread ID
- ☐ Process Held

```
    for (int i = 0; i < 60; i ++) {

        // Checks if the process should be aborted
        mutex.lock();
        bool abort = _abort;
        mutex.unlock();

        if (abort) {
            qDebug()<<"Aborting worker process in Thread "<<thread()->currentT
            break;
        }

        // This will stupidly wait 1 sec doing nothing...
        QEventLoop loop;
        int delay = qrand() % 3;
        qDebug()<<"Delaying " << delay;
        QTimer::singleShot(1000 * delay, &loop, SLOT(quit()));
        loop.exec();

        // Once we're done waiting, value is updated
        emit valueChanged(QString::number(i));
    }

    // Set _working to false, meaning the process can't be aborted anymore.
    mutex.lock();
    _working = false;
    mutex.unlock();

    qDebug()<<"Worker process finished in Thread "<<thread()->currentThreadId(

    //Once 60 sec passed, the finished signal is sent
    emit finished();
}
```

| | |
|---|---|
| C++ | Worker::doWork |
| C++ | Worker::qt_static_metacall |
| | _ZN11QMetaObject8activateEP7QOb... |
| | _ZN7QThread11setPriorityENS_8Pri... |
| | start_thread |
| | __clone |

| Name | Type | Value |
|---|---|---|
| Arguments | | |
| this | Worker * const | 0x014c9 |
| Block at Line 79 | | |
| abort | bool | <Bad a |

| | ID | Type | Stop | File | Line |
|---|---|---|---|---|---|
| ☑ | 2 | BP | Thread | worker.cpp | 79 |

| Name | Type | Value |
|---|---|---|
| [Add New Expression] | | |

Process: QtThreadExample (1) Thread: 1.3 - Breakpoint Frame: Worker::doWork File: /home/bburns/Downloads/QtThreadExample/worker.cpp Line: 79

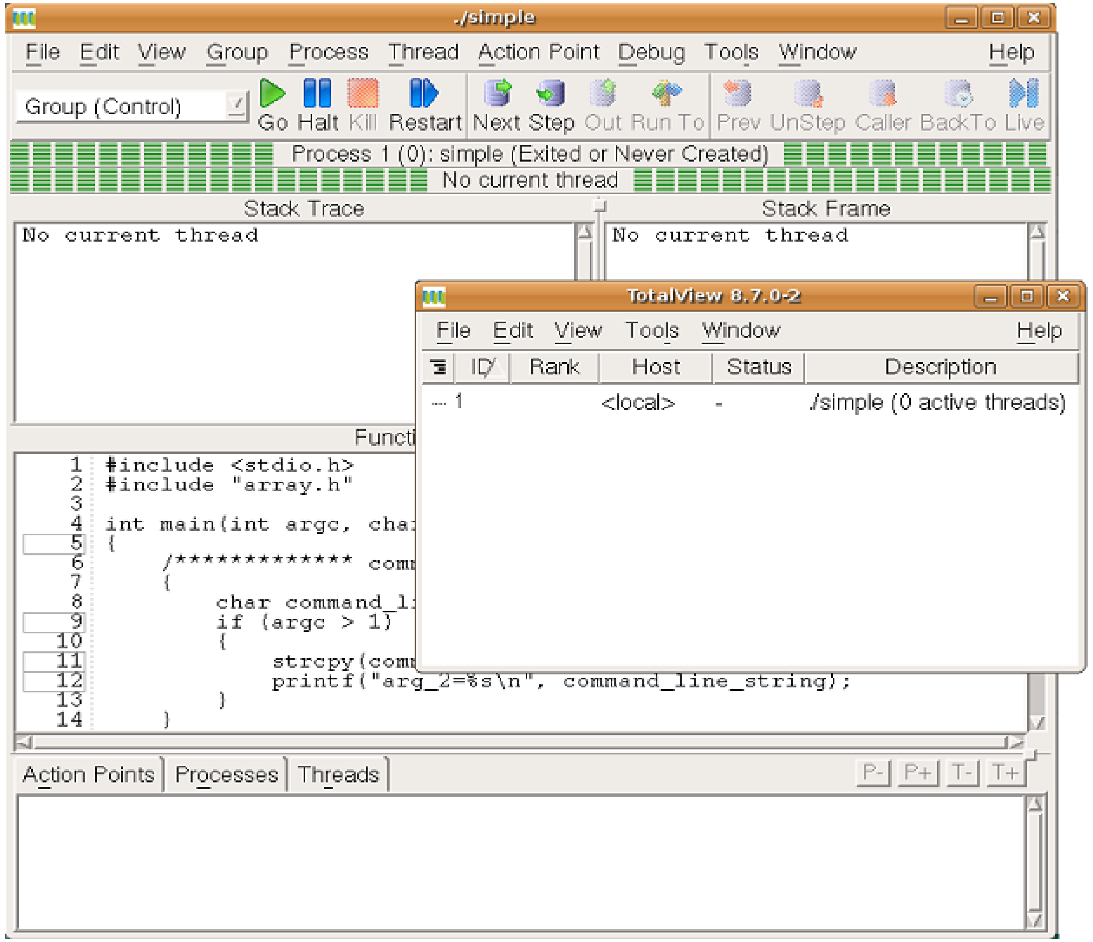
- Allowing YOU to have
  - Predictable development schedules
  - Less time spent debugging

# TotalView’s GUI

# TotalView’s Classic UI

- Original powerful design
- Better tested for high-scale MPI jobs
- Assembler support
- Better supported for Remote Display Client
- To use:
  - Set UI preference
  - Or command line argument

```
totalview -oldUI
```



./simple
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
Stack Trace
Stack Frame
Functi
Action Points Processes Threads
P- P+ T- T+
TotalView 8.7.0-2
File Edit View Tools Window Help

Process 1 (0): simple (Exited or Never Created)

No current thread

No current thread

No current thread

```
#include <stdio.h>
#include "array.h"

int main(int argc, cha
{
     /************** comm
     {
          char command_li
          if (argc > 1)
          {
               strcpy(comm
               printf("arg_2=%s\n", command_line_string);
          }
     }
```

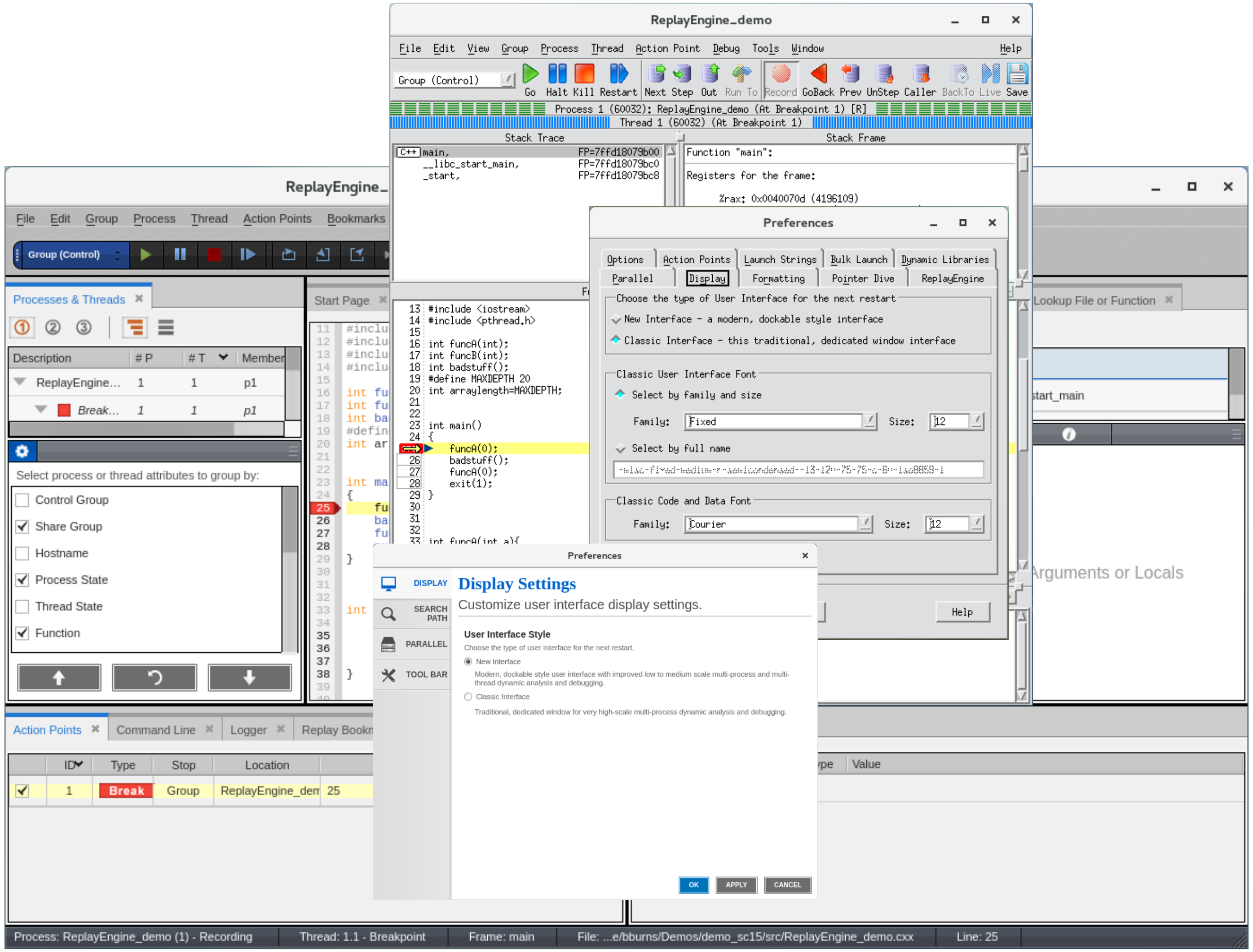
| ID | Rank | Host | Status | Description |
|---|---|---|---|---|
| 1 | | <local> | - | ./simple (0 active threads) |

# TotalView's New UI (default)

- Provides a modern, dockable interface
- Easier to use, better workflows
- An architecture to grow
- To use:
  - Set UI preference
  - Or command line argument

    `totalview -newUI`

- New UI gaps:
  - Missing array slicing and striding, view across, data visualization
  - No very high-scale support



ReplayEngine_demo

Process 1 (60032): ReplayEngine_demo (At Breakpoint 1) [R]

Thread 1 (60032) (At Breakpoint 1)

Stack Trace

```
C++ main,                FP=7ffd18079b00
    __libc_start_main,   FP=7ffd18079bc0
    _start,              FP=7ffd18079bc8
```

Stack Frame

```
Function "main":

Registers for the frame:

     %rax: 0x0040070d (4196109)
```

```
#include <iostream>
#include <pthread.h>

int funcA(int);
int funcB(int);
int badstuff();
#define MAXDEPTH 20
int arraylength=MAXDEPTH;


int main()
{
    funcA(0);
    badstuff();
    funcA(0);
    exit(1);
}



int funcA(int a){
```

Preferences

Choose the type of User Interface for the next restart

◇ New Interface - a modern, dockable style interface

◆ Classic Interface - this traditional, dedicated window interface

Classic User Interface Font

◆ Select by family and size

Family: Fixed Size: 12

◇ Select by full name

-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1

Classic Code and Data Font

Family: Courier Size: 12

Preferences

Display Settings

Customize user interface display settings.

User Interface Style

Choose the type of user interface for the next restart.

● New Interface

Modern, dockable style user interface with improved low to medium scale multi-process and multi-thread dynamic analysis and debugging.

○ Classic Interface

Traditional, dedicated window for very high-scale multi-process dynamic analysis and debugging.

Processes & Threads

| Description | # P | # T | Member |
|---|---|---|---|
| ReplayEngine... | 1 | 1 | p1 |
| Break... | 1 | 1 | p1 |

Select process or thread attributes to group by:

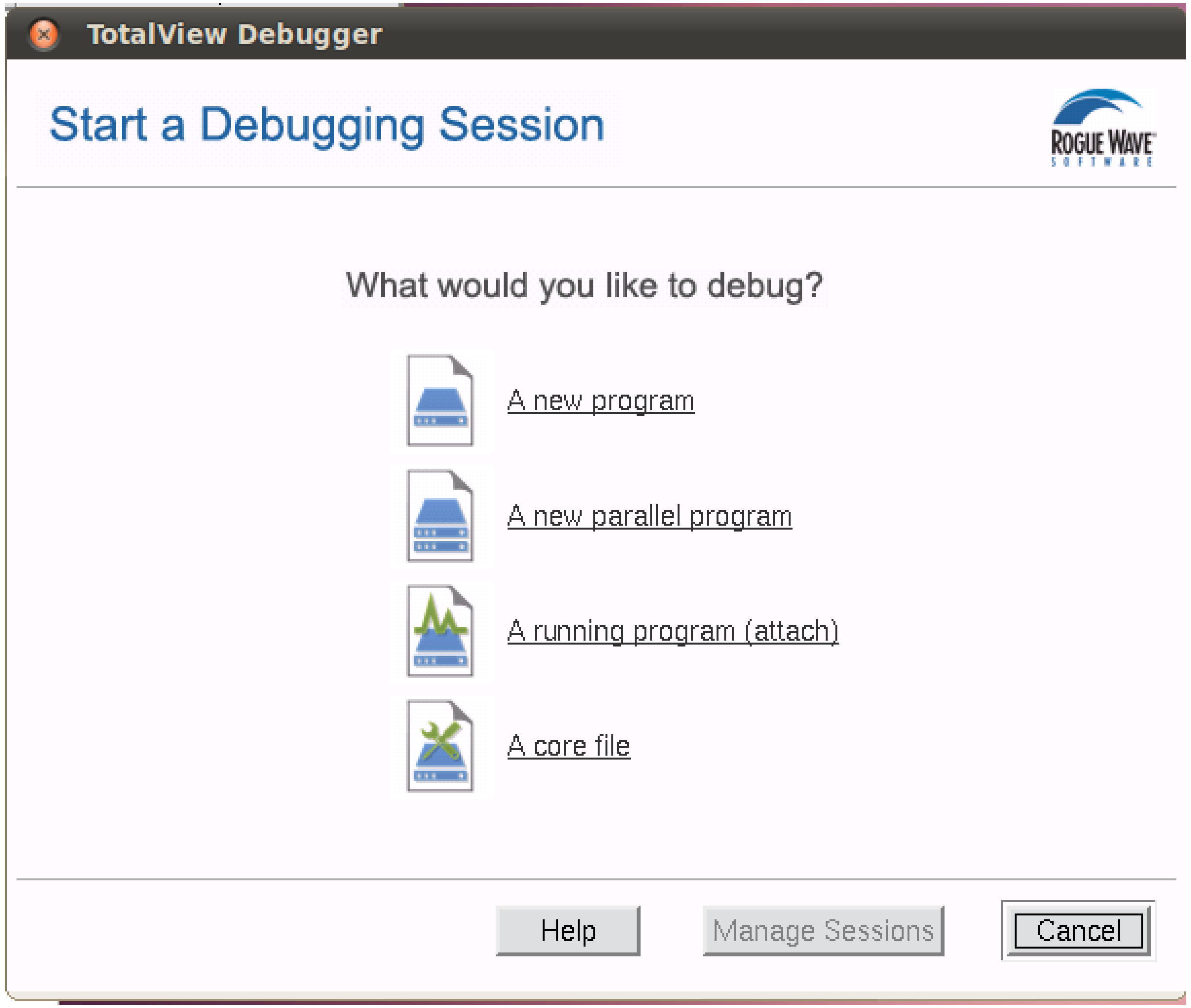
☐ Control Group
☑ Share Group
☐ Hostname
☑ Process State
☐ Thread State
☑ Function

Action Points

| | ID | Type | Stop | Location | |
|---|---|---|---|---|---|
| ☑ | 1 | Break | Group | ReplayEngine_dem | 25 |

Process: ReplayEngine_demo (1) - Recording | Thread: 1.1 - Breakpoint | Frame: main | File: ...e/bburns/Demos/demo_sc15/src/ReplayEngine_demo.cxx | Line: 25

# Intro & Starting Up

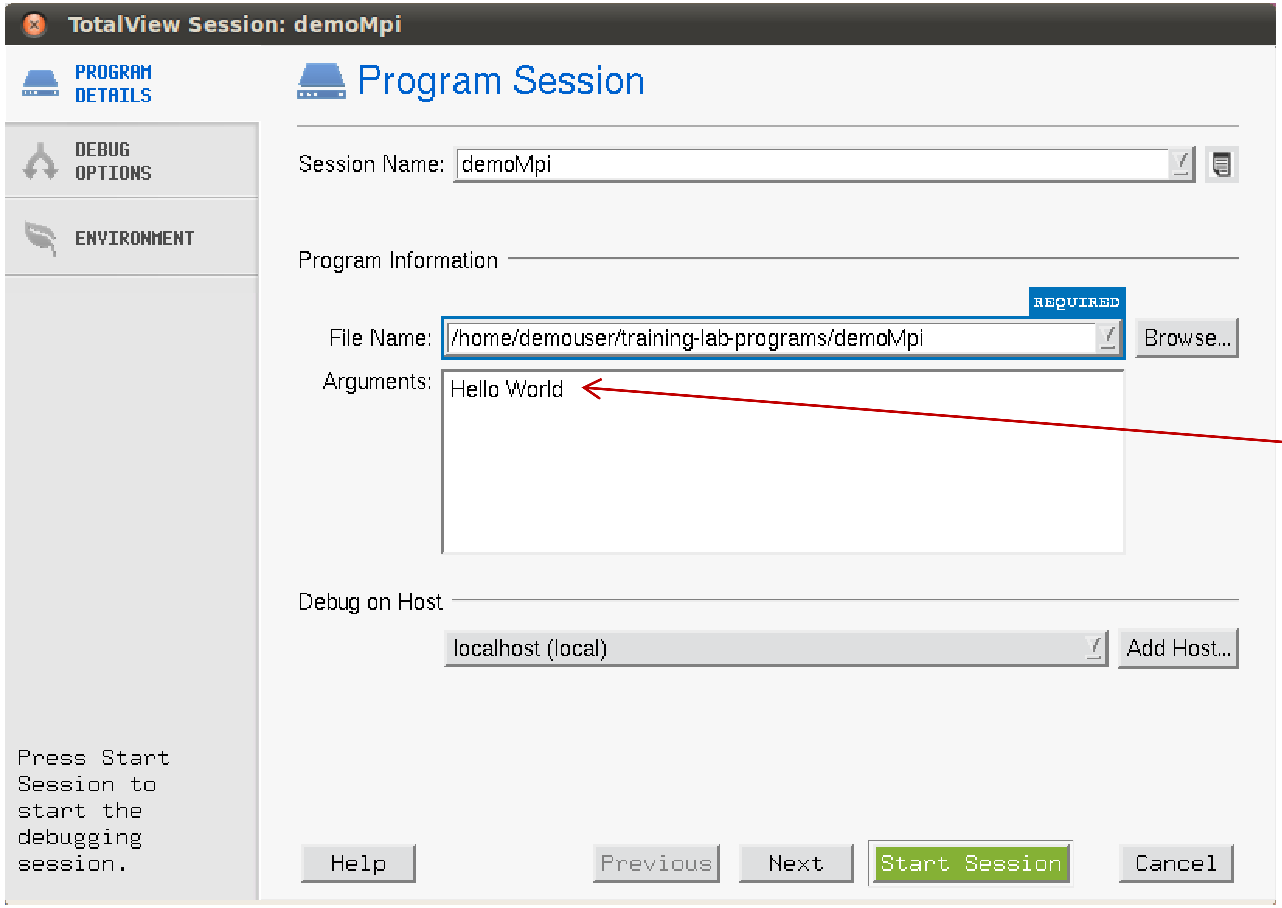


TotalView Debugger

Start a Debugging Session

What would you like to debug?

# Start New Process – Arguments



TotalView Session: demoMpi

Program Session

Session Name: demoMpi

Program Information

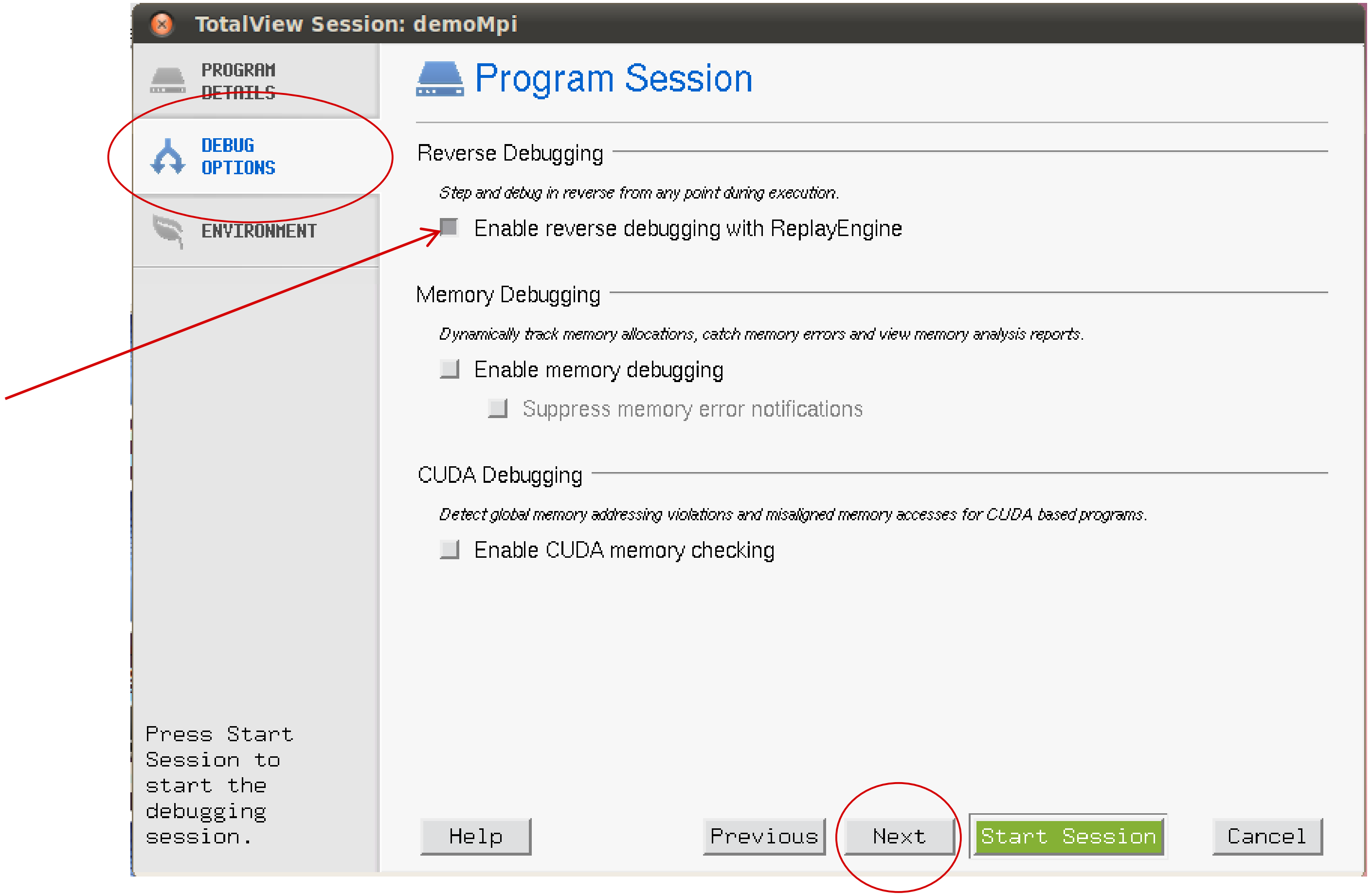
File Name: /home/demouser/training-lab-programs/demoMpi

Arguments: Hello World

Debug on Host: localhost (local)

Press Start Session to start the debugging session.

# Start New Process – Enable ReplayEngine



TotalView Session: demoMpi

## Program Session

Reverse Debugging

*Step and debug in reverse from any point during execution.*

■ Enable reverse debugging with ReplayEngine

Memory Debugging

*Dynamically track memory allocations, catch memory errors and view memory analysis reports.*

□ Enable memory debugging

□ Suppress memory error notifications

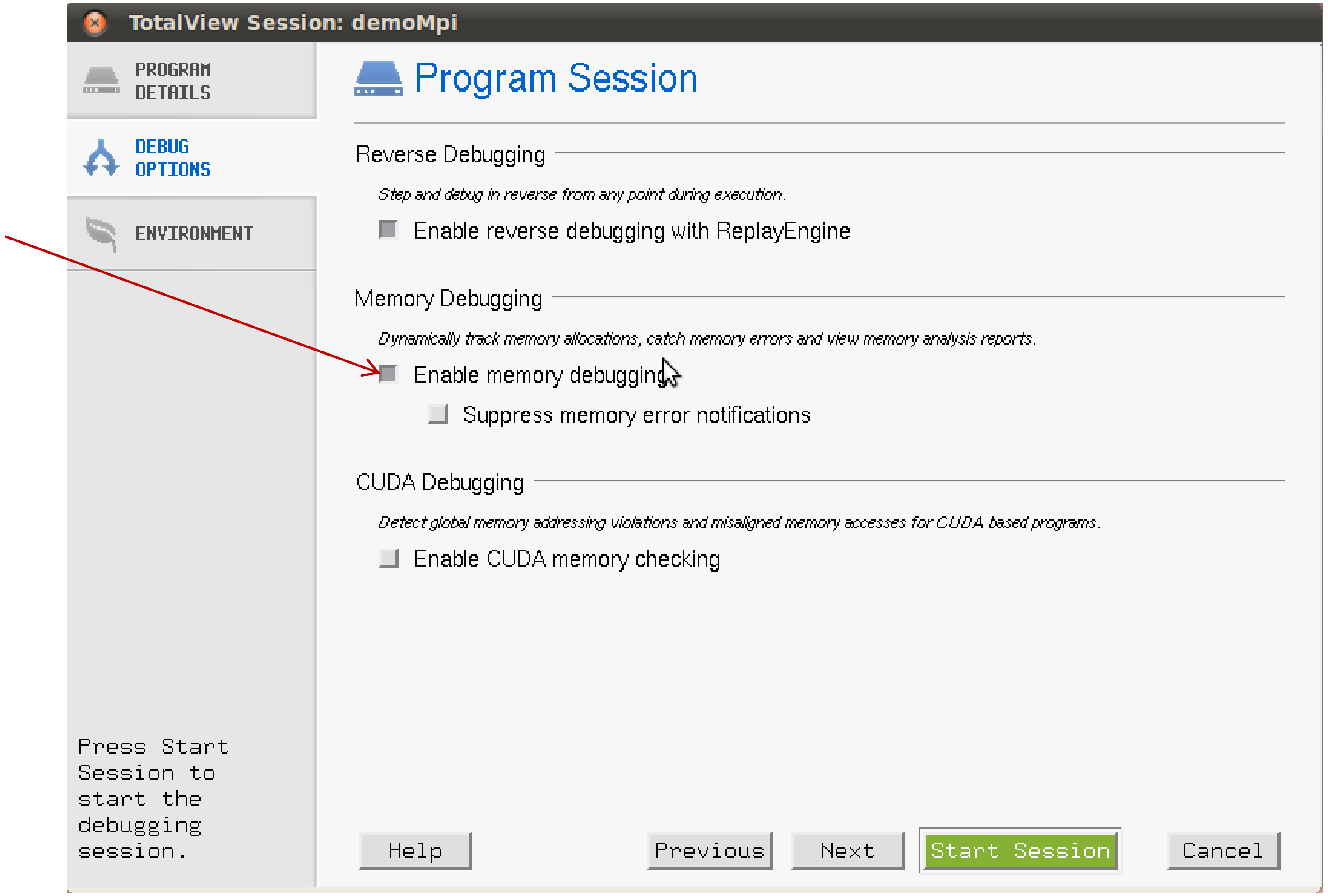
CUDA Debugging

*Detect global memory addressing violations and misaligned memory accesses for CUDA based programs.*

□ Enable CUDA memory checking

Press Start Session to start the debugging session.

# Start New Process – Memory Debugging



TotalView Session: demoMpi

Program Session

Reverse Debugging

*Step and debug in reverse from any point during execution.*

□ Enable reverse debugging with ReplayEngine

Memory Debugging

*Dynamically track memory allocations, catch memory errors and view memory analysis reports.*

□ Enable memory debugging

□ Suppress memory error notifications

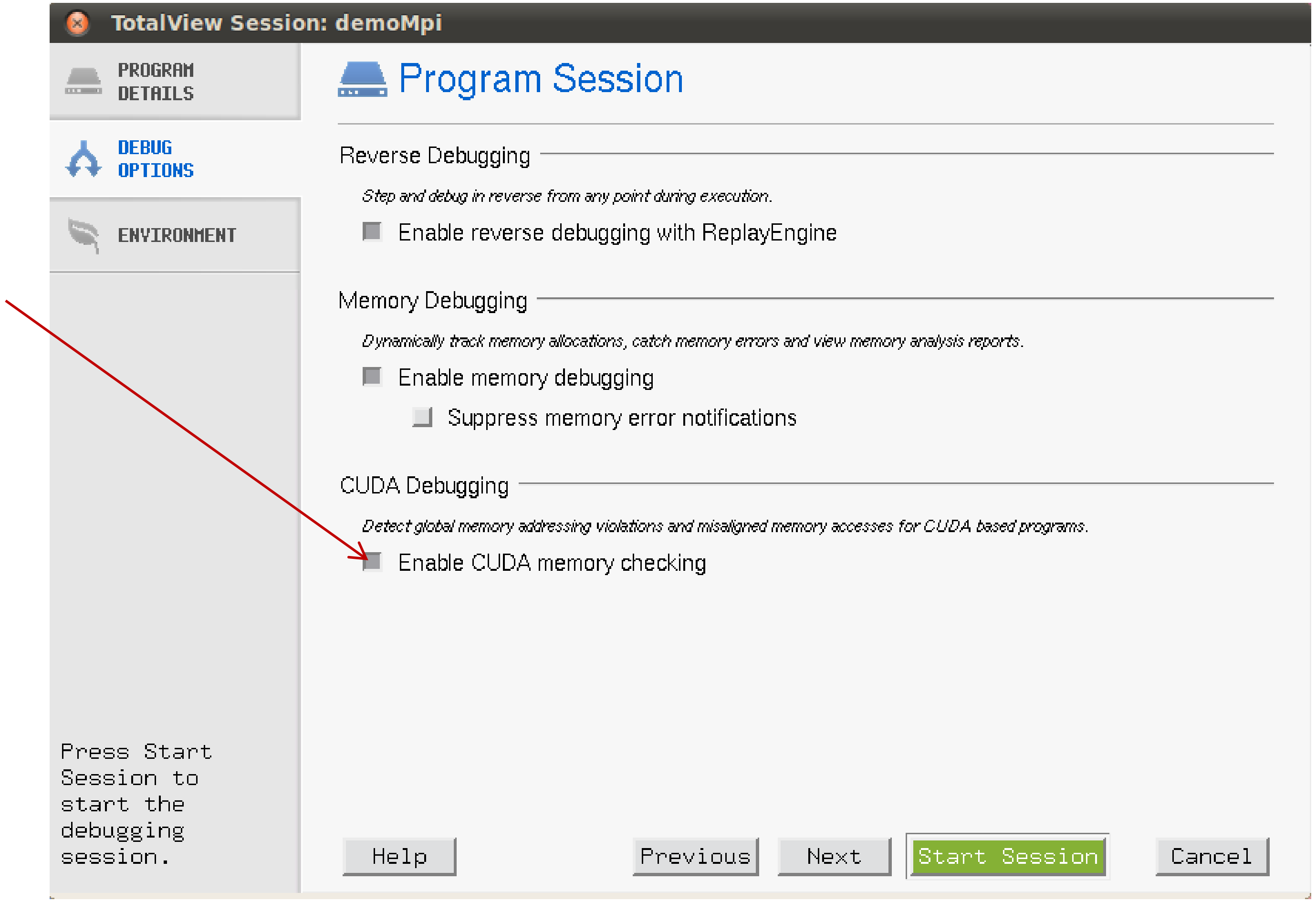
CUDA Debugging

*Detect global memory addressing violations and misaligned memory accesses for CUDA based programs.*

□ Enable CUDA memory checking

Press Start Session to start the debugging session.

# CUDA memory checking



TotalView Session: demoMpi

Program Session

Reverse Debugging

*Step and debug in reverse from any point during execution.*

□ Enable reverse debugging with ReplayEngine

Memory Debugging

*Dynamically track memory allocations, catch memory errors and view memory analysis reports.*

□ Enable memory debugging

□ Suppress memory error notifications

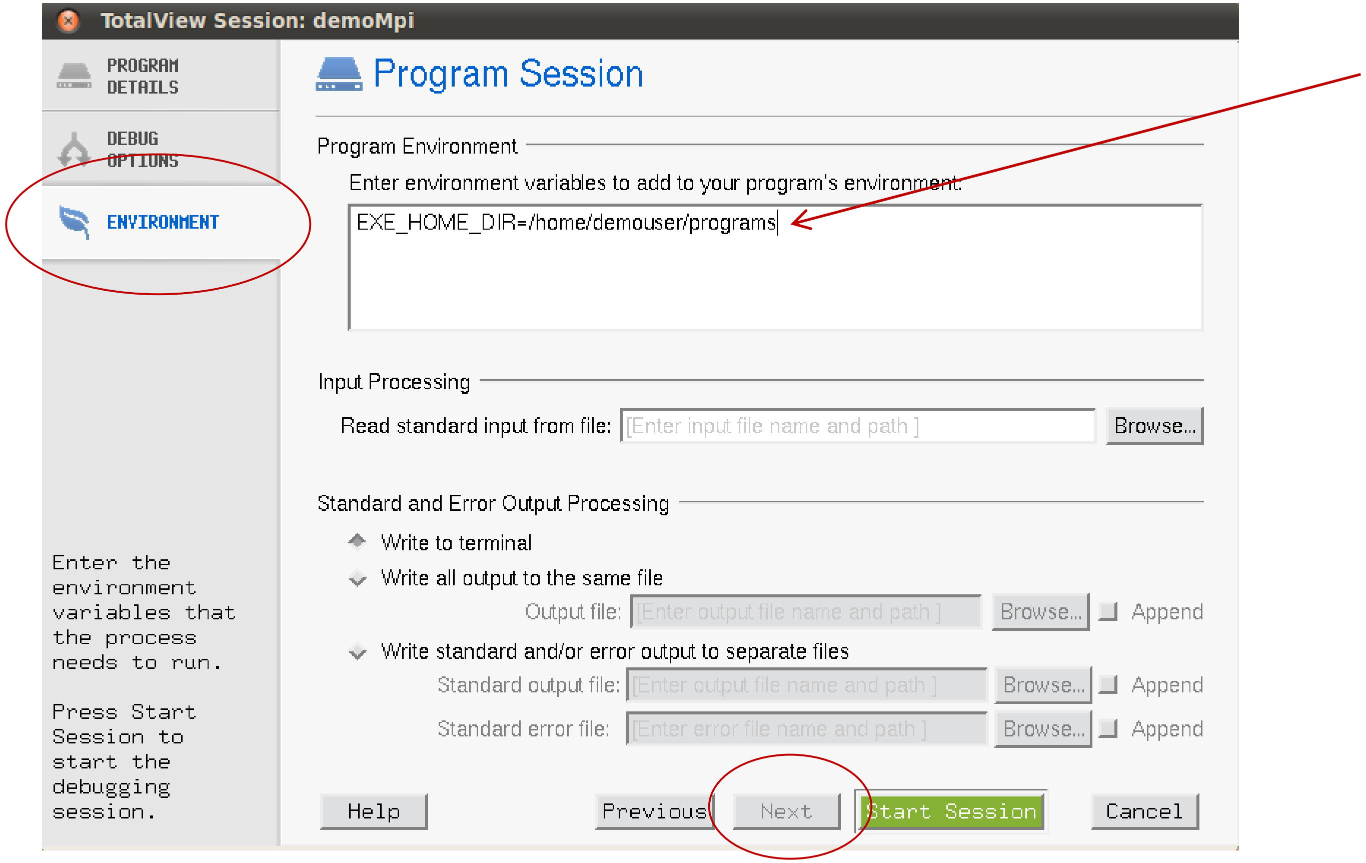
CUDA Debugging

*Detect global memory addressing violations and misaligned memory accesses for CUDA based programs.*

□ Enable CUDA memory checking

Press Start Session to start the debugging session.

# Set environment variables



TotalView Session: demoMpi

Program Session

Program Environment

Enter environment variables to add to your program's environment:

```
EXE_HOME_DIR=/home/demouser/programs
```

Input Processing

Read standard input from file: [Enter input file name and path ]

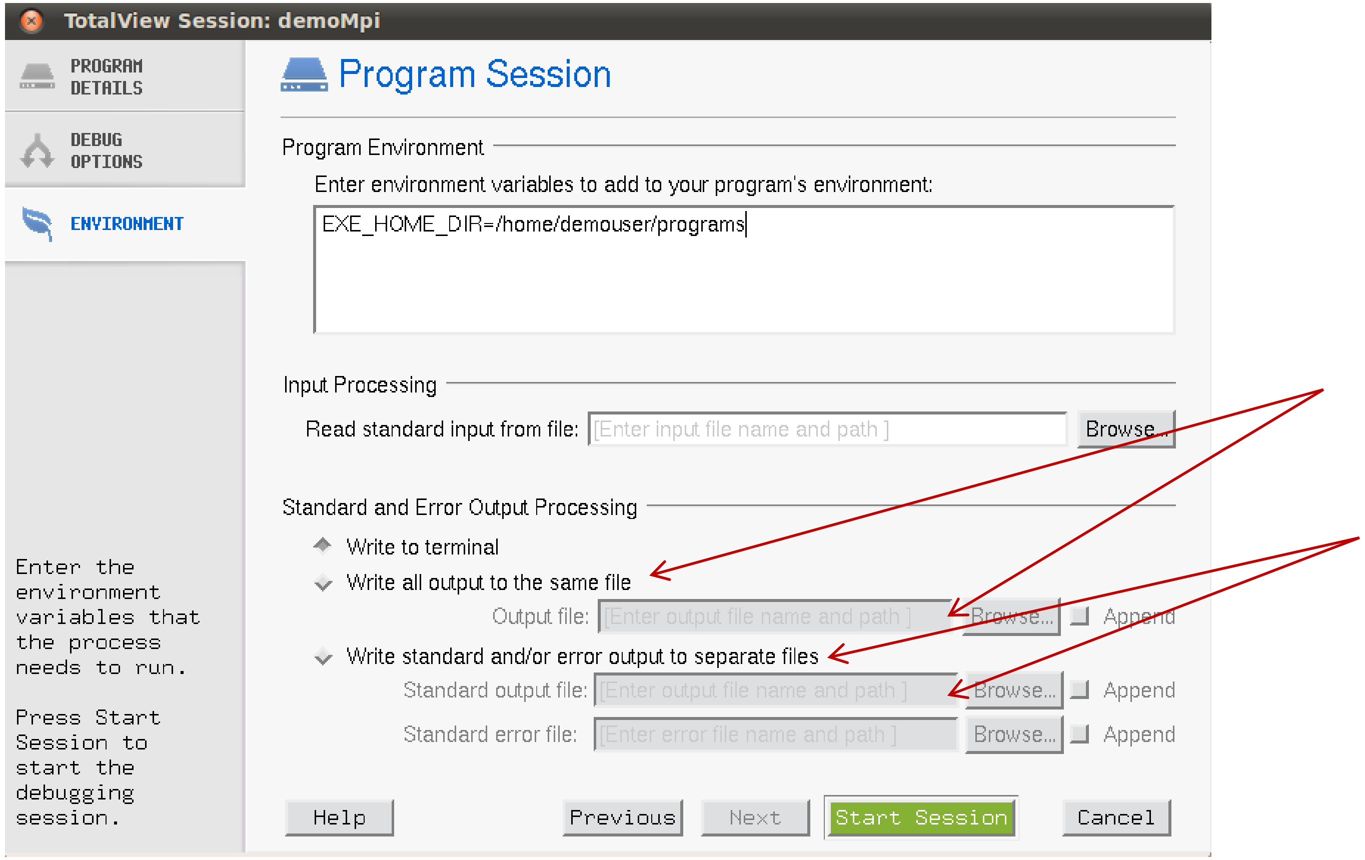
Standard and Error Output Processing

- Write to terminal
- Write all output to the same file
  - Output file: [Enter output file name and path ] □ Append
- Write standard and/or error output to separate files
  - Standard output file: [Enter output file name and path ] □ Append
  - Standard error file: [Enter error file name and path ] □ Append

Enter the environment variables that the process needs to run.

Press Start Session to start the debugging session.

# Standard I/O redirection



## Program Session

Program Environment

Enter environment variables to add to your program's environment:

```
EXE_HOME_DIR=/home/demouser/programs
```

Input Processing

Read standard input from file: [Enter input file name and path ] Browse...

Standard and Error Output Processing

● Write to terminal

○ Write all output to the same file

Output file: [Enter output file name and path ] Browse... □ Append

○ Write standard and/or error output to separate files

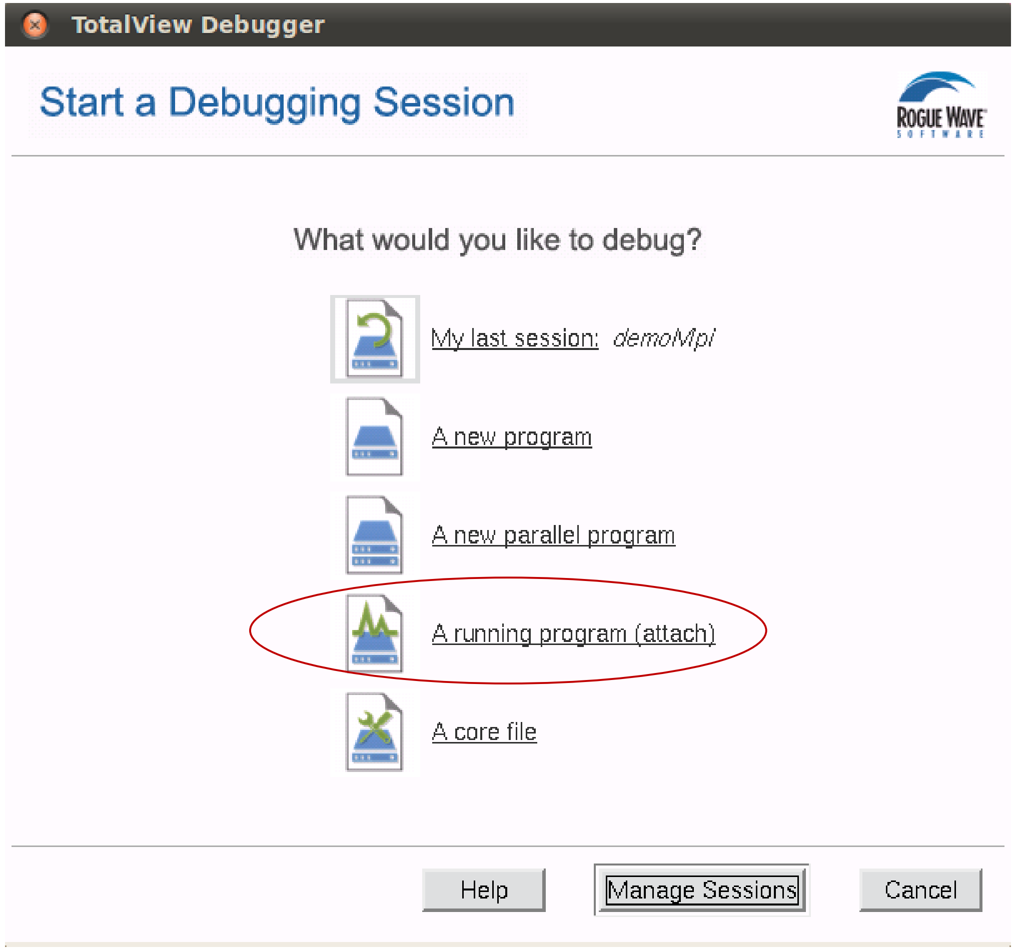
Standard output file: [Enter output file name and path ] Browse... □ Append

Standard error file: [Enter error file name and path ] Browse... □ Append

Enter the environment variables that the process needs to run.

Press Start Session to start the debugging session.

# Attach to Process



TotalView Debugger

Start a Debugging Session

What would you like to debug?

My last session: *demoMpi*

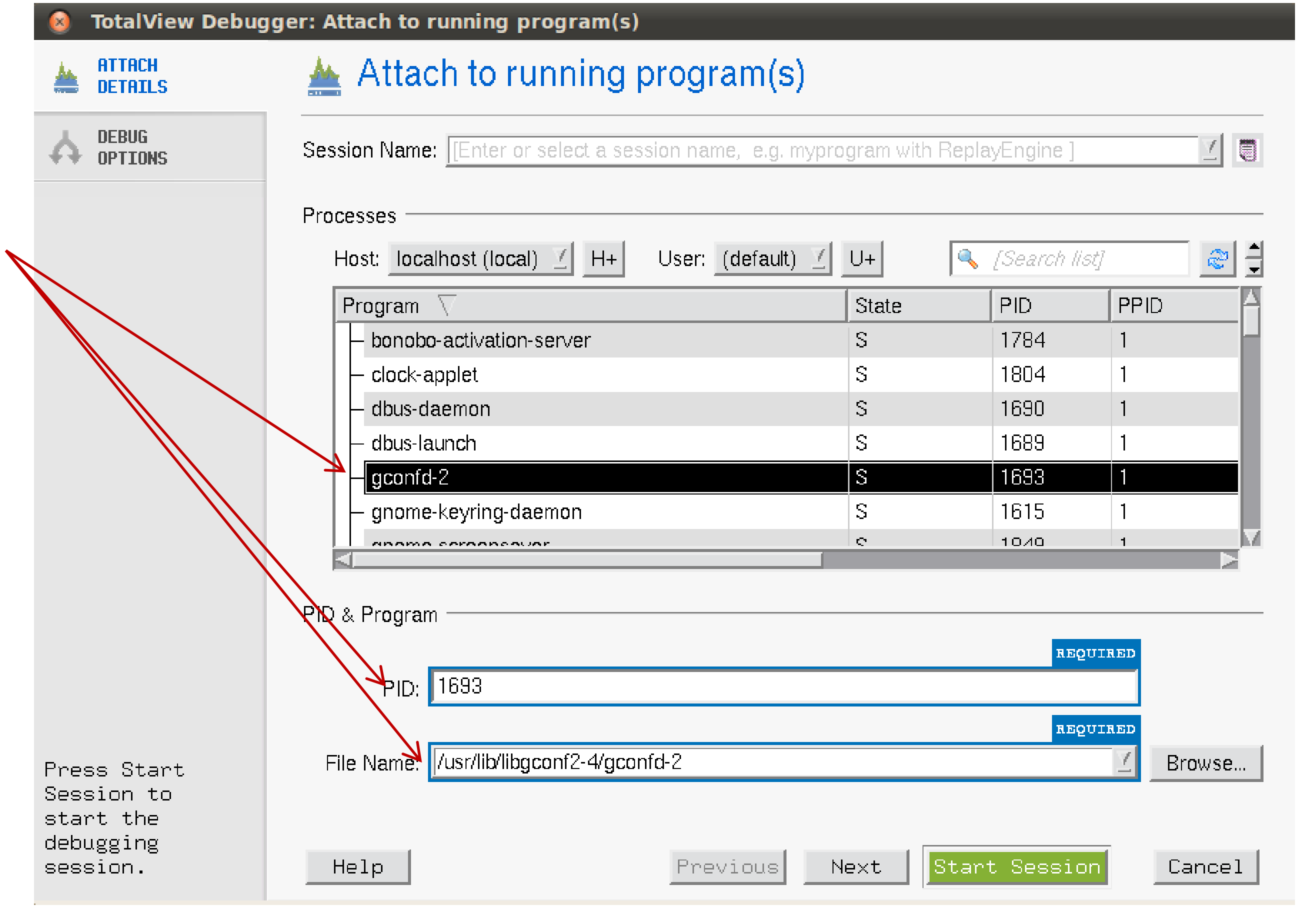
A new program

A new parallel program

A running program (attach)

A core file

# Attach to Process



Session Name: [Enter or select a session name, e.g. myprogram with ReplayEngine ]

Host: localhost (local)

User: (default)

[Search list]

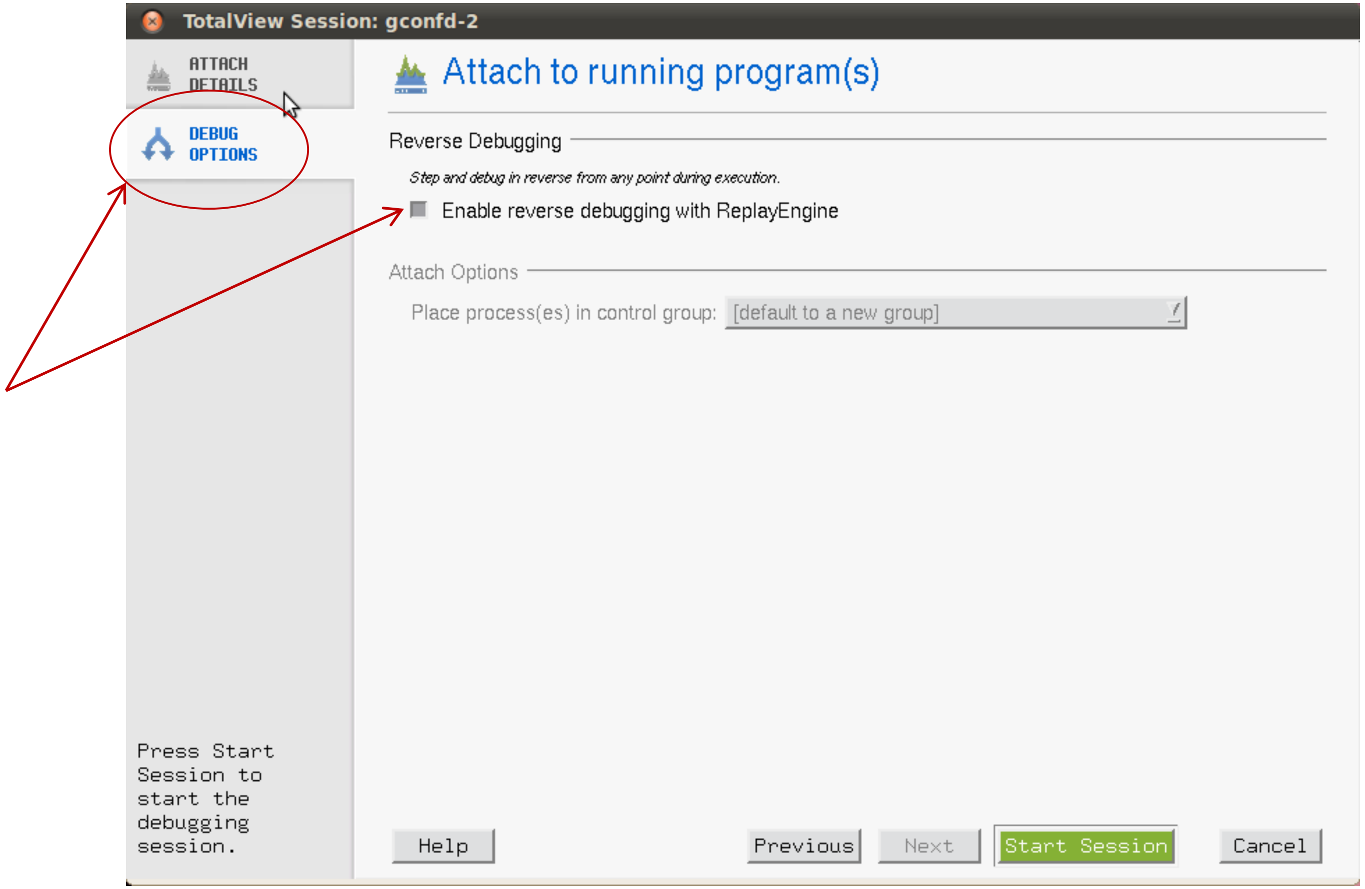
| Program | State | PID | PPID |
|---|---|---|---|
| bonobo-activation-server | S | 1784 | 1 |
| clock-applet | S | 1804 | 1 |
| dbus-daemon | S | 1690 | 1 |
| dbus-launch | S | 1689 | 1 |
| gconfd-2 | S | 1693 | 1 |
| gnome-keyring-daemon | S | 1615 | 1 |
| gnome-screensaver | S | 1949 | 1 |

PID: 1693

File Name: /usr/lib/libgconf2-4/gconfd-2

Press Start Session to start the debugging session.

# Attach to Process – Enable Replay Engine



TotalView Session: gconfd-2

## Attach to running program(s)

Reverse Debugging

*Step and debug in reverse from any point during execution.*

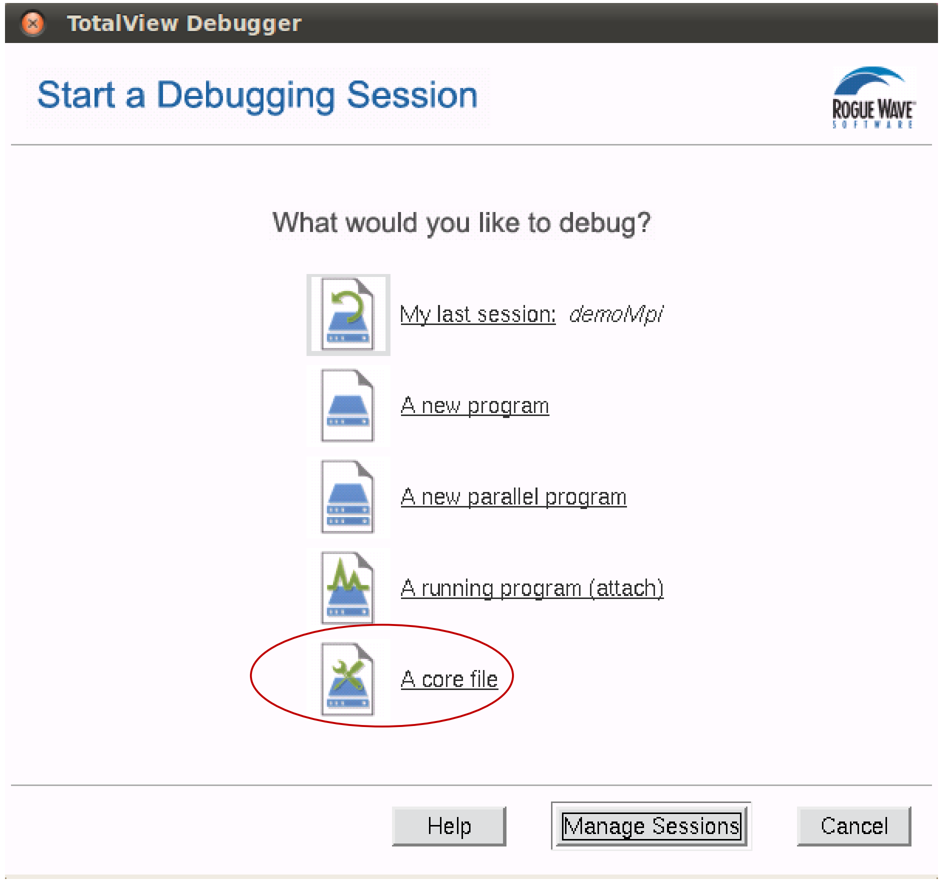
□ Enable reverse debugging with ReplayEngine

Attach Options

Place process(es) in control group: [default to a new group]

Press Start Session to start the debugging session.

# Open a Core File



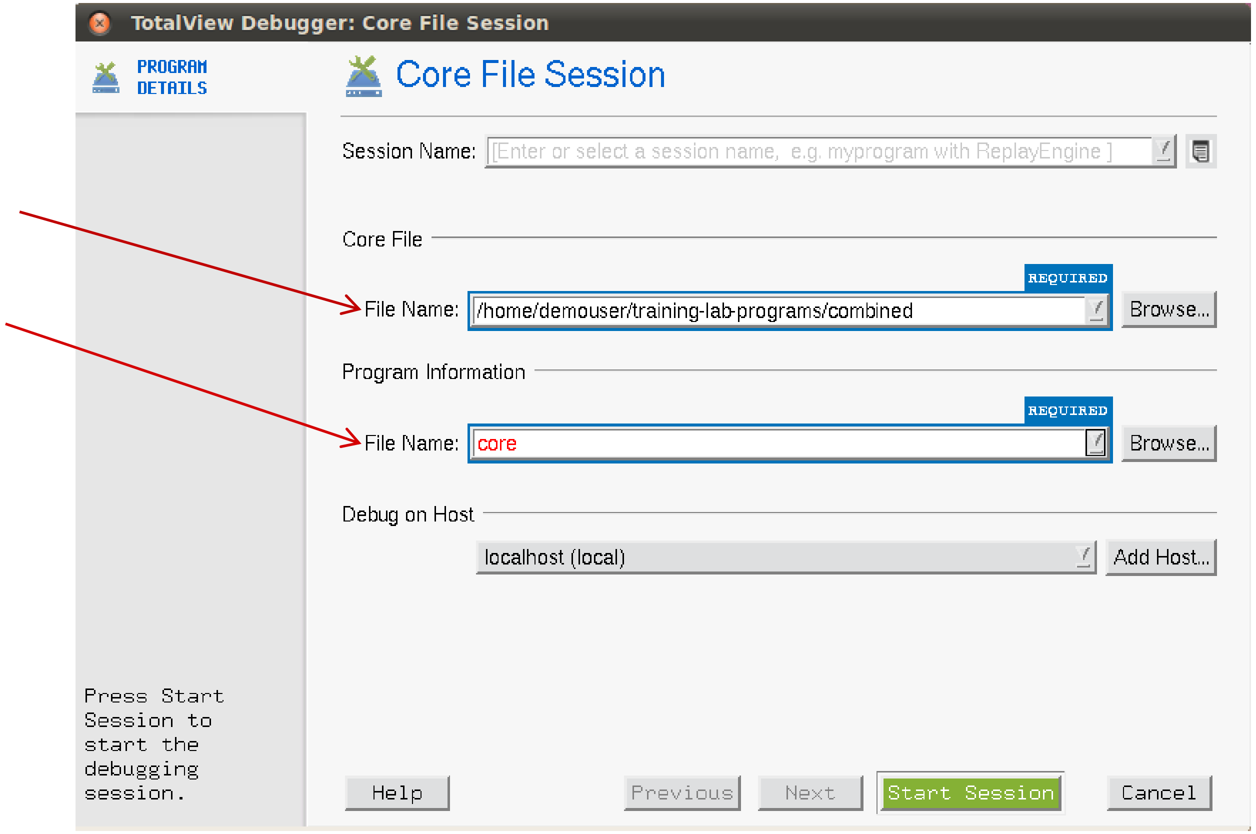
TotalView Debugger

Start a Debugging Session

What would you like to debug?

- My last session: *demoMpi*
- A new program
- A new parallel program
- A running program (attach)
- A core file

# Open a Core File



TotalView Debugger: Core File Session

Core File Session

Session Name: [Enter or select a session name, e.g. myprogram with ReplayEngine ]

Core File

File Name: /home/demouser/training-lab-programs/combined

Program Information

File Name: core

Debug on Host

localhost (local)

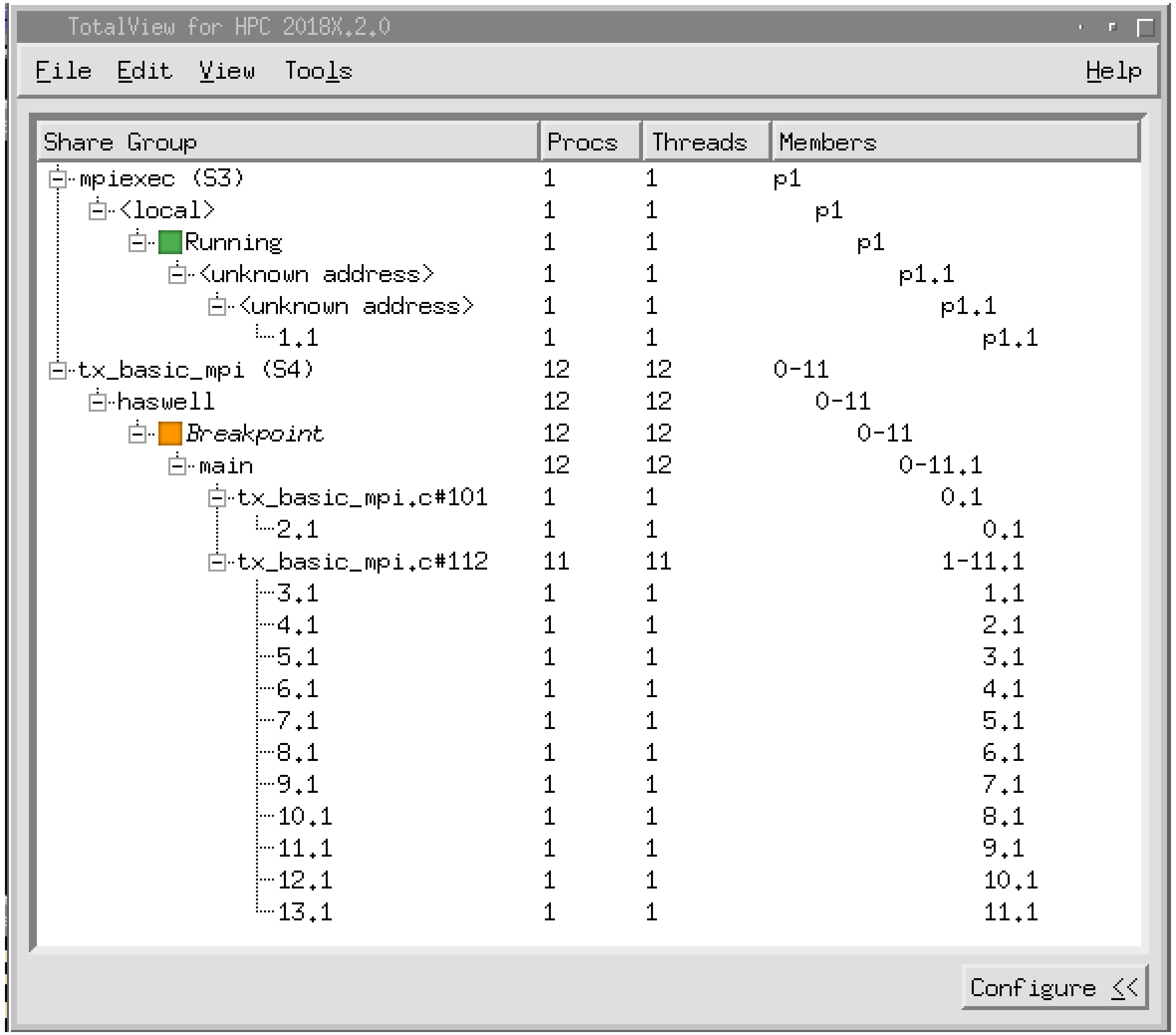
Press Start Session to start the debugging session.

# Process Control & Navigation

# Interface Concepts

## Root Window

- State of all processes being debugged
- Process and Thread status
- Instant navigation access
- Sort and aggregate by status



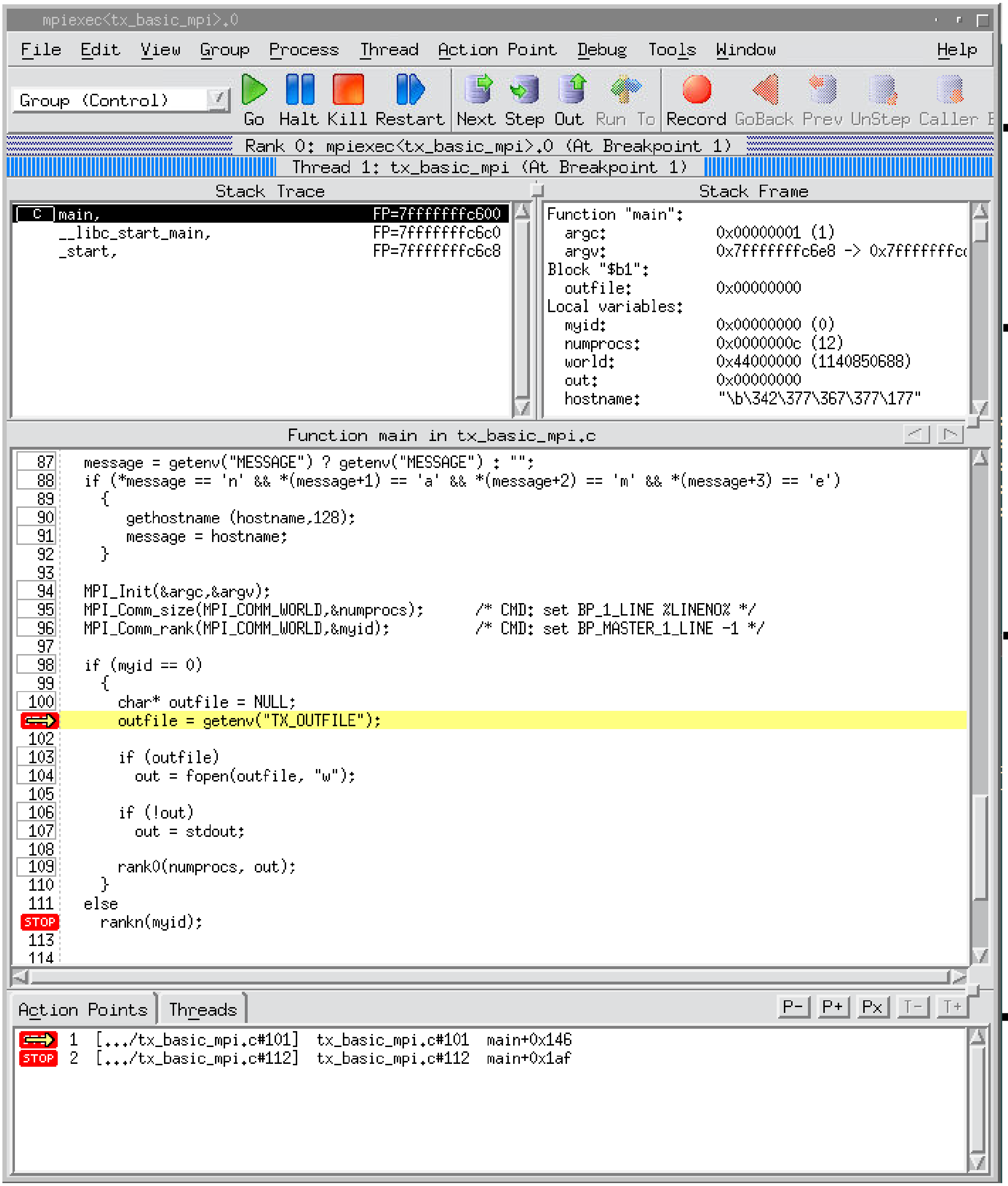
```
Share Group                          Procs   Threads   Members
⊟ mpiexec (S3)                       1       1         p1
   ⊟ <local>                         1       1           p1
      ⊟ 🟩 Running                   1       1             p1
         ⊟ <unknown address>         1       1               p1.1
            ⊟ <unknown address>      1       1                 p1.1
               └ 1.1                 1       1                   p1.1
⊟ tx_basic_mpi (S4)                  12      12        0-11
   ⊟ haswell                         12      12          0-11
      ⊟ 🟧 Breakpoint                12      12            0-11
         ⊟ main                      12      12              0-11.1
            ⊟ tx_basic_mpi.c#101     1       1                 0.1
               └ 2.1                 1       1                   0.1
            ⊟ tx_basic_mpi.c#112     11      11                1-11.1
               ├ 3.1                 1       1                   1.1
               ├ 4.1                 1       1                   2.1
               ├ 5.1                 1       1                   3.1
               ├ 6.1                 1       1                   4.1
               ├ 7.1                 1       1                   5.1
               ├ 8.1                 1       1                   6.1
               ├ 9.1                 1       1                   7.1
               ├ 10.1                1       1                   8.1
               ├ 11.1                1       1                   9.1
               ├ 12.1                1       1                   10.1
               └ 13.1                1       1                   11.1
```

# Process Window Overview

Stack Trace Pane

**Provides detailed state of one process, or a single thread within a process**

**A single point of control for the process and other related processes**



Rank 0: mpiexec<tx_basic_mpi>.0 (At Breakpoint 1)

Thread 1: tx_basic_mpi (At Breakpoint 1)

```
C  main,                 FP=7fffffffc600
   __libc_start_main,    FP=7fffffffc6c0
   _start,               FP=7fffffffc6c8
```

```
Function "main":
  argc:          0x00000001 (1)
  argv:          0x7fffffffc6e8 -> 0x7fffffffc
Block "$b1":
  outfile:       0x00000000
Local variables:
  myid:          0x00000000 (0)
  numprocs:      0x0000000c (12)
  world:         0x44000000 (1140850688)
  out:           0x00000000
  hostname:      "\b\342\377\367\377\177"
```

Function main in tx_basic_mpi.c

```
  message = getenv("MESSAGE") ? getenv("MESSAGE") : "";
  if (*message == 'n' && *(message+1) == 'a' && *(message+2) == 'm' && *(message+3) == 'e')
    {
      gethostname (hostname,128);
      message = hostname;
    }

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);      /* CMD: set BP_1_LINE %LINENO% */
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);          /* CMD: set BP_MASTER_1_LINE -1 */

  if (myid == 0)
    {
      char* outfile = NULL;
      outfile = getenv("TX_OUTFILE");

      if (outfile)
        out = fopen(outfile, "w");

      if (!out)
        out = stdout;

      rank0(numprocs, out);
    }
  else
    rankn(myid);
```

```
1  [.../tx_basic_mpi.c#101]  tx_basic_mpi.c#101  main+0x146
STOP 2  [.../tx_basic_mpi.c#112]  tx_basic_mpi.c#112  main+0x1af
```

Toolbar

Stack Frame Pane

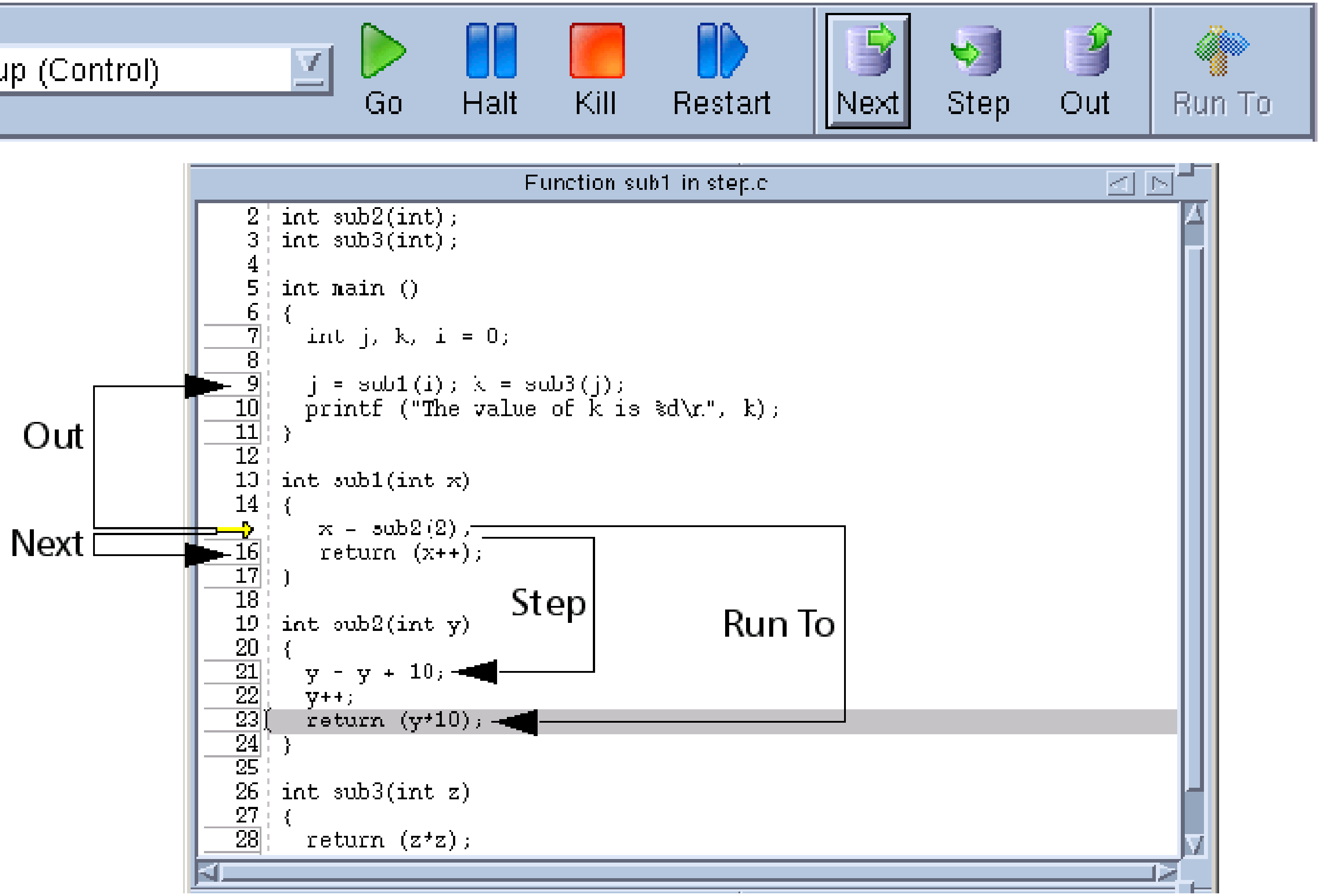
Source Pane

Tabbed Area

# Stepping Commands

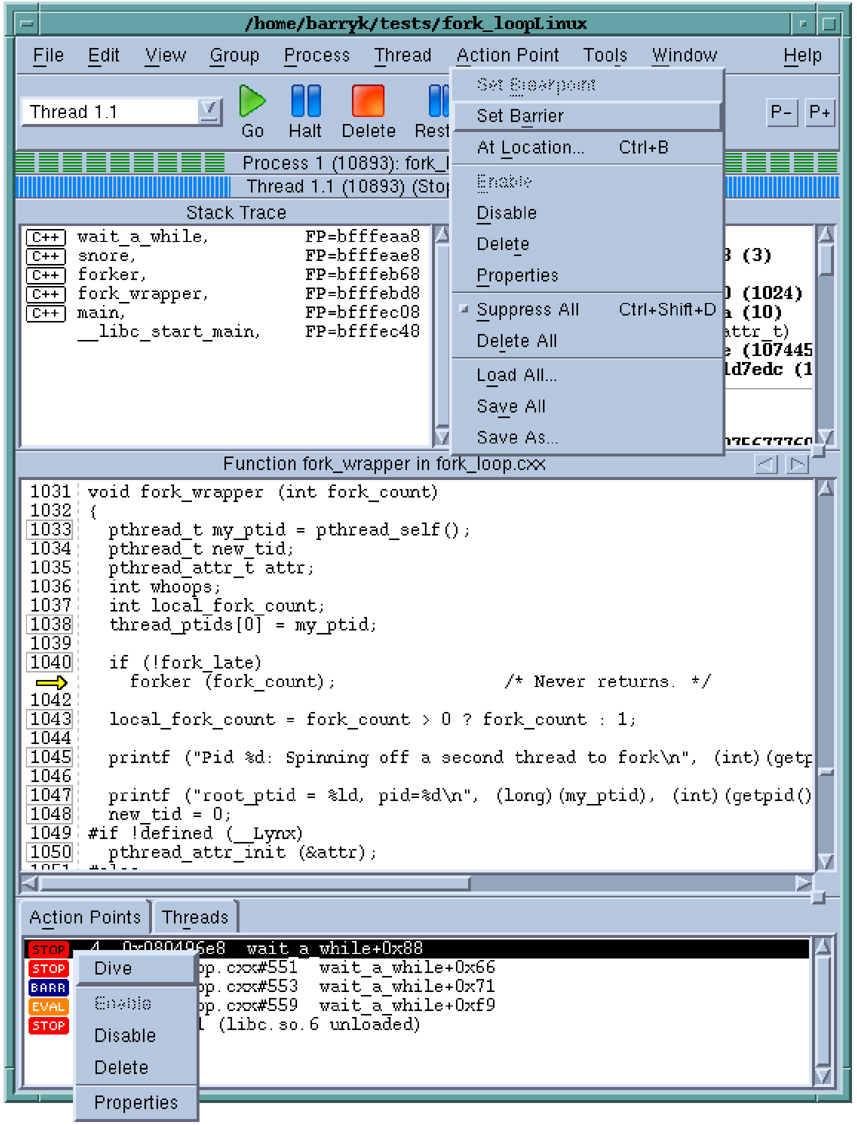
Based on

PC location



```
int sub2(int);
int sub3(int);

int main ()
{
  int j, k, i = 0;

  j = sub1(i); k = sub3(j);
  printf ("The value of k is %d\n", k);
}

int sub1(int x)
{
   x = sub2(2);
   return (x++);
}

int sub2(int y)
{
  y = y + 10;
  y++;
  return (y+10);
}

int sub3(int z)
{
  return (z*z);
```

# Action Points



```
C++  wait_a_while,           FP=bfffeaa8
C++  snore,                  FP=bfffeae8
C++  forker,                 FP=bfffeb68
C++  fork_wrapper,           FP=bfffebd8
C++  main,                   FP=bfffec08
     __libc_start_main,      FP=bfffec48
```

```
void fork_wrapper (int fork_count)
{
  pthread_t my_ptid = pthread_self();
  pthread_t new_tid;
  pthread_attr_t attr;
  int whoops;
  int local_fork_count;
  thread_ptids[0] = my_ptid;

  if (!fork_late)
    forker (fork_count);                  /* Never returns. */

  local_fork_count = fork_count > 0 ? fork_count : 1;

  printf ("Pid %d: Spinning off a second thread to fork\n", (int)(getp

  printf ("root_ptid = %ld, pid=%d\n", (long)(my_ptid), (int)(getpid()
  new_tid = 0;
#if !defined (__Lynx)
  pthread_attr_init (&attr);
```

```
STOP  4  0x080496e8  wait_a_while+0x88
STOP     op.cxx#551  wait_a_while+0x66
BARR     op.cxx#553  wait_a_while+0x71
EVAL     op.cxx#559  wait_a_while+0xf9
STOP     1 (libc.so.6 unloaded)
```

***Breakpoints***

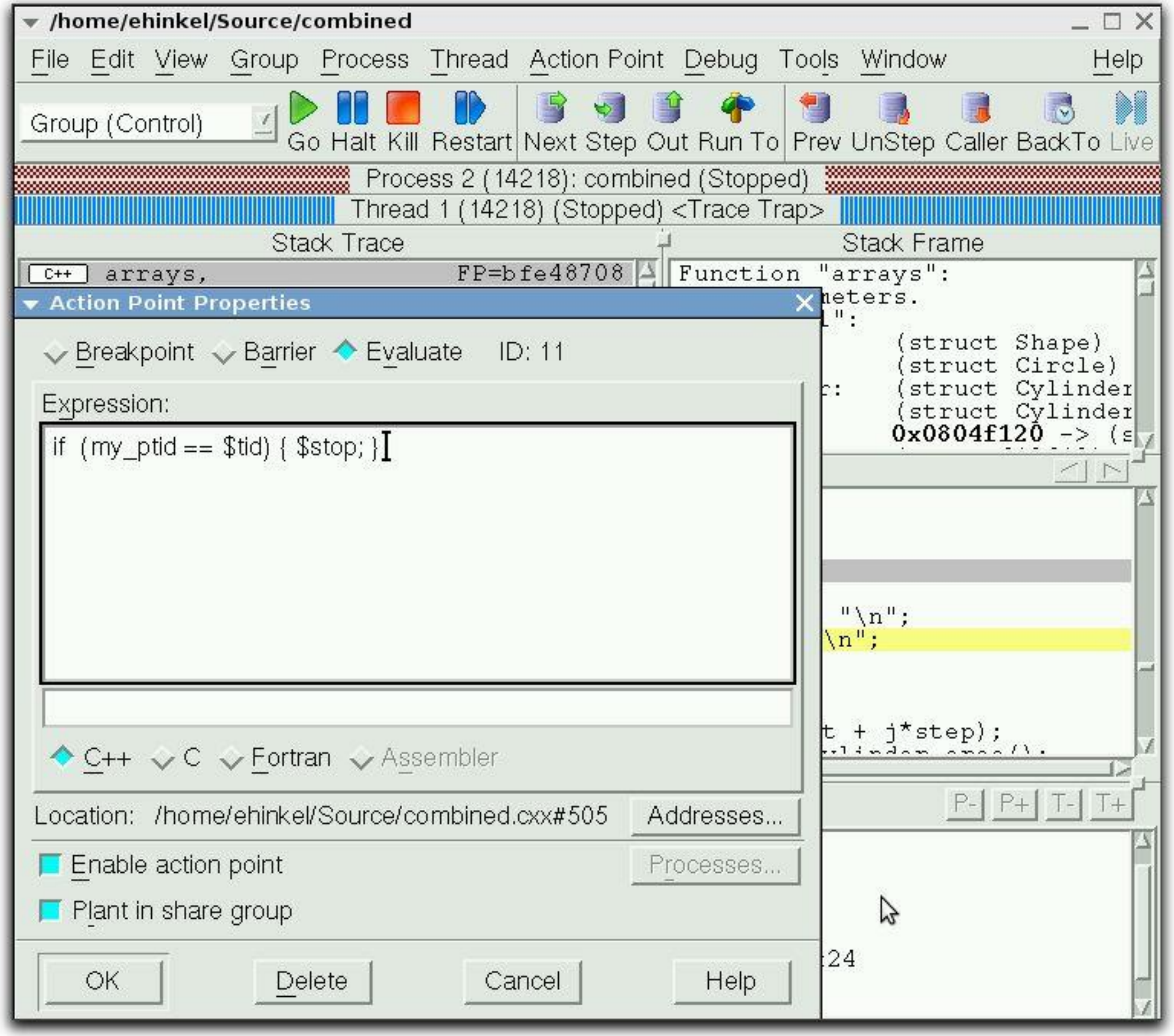
***Barrier Points***

***Conditional Breakpoints***

***Evaluation Points***

***Watchpoints***

# Conditional Breakpoint



/home/ehinkel/Source/combined

Process 2 (14218): combined (Stopped)

Thread 1 (14218) (Stopped) <Trace Trap>

Action Point Properties

○ Breakpoint ○ Barrier ● Evaluate ID: 11

Expression:

```
if (my_ptid == $tid) { $stop; }
```

● C++ ○ C ○ Fortran ○ Assembler

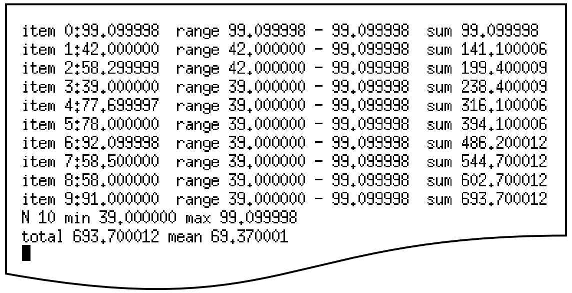
Location: /home/ehinkel/Source/combined.cxx#505

☑ Enable action point

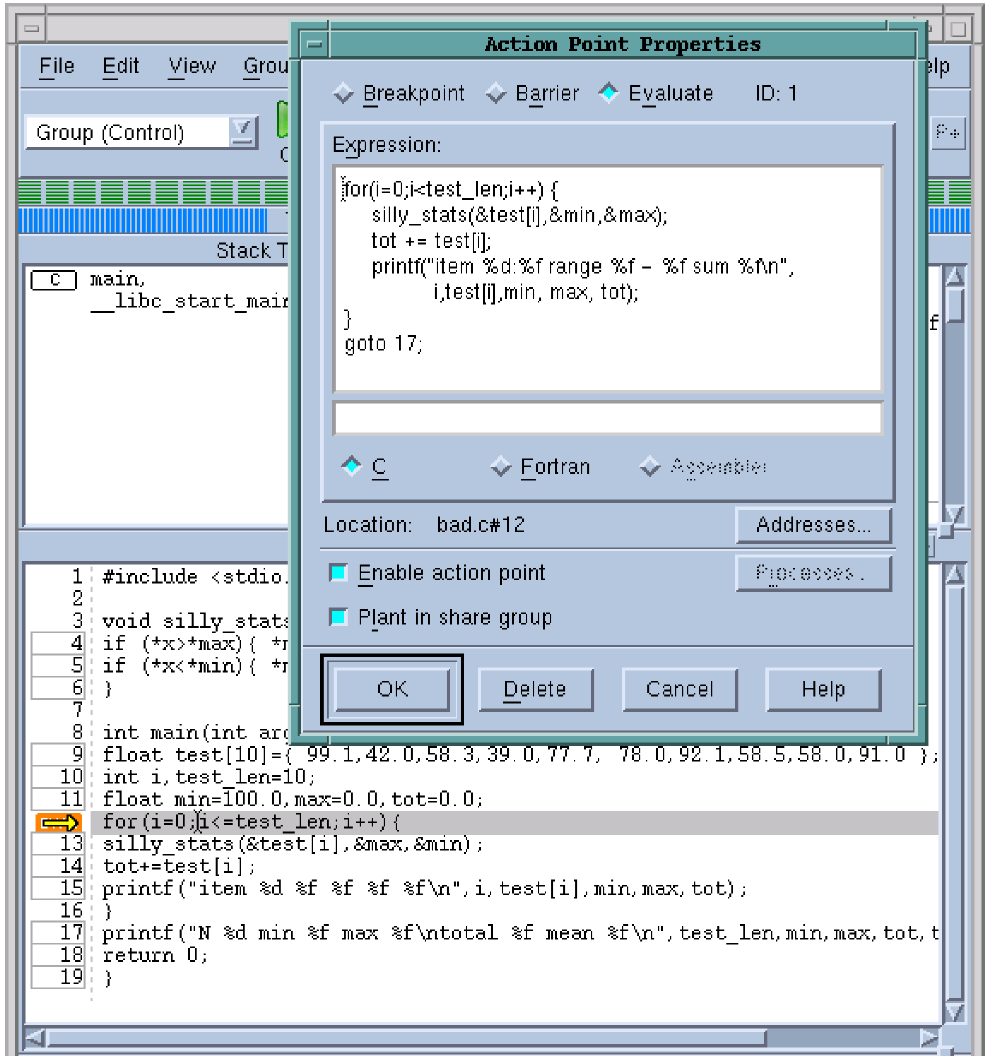
☑ Plant in share group

# Evaluation Point – Test Fixes on the Fly!

- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Some limitations:
  - Can’t use C++ constructors



```
item 0:99.099998  range 99.099998 - 99.099998  sum 99.099998
item 1:42.000000  range 42.000000 - 99.099998  sum 141.100006
item 2:58.299999  range 42.000000 - 99.099998  sum 199.400009
item 3:39.000000  range 39.000000 - 99.099998  sum 238.400009
item 4:77.699997  range 39.000000 - 99.099998  sum 316.100006
item 5:78.000000  range 39.000000 - 99.099998  sum 394.100006
item 6:92.099998  range 39.000000 - 99.099998  sum 486.200012
item 7:58.500000  range 39.000000 - 99.099998  sum 544.700012
item 8:58.000000  range 39.000000 - 99.099998  sum 602.700012
item 9:91.000000  range 39.000000 - 99.099998  sum 693.700012
N 10 min 39.000000 max 99.099998
total 693.700012 mean 69.370001
█
```



○ Breakpoint ○ Barrier ● Evaluate ID: 1

Expression:

```
for(i=0;i<test_len;i++) {
    silly_stats(&test[i],&min,&max);
    tot += test[i];
    printf("item %d:%f range %f - %f sum %f\n",
          i,test[i],min, max, tot);
}
goto 17;
```

● C ○ Fortran ○ Assembler

Location: bad.c#12

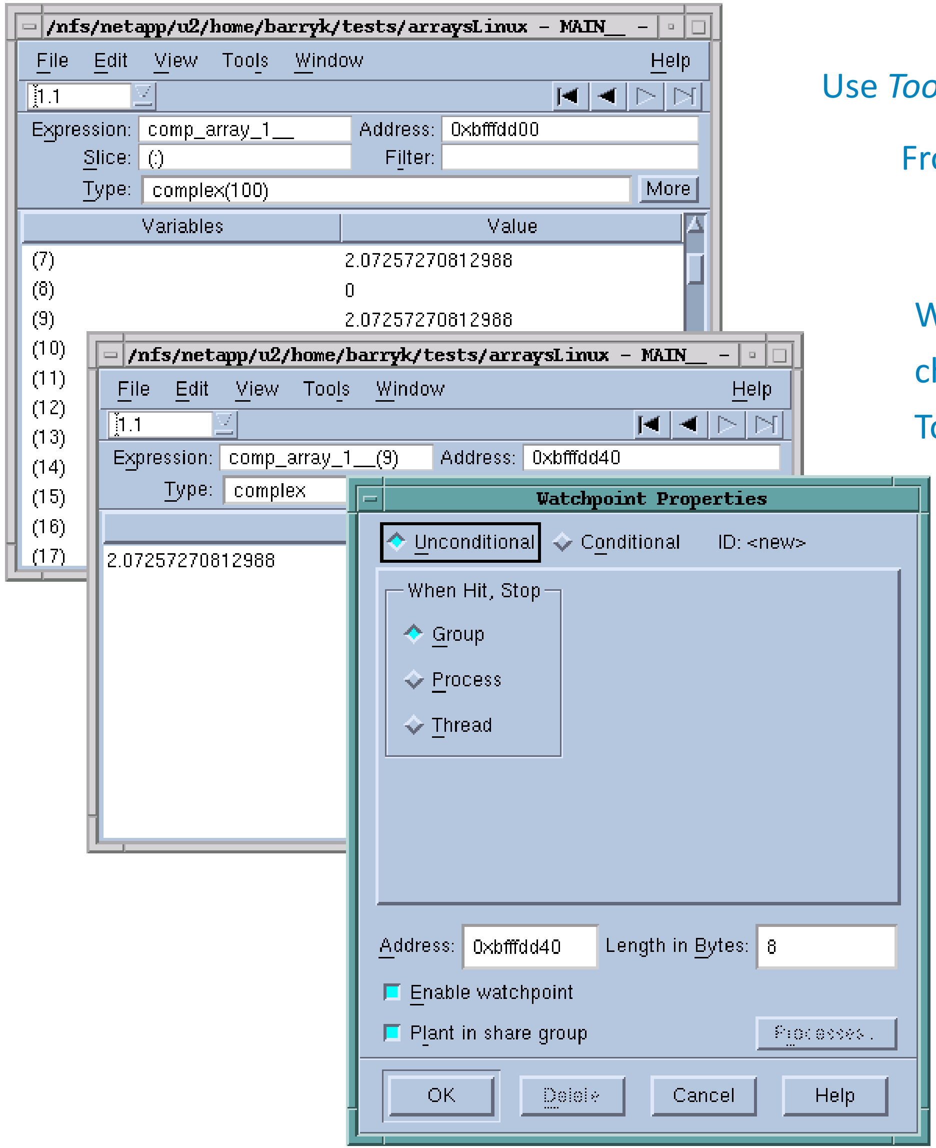
☑ Enable action point

☑ Plant in share group

```
main,
__libc_start_mair
```

```
 1 #include <stdio.
 2
 3 void silly_stats
 4 if (*x>*max){ *r
 5 if (*x<*min){ *r
 6 }
 7
 8 int main(int arg
 9 float test[10]={ 99.1,42.0,58.3,39.0,77.7, 78.0,92.1,58.5,58.0,91.0 };
10 int i,test_len=10;
11 float min=100.0,max=0.0,tot=0.0;
   for(i=0;i<=test_len;i++){
13 silly_stats(&test[i],&max,&min);
14 tot+=test[i];
15 printf("item %d %f %f %f %f\n",i,test[i],min,max,tot);
16 }
17 printf("N %d min %f max %f\ntotal %f mean %f\n",test_len,min,max,tot,t
18 return 0;
19 }
```

# Watchpoints



/nfs/netapp/u2/home/barryk/tests/arraysLinux - MAIN__
File
Edit
View
Tools
Window
Help
More
/nfs/netapp/u2/home/barryk/tests/arraysLinux - MAIN__
Watchpoint Properties
Processes...
OK
Delete
Cancel
Help

1.1

Expression: comp_array_1__ Address: 0xbfffdd00

Slice: (:) Filter:

Type: complex(100)

| Variables | Value |
|---|---|
| (7) | 2.07257270812988 |
| (8) | 0 |
| (9) | 2.07257270812988 |
| (10) | |
| (11) | |
| (12) | |
| (13) | |
| (14) | |
| (15) | |
| (16) | |
| (17) | |

1.1

Expression: comp_array_1__(9) Address: 0xbfffdd40

Type: complex

2.07257270812988

● Unconditional ○ Conditional ID: <new>

When Hit, Stop

● Group

○ Process

○ Thread

Address: 0xbfffdd40 Length in Bytes: 8

☑ Enable watchpoint

☑ Plant in share group

Watchpoints are set on a fixed memory region

Use *Tools > Watchpoint* from a Variable Window or

From source pane with contextual menu

When the contents of watched memory change, the watchpoint is triggered and TotalView stops the program.

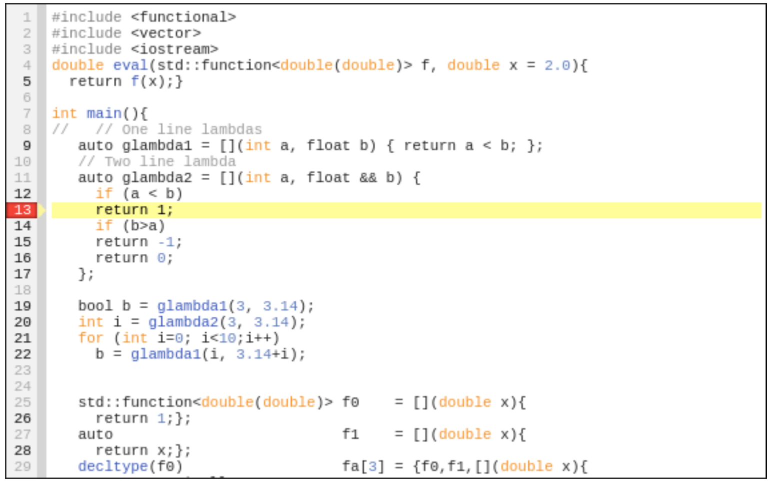
Watchpoints are not set on a variable. You you need to be aware of the variable scope.

Watchpoints can be conditional or unconditional

Uses Hardware Watchpoints with various limitations based on architecture

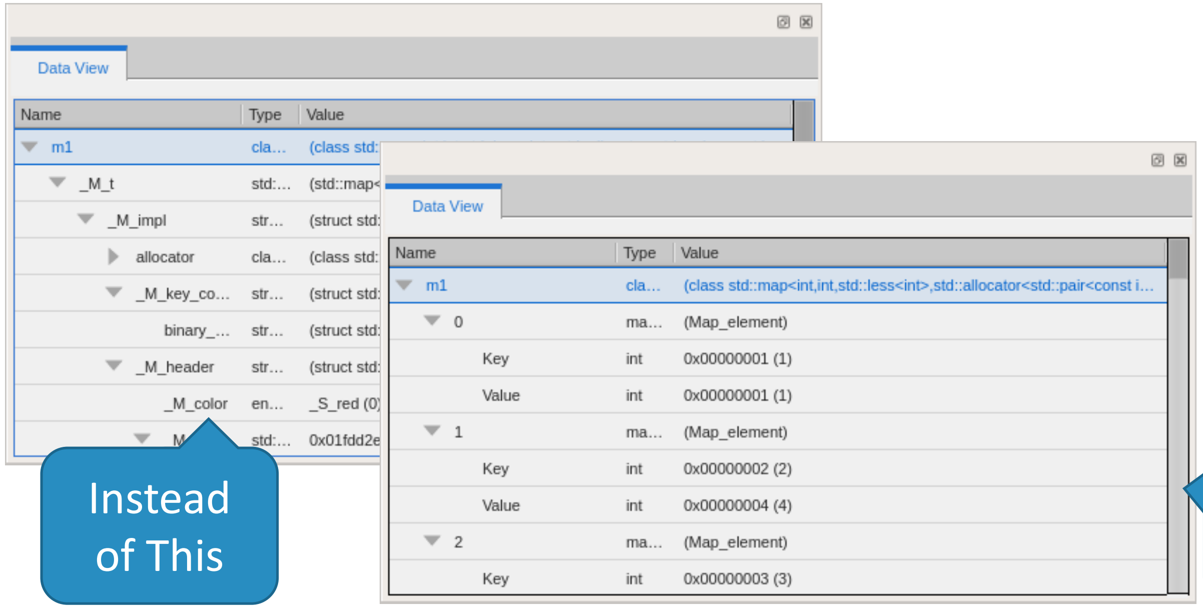
# Advanced C++ and Data Debugging

# Advanced C++ and Data Debugging



```
#include <functional>
#include <vector>
#include <iostream>
double eval(std::function<double(double)> f, double x = 2.0){
  return f(x);}

int main(){
//   // One line lambdas
   auto glambda1 = [](int a, float b) { return a < b; };
   // Two line lambda
   auto glambda2 = [](int a, float && b) {
     if (a < b)
     return 1;
     if (b>a)
     return -1;
     return 0;
   };

   bool b = glambda1(3, 3.14);
   int i = glambda2(3, 3.14);
   for (int i=0; i<10;i++)
     b = glambda1(i, 3.14+i);


   std::function<double(double)> f0    = [](double x){
     return 1;};
   auto                          f1    = [](double x){
     return x;};
   decltype(f0)                  fa[3] = {f0,f1,[](double x){
```

- TotalView supports debugging the latest C++11/14 features including:
  - lambdas, transformations for smart pointers, auto types, R-Value references, range-based loops, strongly-typed enums, initializer lists, user defined literals



Data View

| Name | Type | Value |
|---|---|---|
| m1 | cla... | (class std: |
| _M_t | std:... | (std::map< |
| _M_impl | str... | (struct std: |
| allocator | cla... | (class std: |
| _M_key_co... | str... | (struct std: |
| binary_... | str... | (struct std: |
| _M_header | str... | (struct std: |
| _M_color | en... | _S_red (0) |
| M | std:... | 0x01fdd2e |

Instead of This

Data View

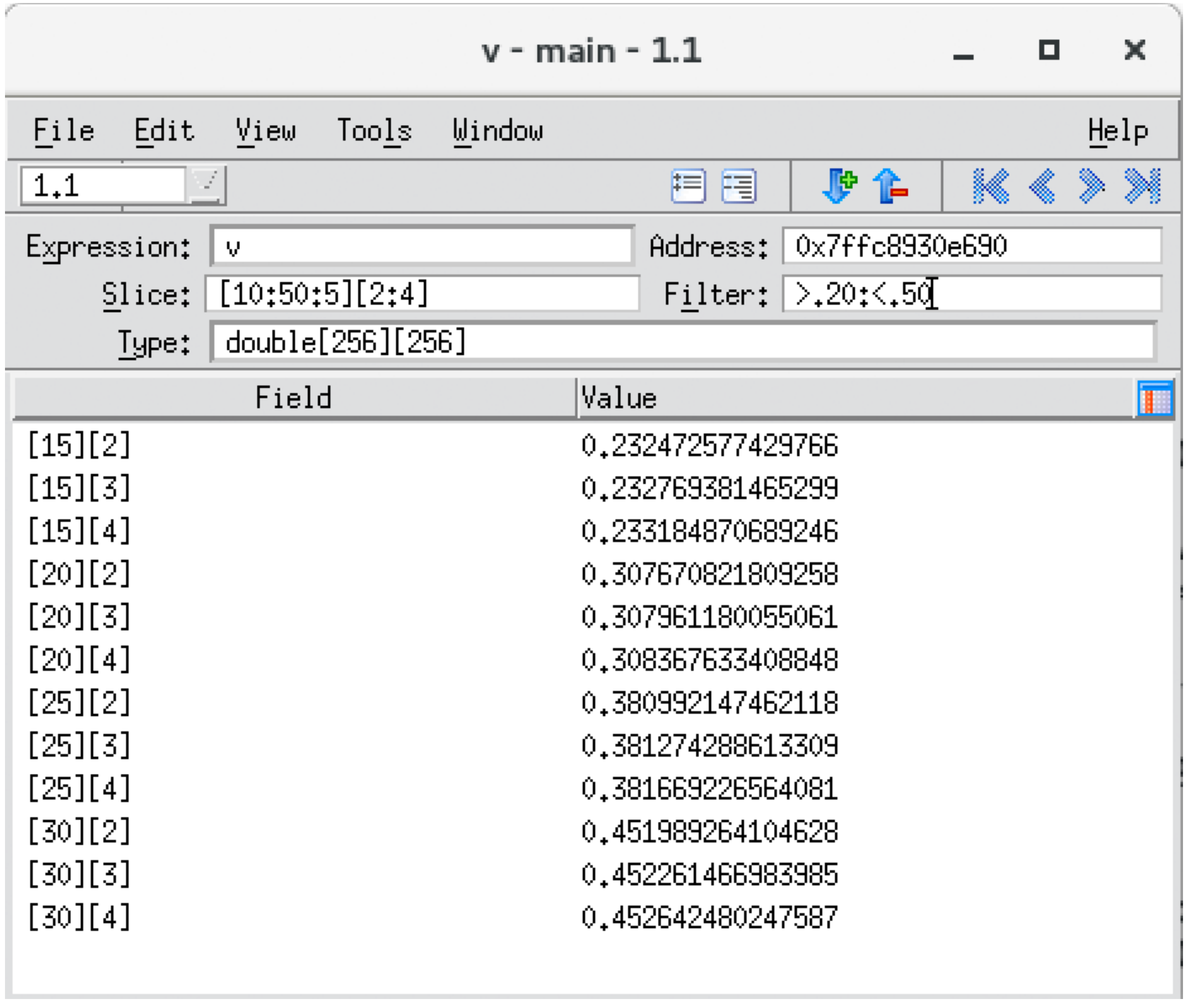
| Name | Type | Value |
|---|---|---|
| m1 | cla... | (class std::map<int,int,std::less<int>,std::allocator<std::pair<const i... |
| 0 | ma... | (Map_element) |
| Key | int | 0x00000001 (1) |
| Value | int | 0x00000001 (1) |
| 1 | ma... | (Map_element) |
| Key | int | 0x00000002 (2) |
| Value | int | 0x00000004 (4) |
| 2 | ma... | (Map_element) |
| Key | int | 0x00000003 (3) |

See This!

- TotalView transforms many of the C++ and STL containers such as:
  - array, forward_list, tuple, map, set, vector and others.

# Array Slicing, Striding and Filtering (classic UI)

- Slicing – reduce display to a portion of the array
  - [lower_bound:upper_bound]
  - [5:10]
- Striding – Skip over elements
  - [::stride]
  - [::5], [5:10:-1]
  - Filtering
    - Comparison: ==, !=, <, <=, >, >=
    - Range of values: [**>**] *low-value* **:** [**<**] *high-value*
    - IEEE values: $nan, $inf, $denorm

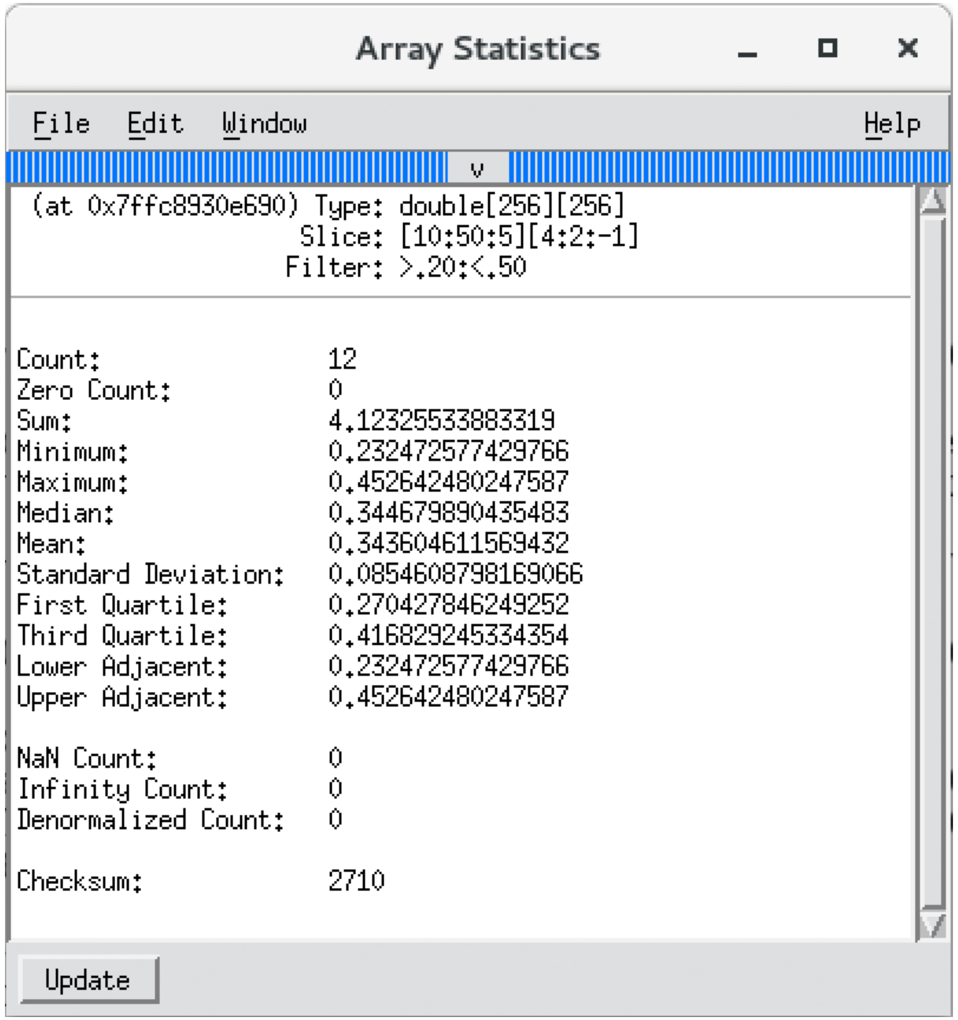


Expression: v
Address: 0x7ffc8930e690
Slice: [10:50:5][2:4]
Filter: >.20:<.50
Type: double[256][256]

| Field | Value |
|---|---|
| [15][2] | 0.232472577429766 |
| [15][3] | 0.232769381465299 |
| [15][4] | 0.233184870689246 |
| [20][2] | 0.307670821809258 |
| [20][3] | 0.307961180055061 |
| [20][4] | 0.308367633408848 |
| [25][2] | 0.380992147462118 |
| [25][3] | 0.381274288613309 |
| [25][4] | 0.381669226564081 |
| [30][2] | 0.451989264104628 |
| [30][3] | 0.452261466983985 |
| [30][4] | 0.452642480247587 |

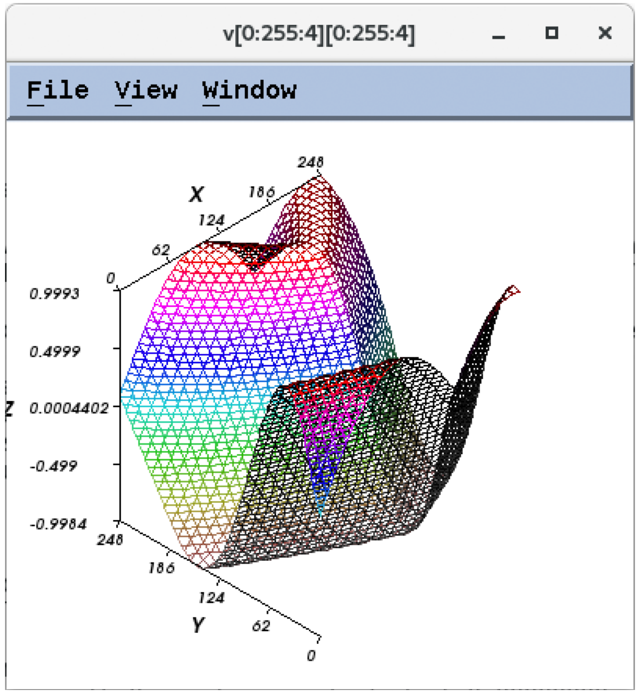
# Array Statistics

- Easily display a set of statistics for the filtered portion of your array

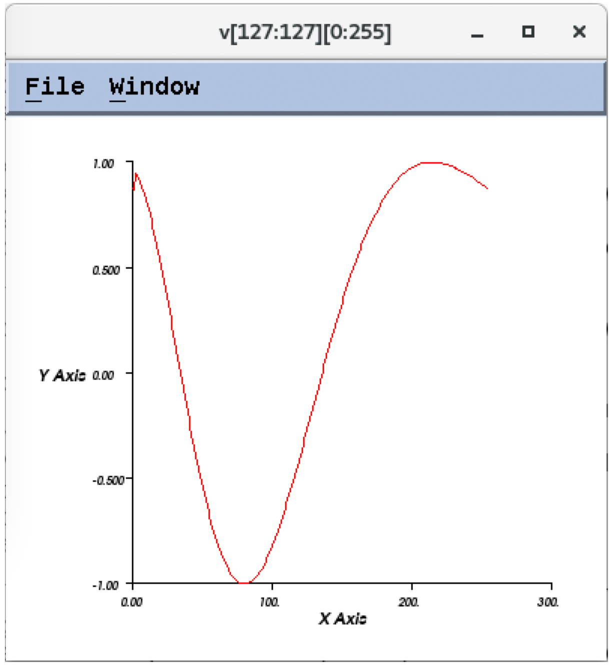


```
(at 0x7ffc8930e690) Type: double[256][256]
                   Slice: [10:50:5][4:2:-1]
                  Filter: >.20:<.50

Count:                 12
Zero Count:            0
Sum:                   4.12325533883319
Minimum:               0.232472577429766
Maximum:               0.452642480247587
Median:                0.344679890435483
Mean:                  0.343604611569432
Standard Deviation:    0.0854608798169066
First Quartile:        0.270427846249252
Third Quartile:        0.416829245334354
Lower Adjacent:        0.232472577429766
Upper Adjacent:        0.452642480247587

NaN Count:             0
Infinity Count:        0
Denormalized Count:    0

Checksum:              2710
```

# Visualizing Array Data

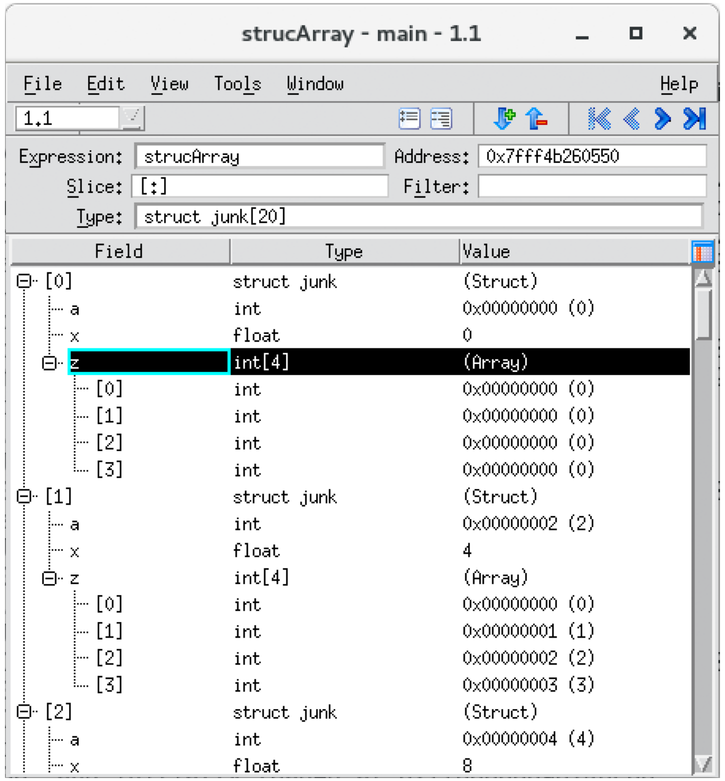


- Visualizer creates graphic images of your program’s array data.
- Visualize one or two dimensional arrays
- View data manually through the Window > Visualize command on the Data Window
- Visualize data programmatically using the $visualize function



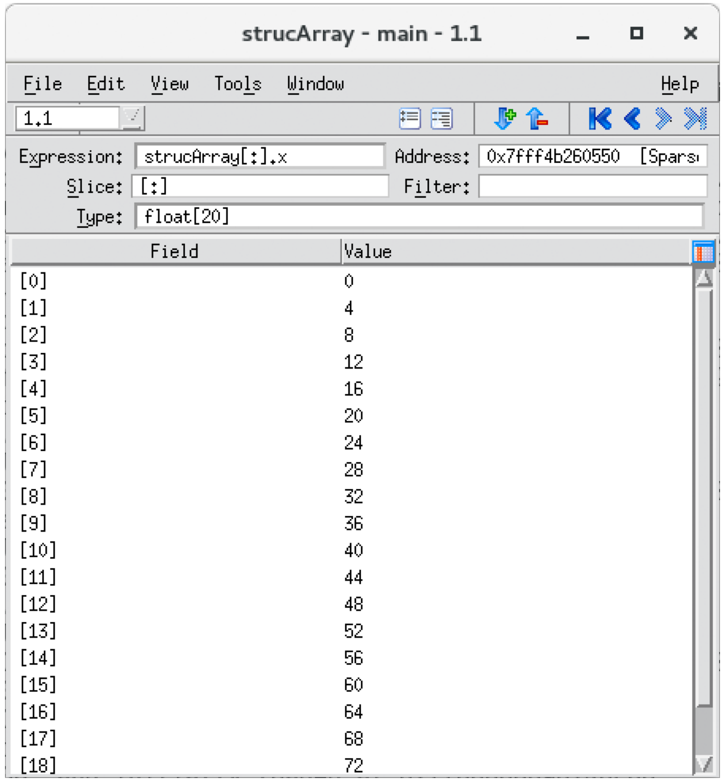
# Dive in All

- Dive in All
  - Use Dive in All to easily see each member of a data structure from an array of structures



Expression: strucArray Address: 0x7fff4b260550
Slice: [:] Filter:
Type: struct junk[20]

| Field | Type | Value |
|---|---|---|
| [0] | struct junk | (Struct) |
| a | int | 0x00000000 (0) |
| x | float | 0 |
| z | int[4] | (Array) |
| [0] | int | 0x00000000 (0) |
| [1] | int | 0x00000000 (0) |
| [2] | int | 0x00000000 (0) |
| [3] | int | 0x00000000 (0) |
| [1] | struct junk | (Struct) |
| a | int | 0x00000002 (2) |
| x | float | 4 |
| z | int[4] | (Array) |
| [0] | int | 0x00000000 (0) |
| [1] | int | 0x00000001 (1) |
| [2] | int | 0x00000002 (2) |
| [3] | int | 0x00000003 (3) |
| [2] | struct junk | (Struct) |
| a | int | 0x00000004 (4) |
| x | float | 8 |

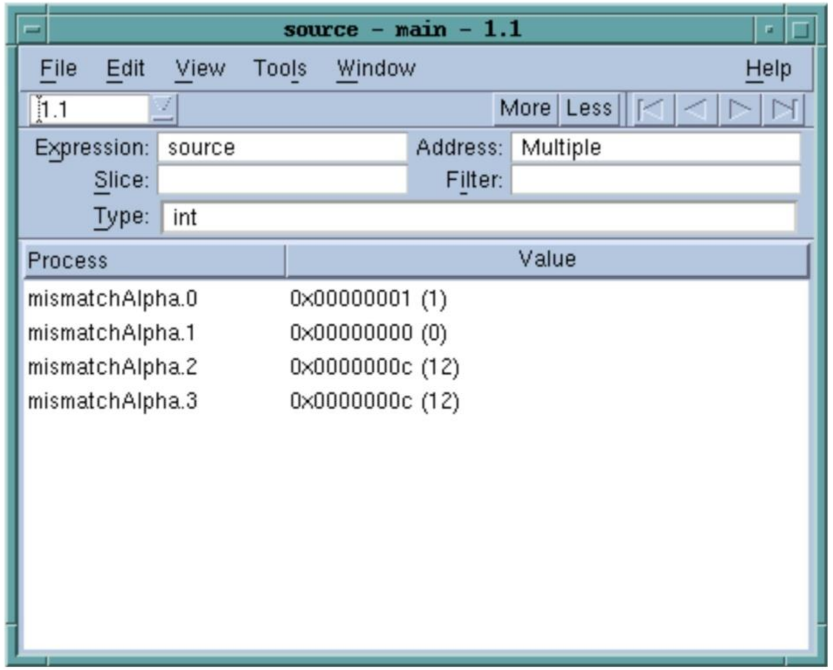


Expression: strucArray[:].x Address: 0x7fff4b260550 [Spars
Slice: [:] Filter:
Type: float[20]

| Field | Value |
|---|---|
| [0] | 0 |
| [1] | 4 |
| [2] | 8 |
| [3] | 12 |
| [4] | 16 |
| [5] | 20 |
| [6] | 24 |
| [7] | 28 |
| [8] | 32 |
| [9] | 36 |
| [10] | 40 |
| [11] | 44 |
| [12] | 48 |
| [13] | 52 |
| [14] | 56 |
| [15] | 60 |
| [16] | 64 |
| [17] | 68 |
| [18] | 72 |

# Looking at Variables Across Processes

- TotalView allows you to look at the value of a variable in all MPI processes
  - Right Click on the variable
  - Select the View > View Across
- TotalView creates an array indexed by process
- You can filter and visualize
- Use for viewing distributed arrays as well.
- You can also View Across Threads



1.1

Expression: source
Address: Multiple
Slice:
Filter:
Type: int

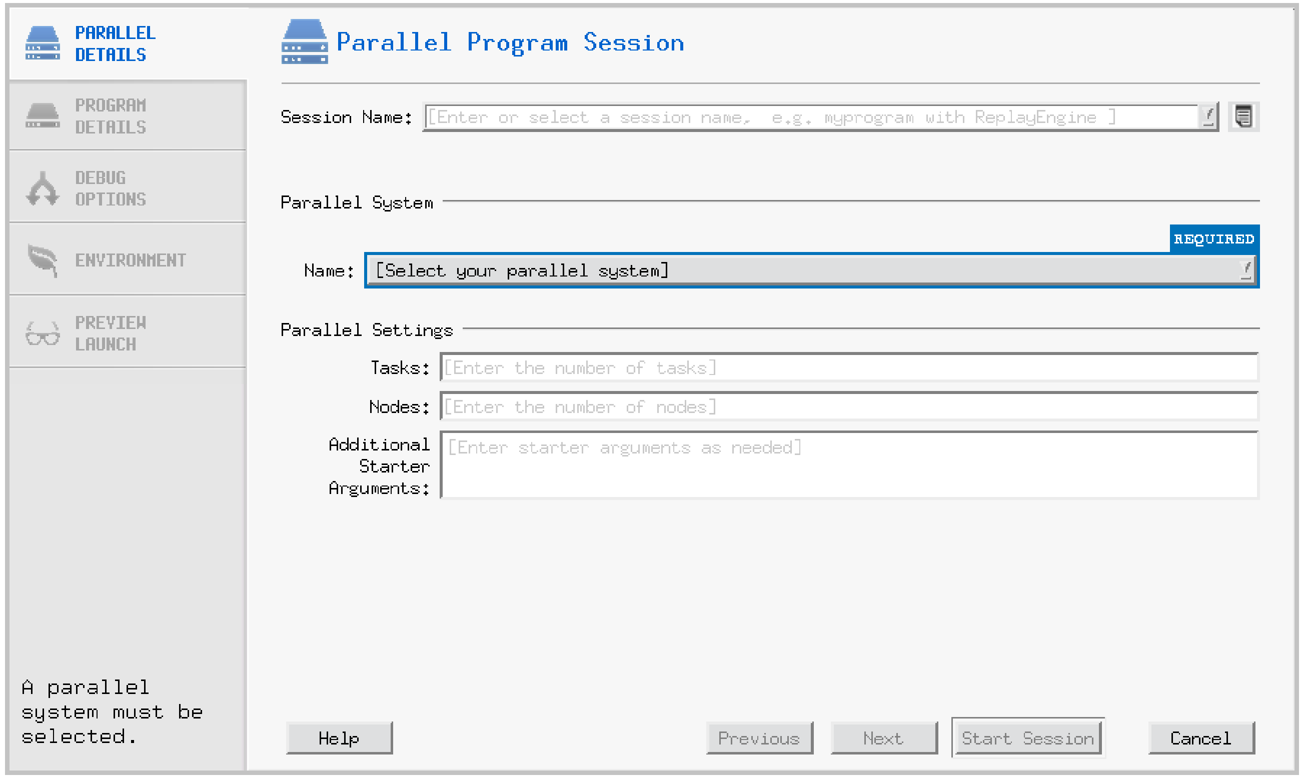
| Process | Value |
|---|---|
| mismatchAlpha.0 | 0x00000001 (1) |
| mismatchAlpha.1 | 0x00000000 (0) |
| mismatchAlpha.2 | 0x0000000c (12) |
| mismatchAlpha.3 | 0x0000000c (12) |

# Multi-Thread and Multi-Process Parallel Debugging

# In the Parallel Program Session select:

Select:

- MPI preference
- number of tasks
- number of nodes
- starter arguments



A parallel system must be selected.

Parallel Program Session

Session Name: [Enter or select a session name, e.g. myprogram with ReplayEngine ]

Parallel System

REQUIRED

Name: [Select your parallel system]

Parallel Settings

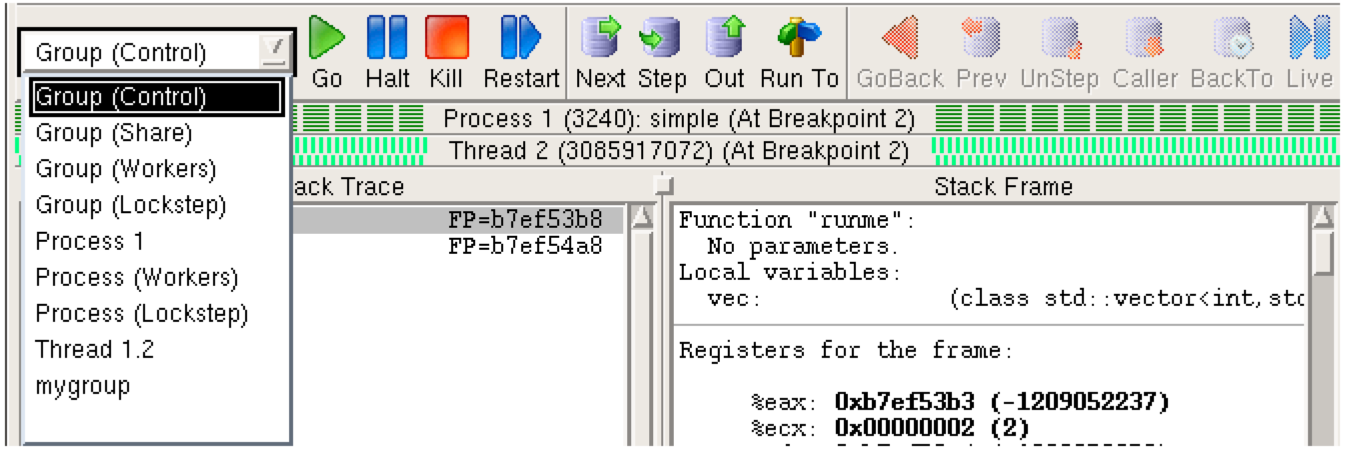
Tasks: [Enter the number of tasks]

Nodes: [Enter the number of nodes]

Additional Starter Arguments: [Enter starter arguments as needed]

… then save all this in Session

# Stepping Commands

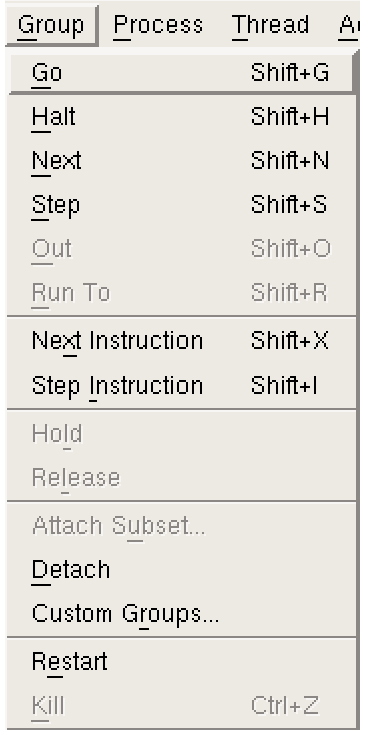


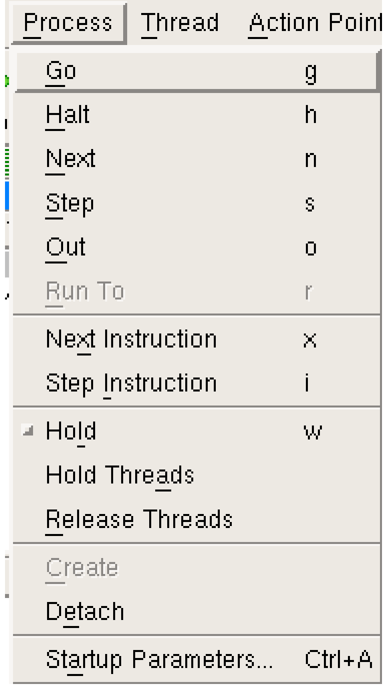
Process 1 (3240): simple (At Breakpoint 2)

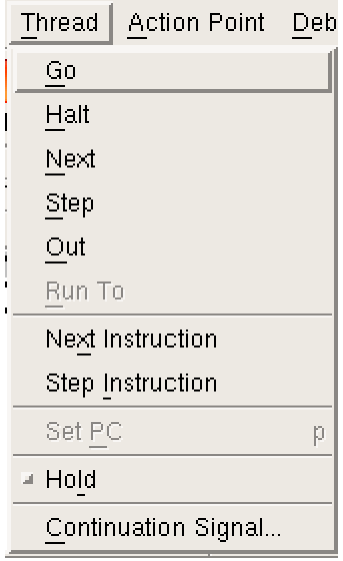
Thread 2 (3085917072) (At Breakpoint 2)

```
FP=b7ef53b8
FP=b7ef54a8
```

```
Function "runme":
  No parameters.
Local variables:
  vec:                (class std::vector<int,std

Registers for the frame:

     %eax:  0xb7ef53b3 (-1209052237)
     %ecx:  0x00000002 (2)
```

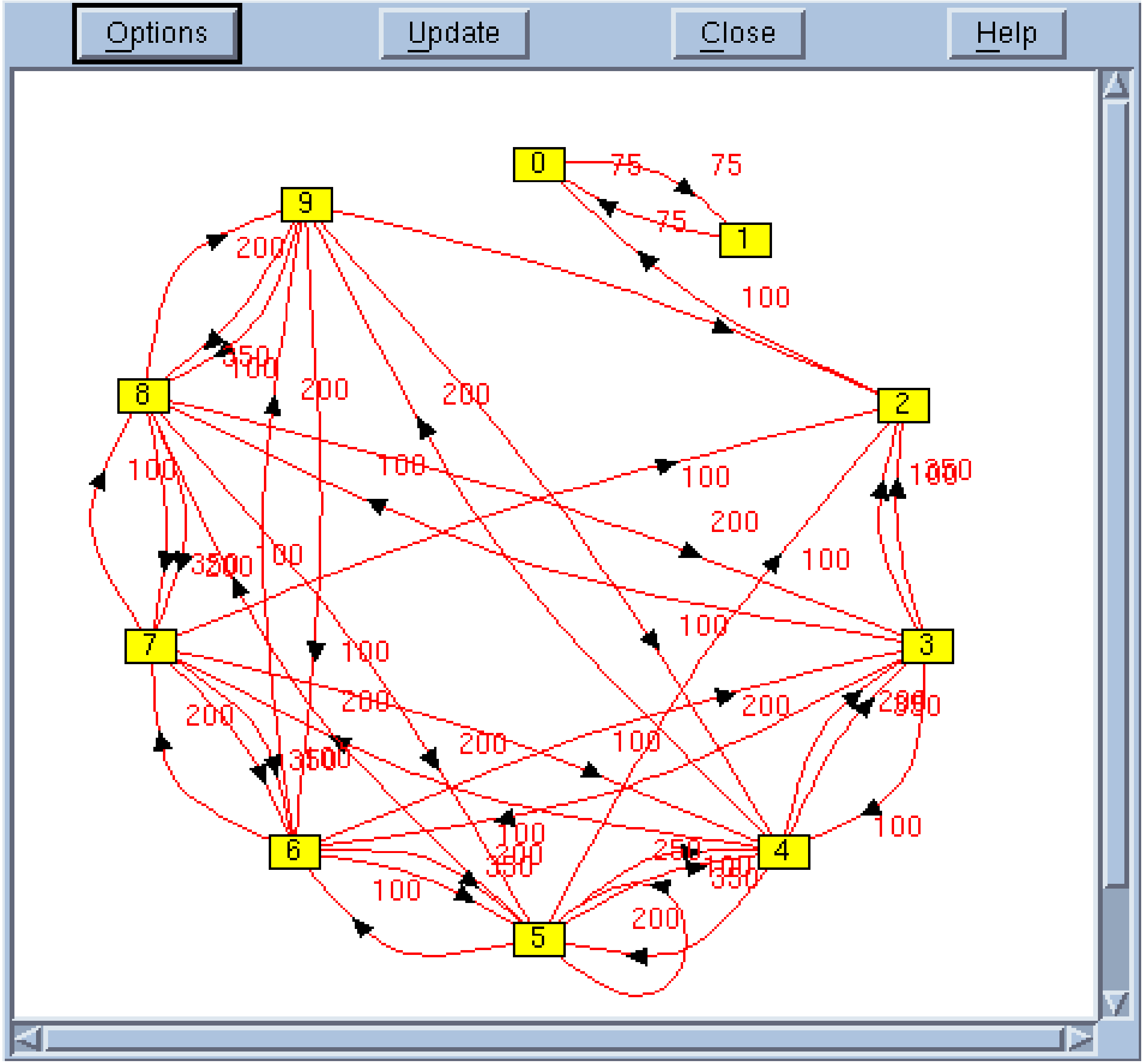






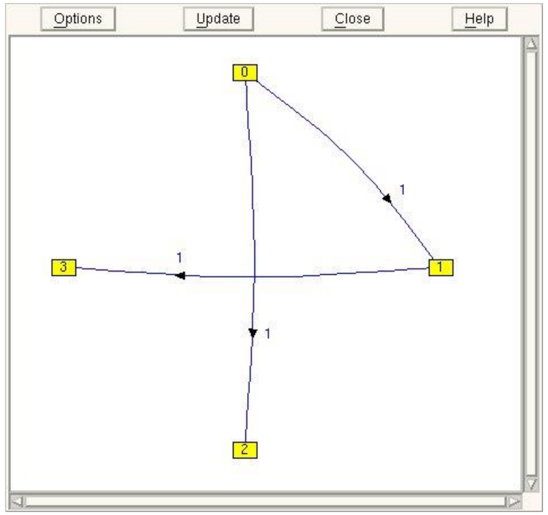
# Message Queue Graph

- Hangs & Deadlocks
- Pending Messages
  - Receives
  - Sends
  - Unexpected
- Inspect
  - Individual entries
- Patterns

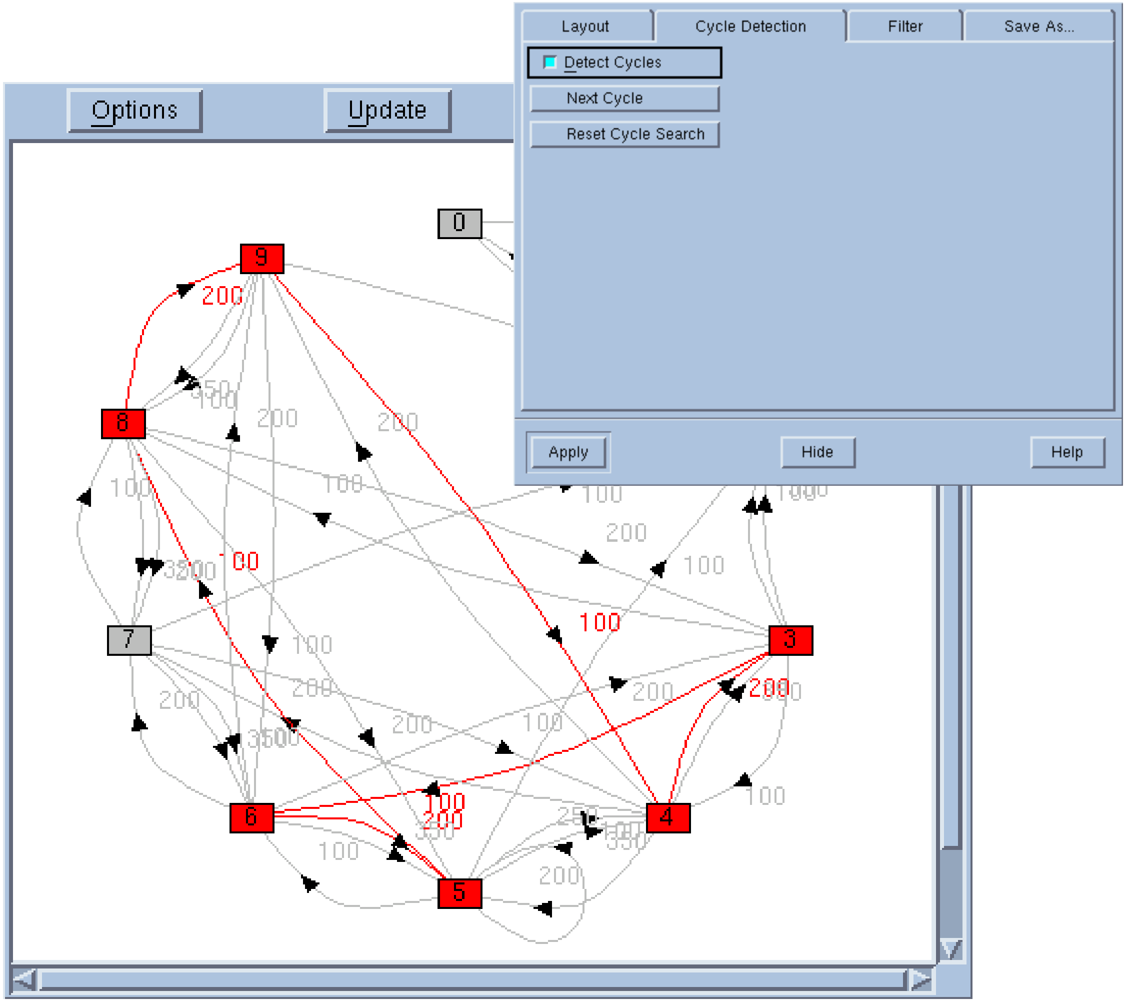


# Find Deadlocks and Performance Sinks

- Filtering
  - Choose messages to track
  - Choose MPI Communicators
- Cycle detection →



← Sink



# Reverse Debugging

# Reverse Debugging

## Replay Engine – The right way to debug

Step forward over functions

Step *backward* over functions

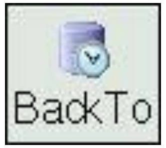
Step forward into functions

Step *backward* into functions

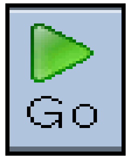
Advance forward out of current
Function, after the call

Advance backward out of current
Function, to before the call

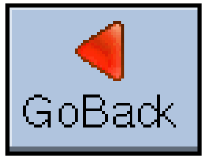
Advance forward to selected line



Advance backward to selected line



Run forward



Run *backward*

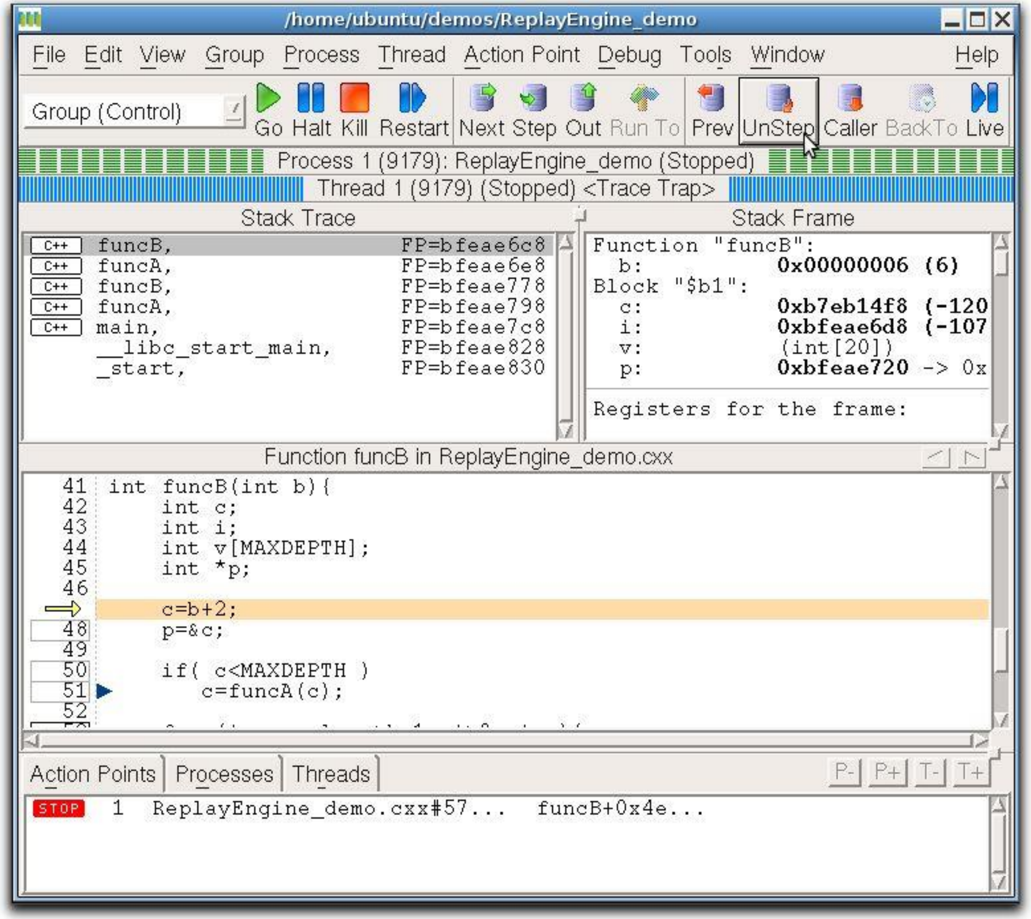
Advance forward to “live” session

# ReplayEngine

- **Captures execution history**
  - Records all external input to program
  - Records internal sources of non-determinism
- **Replays execution history**
  - Examine any part of the execution history
  - Step back as easily as forward
  - Jump to points of interest
- **An add-on product to TotalView**
  - Support for
    - Linux/x86
    - Linux x86- 64

Next

Step


/home/ubuntu/demos/ReplayEngine_demo
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
Stack Trace
Stack Frame
Function funcB in ReplayEngine_demo.cxx
Action Points Processes Threads
P- P+ T- T+

Process 1 (9179): ReplayEngine_demo (Stopped)

Thread 1 (9179) (Stopped) <Trace Trap>

```
C++  funcB,                 FP=bfeae6c8
C++  funcA,                 FP=bfeae6e8
C++  funcB,                 FP=bfeae778
C++  funcA,                 FP=bfeae798
C++  main,                  FP=bfeae7c8
     __libc_start_main,     FP=bfeae828
     _start,                FP=bfeae830
```

```
Function "funcB":
  b:                0x00000006 (6)
Block "$b1":
  c:                0xb7eb14f8 (-120
  i:                0xbfeae6d8 (-107
  v:                (int[20])
  p:                0xbfeae720 -> 0x

Registers for the frame:
```

```
int funcB(int b){
    int c;
    int i;
    int v[MAXDEPTH];
    int *p;

    c=b+2;
    p=&c;

    if( c<MAXDEPTH )
       c=funcA(c);
```

```
STOP  1  ReplayEngine_demo.cxx#57...   funcB+0x4e...
```

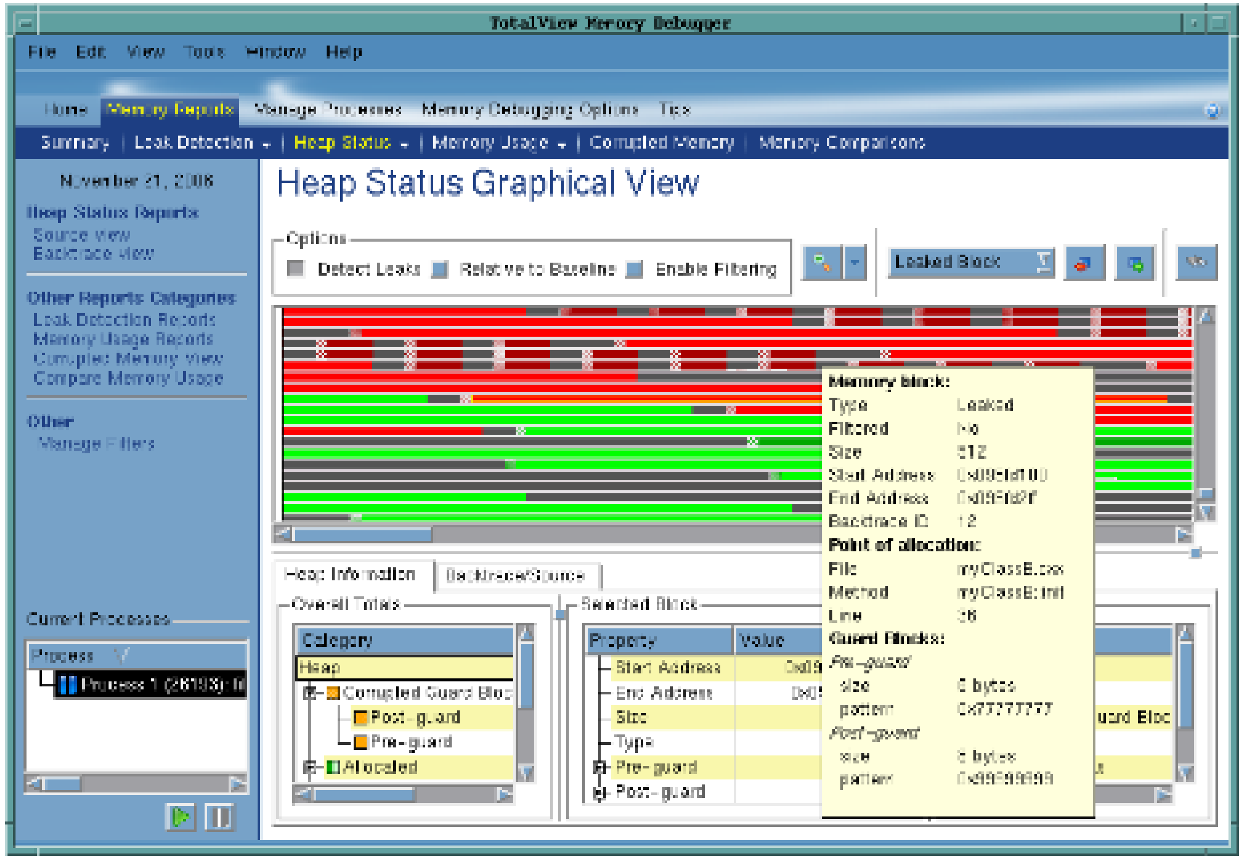
Prev

UnStep

# Memory Debugging

# Memory Debugging

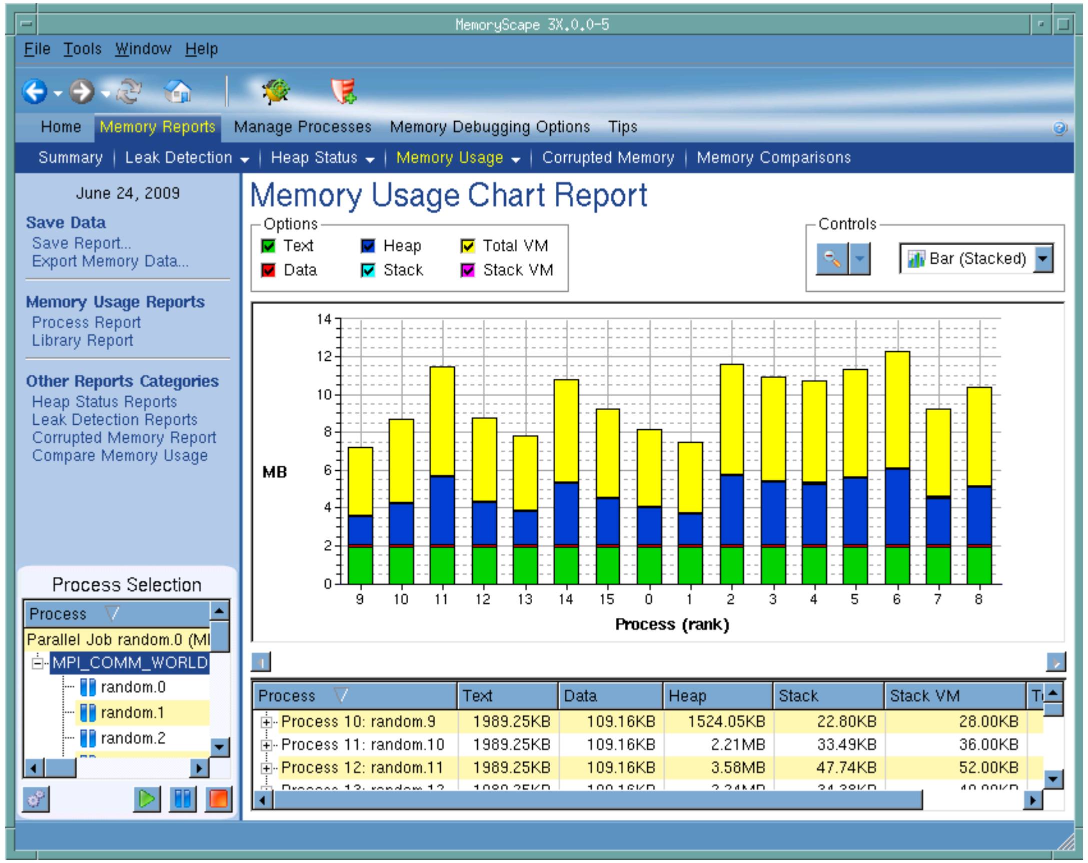
- TotalView's memory debugging technology allows you to
  - Easily find memory leaks and other memory errors
  - Detect malloc/free new/delete API misuse
  - Dangling pointer detection
  - Detect buffer overruns
  - Paint memory blocks on allocation and deallocation
- Memory debugging results can be easily shared as
  - HTML reports or raw memory debugging files.
- Compare memory results between runs to verify elimination of leaks
- Supports parallel applications
- Low overhead and does not require recompilation or instrumentation



**Memory block:**
Type Leaked
Filtered No
Size 512
Start Address 0x095fd100
End Address 0x095fd2ff
Backtrace ID 12
**Point of allocation:**
File myClassE.cxx
Method myClassE::init
Line 36
**Guard Blocks:**
*Pre-guard*
size 8 bytes
pattern 0x77777777
*Post-guard*
size 8 bytes
pattern 0x99999999

# Strategies for Parallel Memory Debugging

- Run the application and see if memory events are detected
- View memory usage across the MPI job
  - Compare memory footprint of the processes
    - Are there any outliers? Are they expected?
- Gather heap information in all processes of the MPI job
  - Select and examine individually
    - Look at the allocation pattern.
      Does it make sense?
    - Look for leaks
  - Compare with the 'diff' mechanism
    - Are there any major differences?
      Are they expected?



Memory Usage Chart Report

Options: ☑ Text ☑ Data ☑ Heap ☑ Stack ☑ Total VM ☑ Stack VM

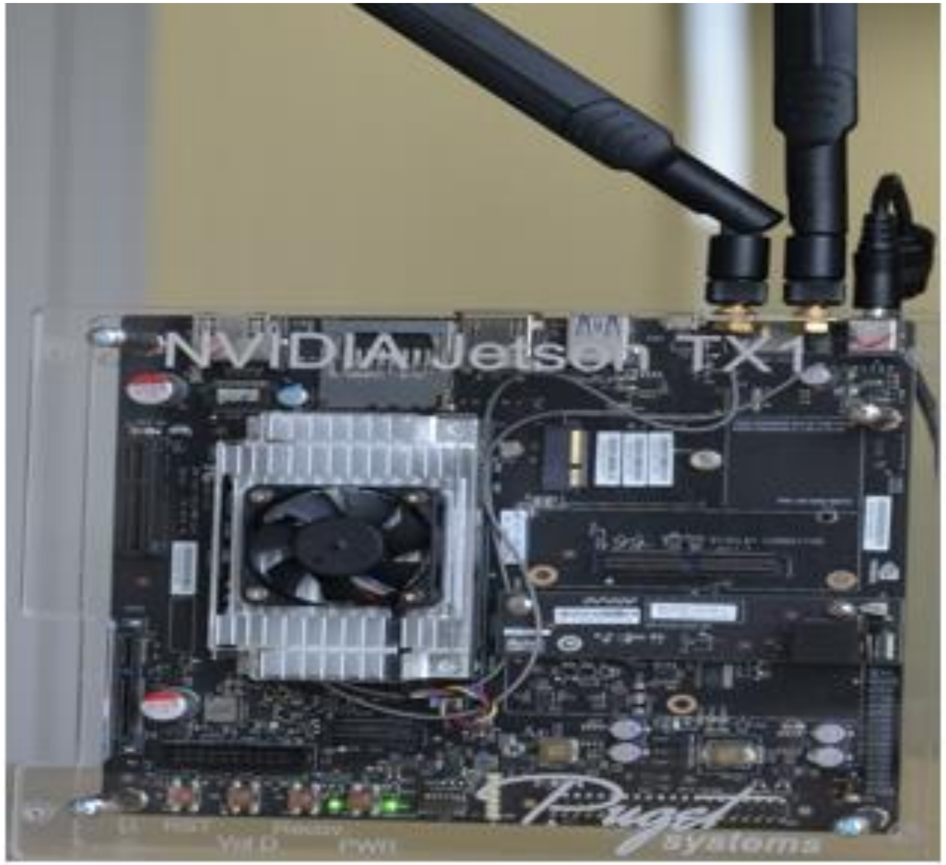
Controls: Bar (Stacked)

| Process | Text | Data | Heap | Stack | Stack VM |
|---|---|---|---|---|---|
| Process 10: random.9 | 1989.25KB | 109.16KB | 1524.05KB | 22.80KB | 28.00KB |
| Process 11: random.10 | 1989.25KB | 109.16KB | 2.21MB | 33.49KB | 36.00KB |
| Process 12: random.11 | 1989.25KB | 109.16KB | 3.58MB | 47.74KB | 52.00KB |

# GPU Debugging

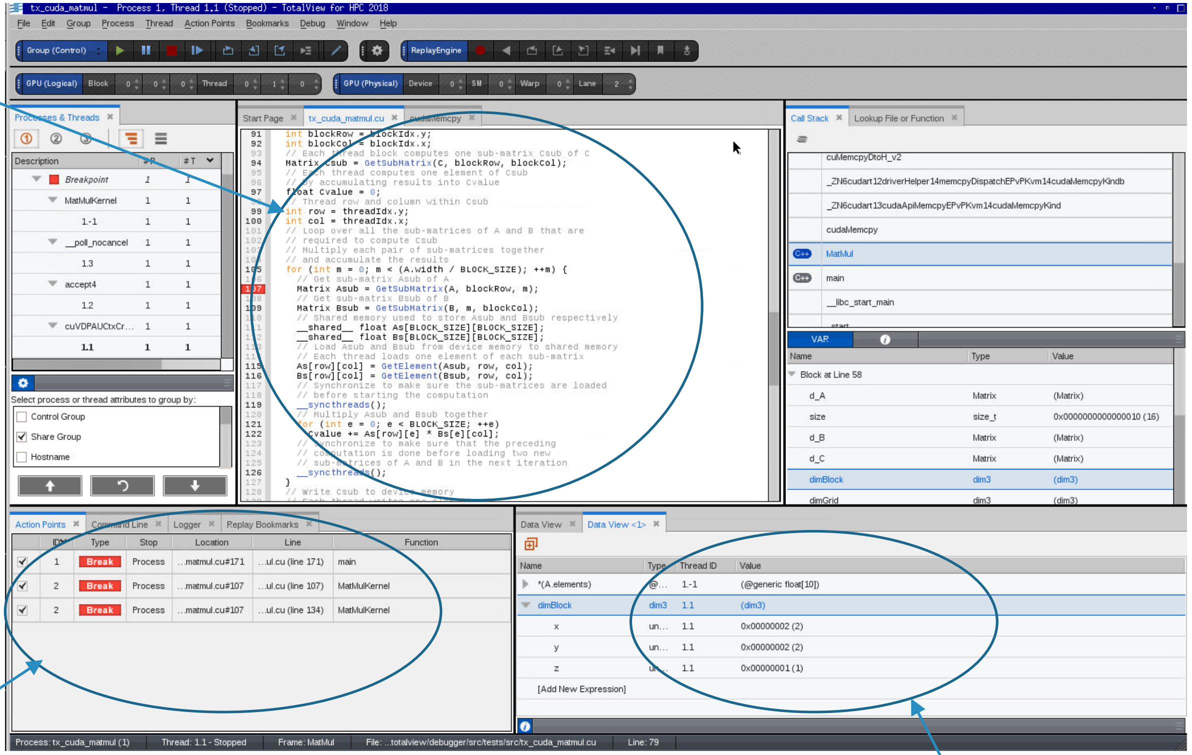
# GPU debugging with TotalView

- NVIDIA CUDA support
  - Multiple platforms : X86-64,PowerLE, ARM64
  - Multiple cards:  from Jetson to Turing
- Features and capabilities include
  - Support for dynamic parallelism
  - Support for MPI based clusters and multi-card configurations
  - Flexible Display and Navigation on the CUDA device
    - Physical (device, SM, Warp, Lane)
    - Logical (Grid, Block) tuples
  - CUDA device window reveals what is running where
  - Support for CUDA Core debugging
  - Leverages CUDA memcheck
  - Support for OpenACC



# GPU Debugging Model Improvements

- First in class Unified Source debugging
- Improves and streamlines debugging CUDA applications



| Description | #P | #T |
|---|---|---|
| Breakpoint | 1 | 1 |
| MatMulKernel | 1 | 1 |
| 1.-1 | 1 | 1 |
| __poll_nocancel | 1 | 1 |
| 1.3 | 1 | 1 |
| accept4 | 1 | 1 |
| 1.2 | 1 | 1 |
| cuVDPAUCtxCr… | 1 | 1 |
| **1.1** | **1** | **1** |

Select process or thread attributes to group by:

- ☐ Control Group
- ☑ Share Group
- ☐ Hostname

```
    int blockRow = blockIdx.y;
    int blockCol = blockIdx.x;
    // Each thread block computes one sub-matrix Csub of C
    Matrix Csub = GetSubMatrix(C, blockRow, blockCol);
    // Each thread computes one element of Csub
    // by accumulating results into Cvalue
    float Cvalue = 0;
    // Thread row and column within Csub
    int row = threadIdx.y;
    int col = threadIdx.x;
    // Loop over all the sub-matrices of A and B that are
    // required to compute Csub
    // Multiply each pair of sub-matrices together
    // and accumulate the results
    for (int m = 0; m < (A.width / BLOCK_SIZE); ++m) {
      // Get sub-matrix Asub of A
      Matrix Asub = GetSubMatrix(A, blockRow, m);
      // Get sub-matrix Bsub of B
      Matrix Bsub = GetSubMatrix(B, m, blockCol);
      // Shared memory used to store Asub and Bsub respectively
      __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
      __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
      // Load Asub and Bsub from device memory to shared memory
      // Each thread loads one element of each sub-matrix
      As[row][col] = GetElement(Asub, row, col);
      Bs[row][col] = GetElement(Bsub, row, col);
      // Synchronize to make sure the sub-matrices are loaded
      // before starting the computation
      __syncthreads();
      // Multiply Asub and Bsub together
      for (int e = 0; e < BLOCK_SIZE; ++e)
        Cvalue += As[row][e] * Bs[e][col];
      // Synchronize to make sure that the preceding
      // computation is done before loading two new
      // sub-matrices of A and B in the next iteration
      __syncthreads();
    }
    // Write Csub to device memory
```

cuMemcpyDtoH_v2
_ZN6cudart12driverHelper14memcpyDispatchEPvPKvm14cudaMemcpyKindb
_ZN6cudart13cudaApiMemcpyEPvPKvm14cudaMemcpyKind
cudaMemcpy
C++ MatMul
C++ main
__libc_start_main
_start

| Name | Type | Value |
|---|---|---|
| Block at Line 58 | | |
| d_A | Matrix | (Matrix) |
| size | size_t | 0x0000000000000010 (16) |
| d_B | Matrix | (Matrix) |
| d_C | Matrix | (Matrix) |
| dimBlock | dim3 | (dim3) |
| dimGrid | dim3 | (dim3) |

| | ID | Type | Stop | Location | Line | Function |
|---|---|---|---|---|---|---|
| ☑ | 1 | Break | Process | …matmul.cu#171 | …ul.cu (line 171) | main |
| ☑ | 2 | Break | Process | …matmul.cu#107 | …ul.cu (line 107) | MatMulKernel |
| ☑ | 2 | Break | Process | …matmul.cu#107 | …ul.cu (line 134) | MatMulKernel |

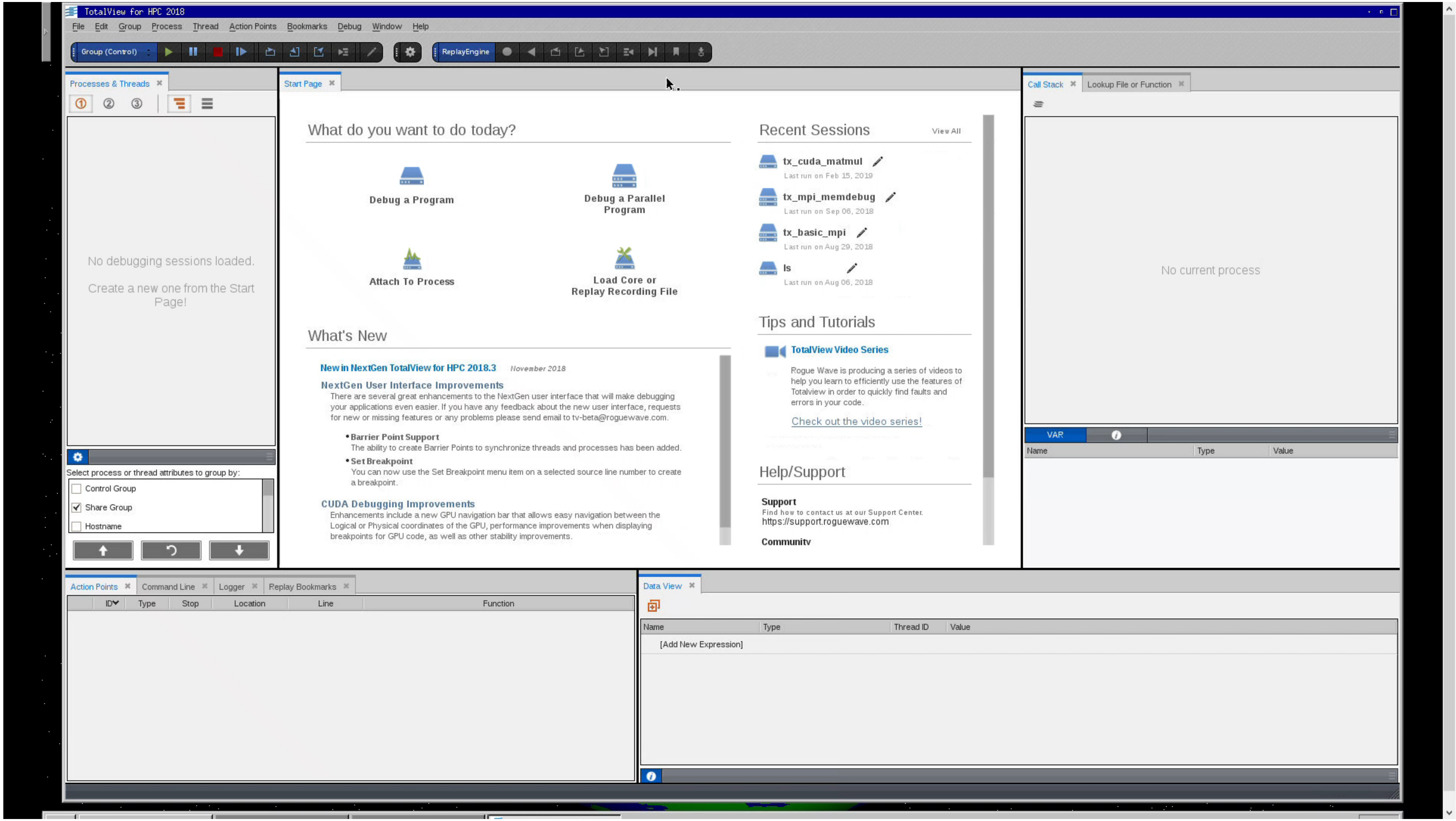
| Name | Type | Thread ID | Value |
|---|---|---|---|
| *(A.elements) | @… | 1.-1 | (@generic float[10]) |
| dimBlock | dim3 | 1.1 | (dim3) |
| x | un… | 1.1 | 0x00000002 (2) |
| y | un… | 1.1 | 0x00000002 (2) |
| z | un… | 1.1 | 0x00000001 (1) |
| [Add New Expression] | | | |

Process: tx_cuda_matmul (1) | Thread: 1.1 - Stopped | Frame: MatMul | File: ...totalview/debugger/src/tests/src/tx_cuda_matmul.cu | Line: 79

- Set breakpoints in CPU **and** GPU kernel code before it is launched on the GPU

- Compare variables in CPU and GPU code together

# CUDA Debugging Demo



No debugging sessions loaded.

Create a new one from the Start Page!

Select process or thread attributes to group by:

- ☐ Control Group
- ☑ Share Group
- ☐ Hostname

## What do you want to do today?

## Recent Sessions

tx_cuda_matmul
Last run on Feb 15, 2019

tx_mpi_memdebug
Last run on Sep 06, 2018

tx_basic_mpi
Last run on Aug 29, 2018

ls
Last run on Aug 06, 2018

## What's New

**New in NextGen TotalView for HPC 2018.3** *November 2018*

**NextGen User Interface Improvements**

There are several great enhancements to the NextGen user interface that will make debugging your applications even easier. If you have any feedback about the new user interface, requests for new or missing features or any problems please send email to tv-beta@roguewave.com.

- **Barrier Point Support**
  The ability to create Barrier Points to synchronize threads and processes has been added.
- **Set Breakpoint**
  You can now use the Set Breakpoint menu item on a selected source line number to create a breakpoint.

**CUDA Debugging Improvements**

Enhancements include a new GPU navigation bar that allows easy navigation between the Logical or Physical coordinates of the GPU, performance improvements when displaying breakpoints for GPU code, as well as other stability improvements.

## Tips and Tutorials

**TotalView Video Series**

Rogue Wave is producing a series of videos to help you learn to efficiently use the features of Totalview in order to quickly find faults and errors in your code.

Check out the video series!

## Help/Support

**Support**
Find how to contact us at our Support Center.
https://support.roguewave.com

**Community**

No current process

| Name | Type | Value |
|---|---|---|

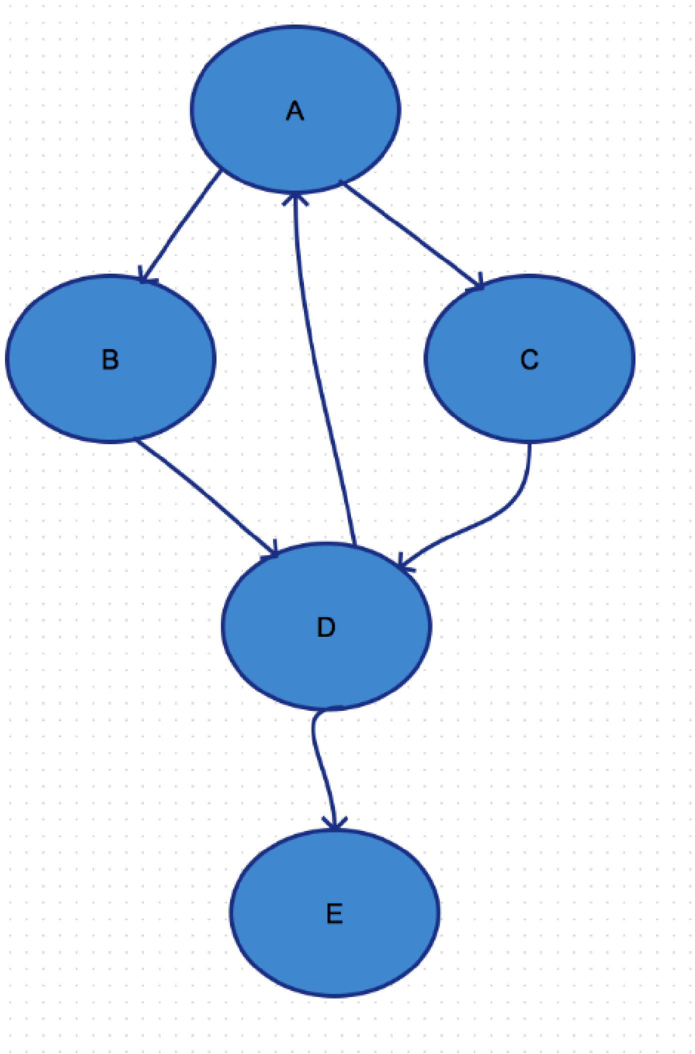
| ID | Type | Stop | Location | Line | Function |
|---|---|---|---|---|---|

| Name | Type | Thread ID | Value |
|---|---|---|---|
| [Add New Expression] | | | |

# Extending Debugging Capabilities:
# How to Debug  (AI) Mixed Python/C++ Code

# Debugging multiple languages

- Debugging one language is difficult enough
  - Especially with many threads/processes
- The language intersection is tougher
  - Data comparison
  - Glue code
- Issues are:
  - Type mismatches
  - Extraneous stack frames



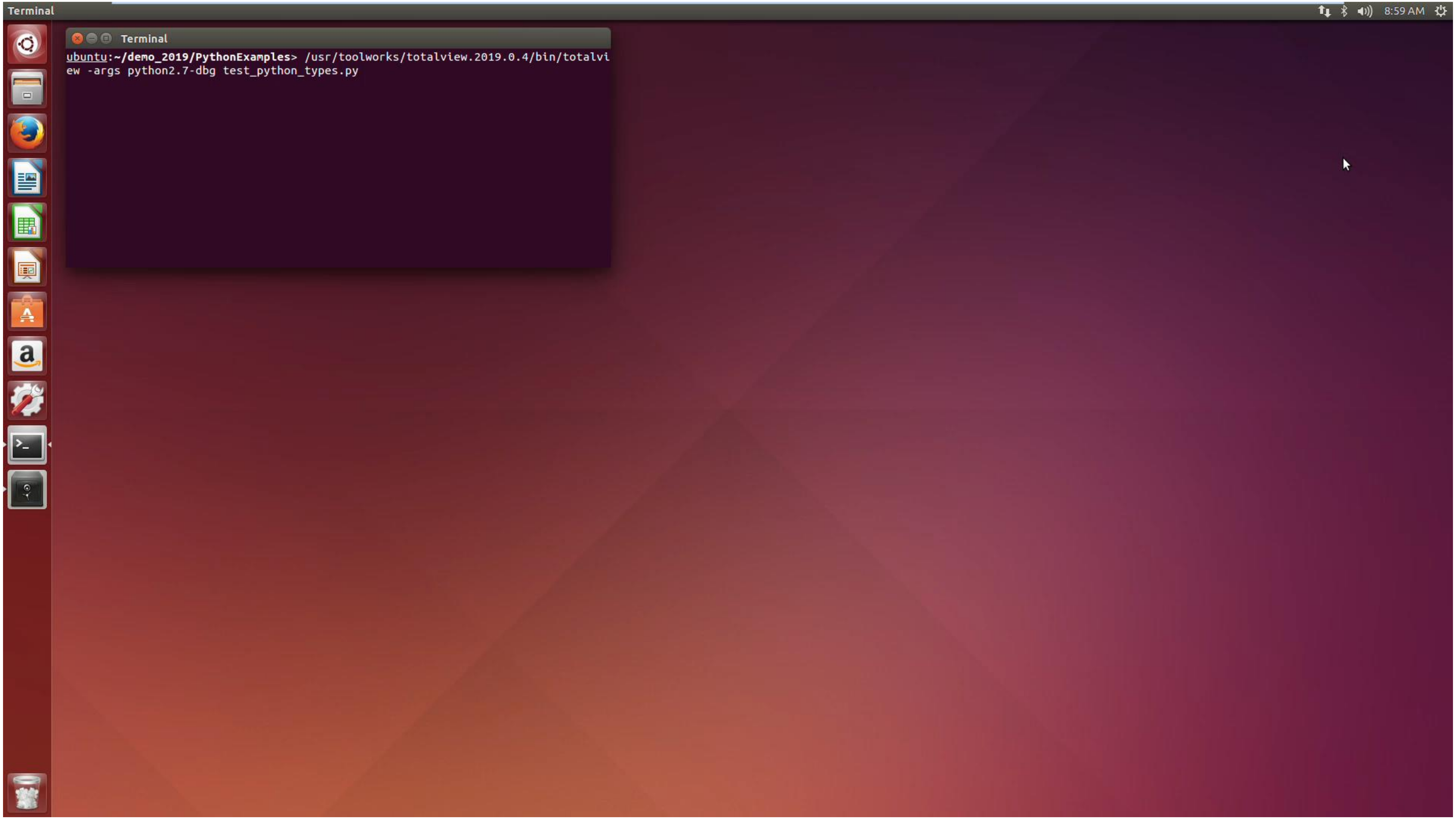
# Python debugging with TotalView (New GUI only)

- What TotalView provides:
  - Easy Python debugging session setup
  - Fully integrated Python and C/C++ call stack
    - ”Glue” layers between the languages removed
  - Easily examine and compare variables in Python and C++
  - Utilize reverse debugging and memory debugging

- What TotalView does not provide (yet):
  - Setting breakpoints and stepping within Python code

# Demo

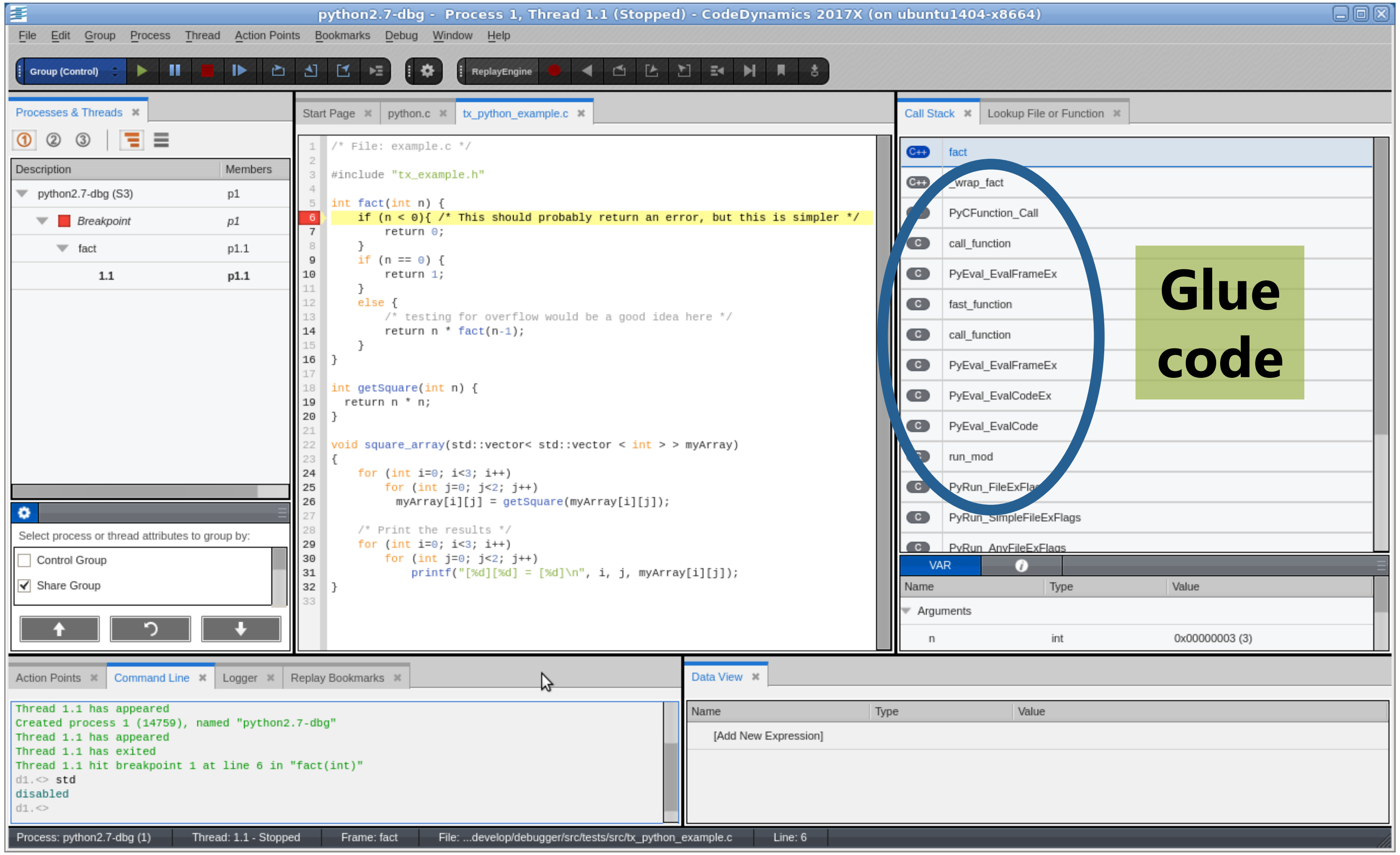
```
#!/usr/bin/python

def callFact():
  import tv_python_example as tp
  a = 3
  b = 10
  c = a+b
  ch = “local string”
   ……
  return tp.fact(a)
if __name__ == '__main__’:
  b = 2
  result = callFact()
  print result
```



```
ubuntu:~/demo_2019/PythonExamples> /usr/toolworks/totalview.2019.0.4/bin/totalvi
ew -args python2.7-dbg test_python_types.py
```

# Python without special debugger support

No viewing of Python data and code



| Description | Members |
|---|---|
| python2.7-dbg (S3) | p1 |
| Breakpoint | p1 |
| fact | p1.1 |
| 1.1 | p1.1 |

Select process or thread attributes to group by:

☐ Control Group

☑ Share Group

```
/* File: example.c */

#include "tx_example.h"

int fact(int n) {
    if (n < 0){ /* This should probably return an error, but this is simpler */
        return 0;
    }
    if (n == 0) {
        return 1;
    }
    else {
        /* testing for overflow would be a good idea here */
        return n * fact(n-1);
    }
}

int getSquare(int n) {
  return n * n;
}

void square_array(std::vector< std::vector < int > > myArray)
{
    for (int i=0; i<3; i++)
        for (int j=0; j<2; j++)
         myArray[i][j] = getSquare(myArray[i][j]);

    /* Print the results */
    for (int i=0; i<3; i++)
        for (int j=0; j<2; j++)
            printf("[%d][%d] = [%d]\n", i, j, myArray[i][j]);
}
```

Call Stack:

```
C++ fact
C++ _wrap_fact
    PyCFunction_Call
C   call_function
C   PyEval_EvalFrameEx
C   fast_function
C   call_function
C   PyEval_EvalFrameEx
C   PyEval_EvalCodeEx
C   PyEval_EvalCode
C   run_mod
C   PyRun_FileExFlags
C   PyRun_SimpleFileExFlags
C   PyRun_AnyFileExFlags
```

Glue code

| Name | Type | Value |
|---|---|---|
| Arguments | | |
| n | int | 0x00000003 (3) |

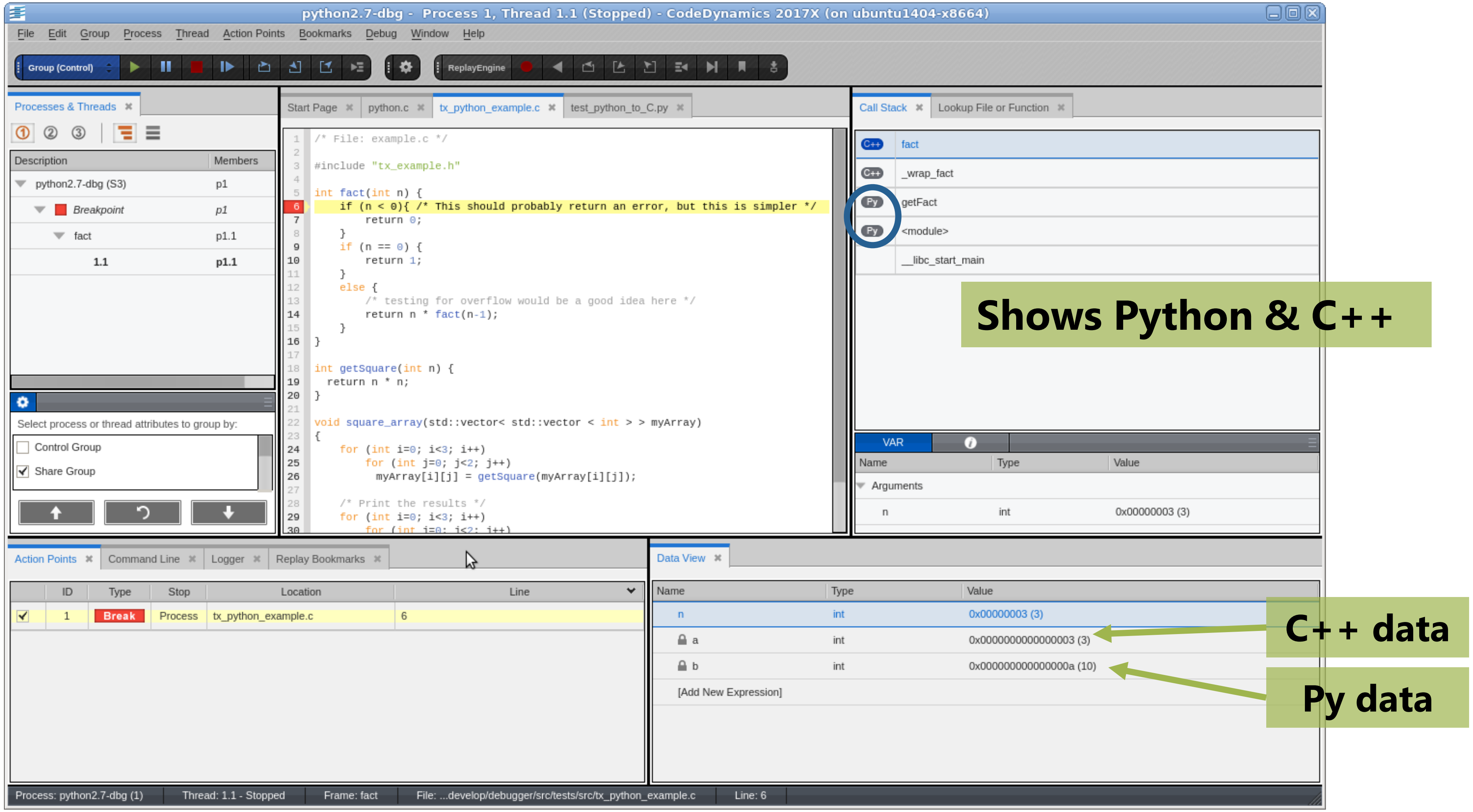
```
Thread 1.1 has appeared
Created process 1 (14759), named "python2.7-dbg"
Thread 1.1 has appeared
Thread 1.1 has exited
Thread 1.1 hit breakpoint 1 at line 6 in "fact(int)"
d1.<> std
disabled
d1.<>
```

| Name | Type | Value |
|---|---|---|
| [Add New Expression] | | |

Process: python2.7-dbg (1) | Thread: 1.1 - Stopped | Frame: fact | File: ...develop/debugger/src/tests/src/tx_python_example.c | Line: 6

# Showing C code with mixed data

- Glue code filtered out
- Python data and code available for viewing



| Description | Members |
|---|---|
| python2.7-dbg (S3) | p1 |
| Breakpoint | p1 |
| fact | p1.1 |
| 1.1 | p1.1 |

Select process or thread attributes to group by:

☐ Control Group

☑ Share Group

```
/* File: example.c */

#include "tx_example.h"

int fact(int n) {
    if (n < 0){ /* This should probably return an error, but this is simpler */
        return 0;
    }
    if (n == 0) {
        return 1;
    }
    else {
        /* testing for overflow would be a good idea here */
        return n * fact(n-1);
    }
}

int getSquare(int n) {
  return n * n;
}

void square_array(std::vector< std::vector < int > > myArray)
{
    for (int i=0; i<3; i++)
        for (int j=0; j<2; j++)
         myArray[i][j] = getSquare(myArray[i][j]);

    /* Print the results */
    for (int i=0; i<3; i++)
        for (int j=0; j<2; j++)
```

| | |
|---|---|
| C++ | fact |
| C++ | _wrap_fact |
| Py | getFact |
| Py | <module> |
| | __libc_start_main |

Shows Python & C++

| Name | Type | Value |
|---|---|---|
| Arguments | | |
| n | int | 0x00000003 (3) |

| | ID | Type | Stop | Location | Line |
|---|---|---|---|---|---|
| ☑ | 1 | Break | Process | tx_python_example.c | 6 |

| Name | Type | Value |
|---|---|---|
| n | int | 0x00000003 (3) |
| a | int | 0x0000000000000003 (3) |
| b | int | 0x000000000000000a (10) |
| [Add New Expression] | | |

C++ data

Py data

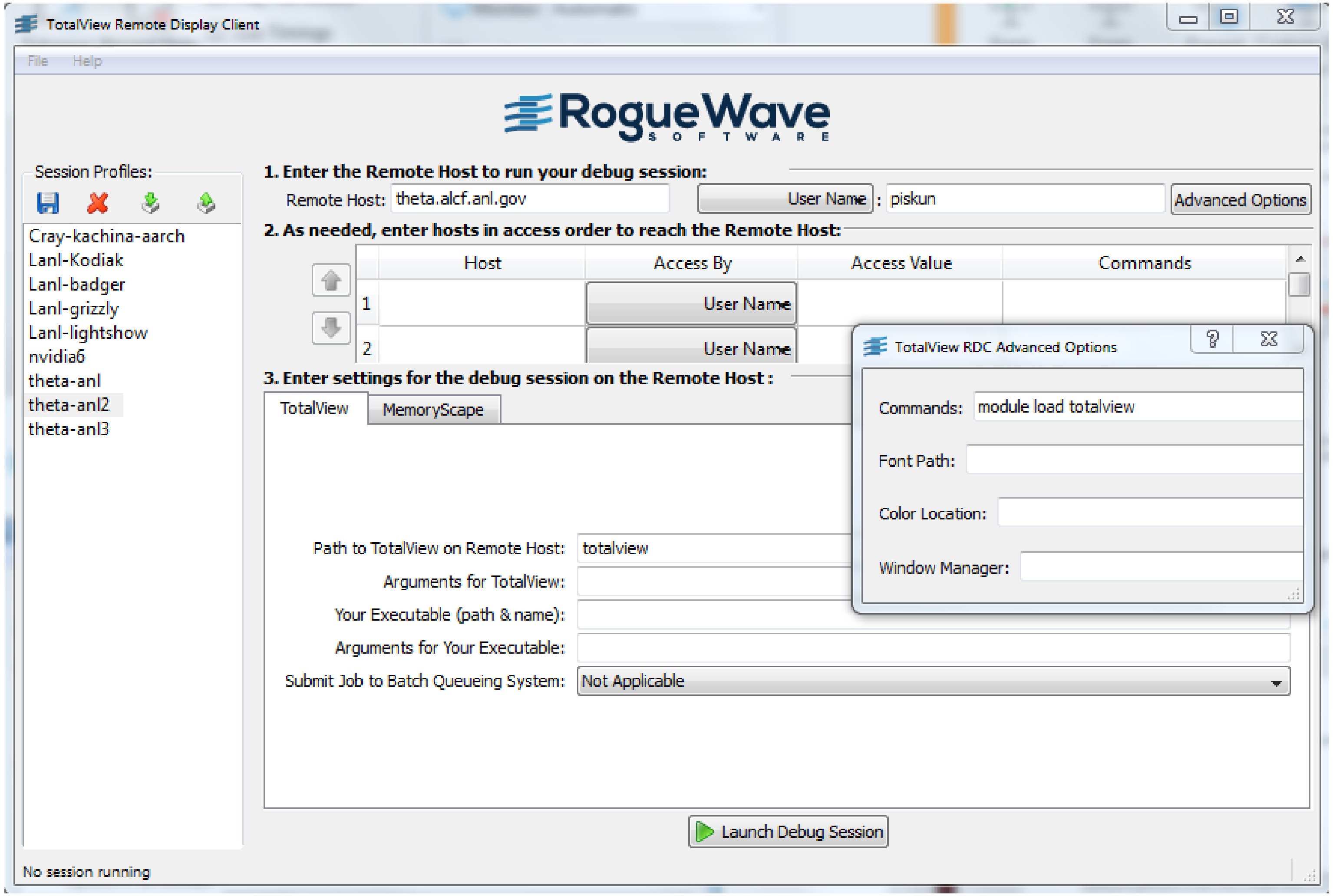
Process: python2.7-dbg (1) Thread: 1.1 - Stopped Frame: fact File: ...develop/debugger/src/tests/src/tx_python_example.c Line: 6

# Remote Display Debugging

# Remote Display Client (RDC)

- Offers users the ability to easily set up and operate a TotalView debug session that is running on another system
- Consists of two components
  - Client – runs on local machine
  - Server – runs on any system supported by TotalView and “invisibly” manages the secure connection between host and client
- Free to install on as many clients as needed
- Remote Display Client is available for:
  - Linux x86, x86-64
  - Windows
  - Mac OS X

# Remote Display Client



Session Profiles:

- Cray-kachina-aarch
- Lanl-Kodiak
- Lanl-badger
- Lanl-grizzly
- Lanl-lightshow
- nvidia6
- theta-anl
- theta-anl2
- theta-anl3

1. Enter the Remote Host to run your debug session:

Remote Host: theta.alcf.anl.gov User Name : piskun

2. As needed, enter hosts in access order to reach the Remote Host:

|  | Host | Access By | Access Value | Commands |
|---|---|---|---|---|
| 1 |  | User Name |  |  |
| 2 |  | User Name |  |  |

3. Enter settings for the debug session on the Remote Host :

Path to TotalView on Remote Host: totalview

Arguments for TotalView:

Your Executable (path & name):

Arguments for Your Executable:

Submit Job to Batch Queueing System: Not Applicable

No session running

TotalView RDC Advanced Options

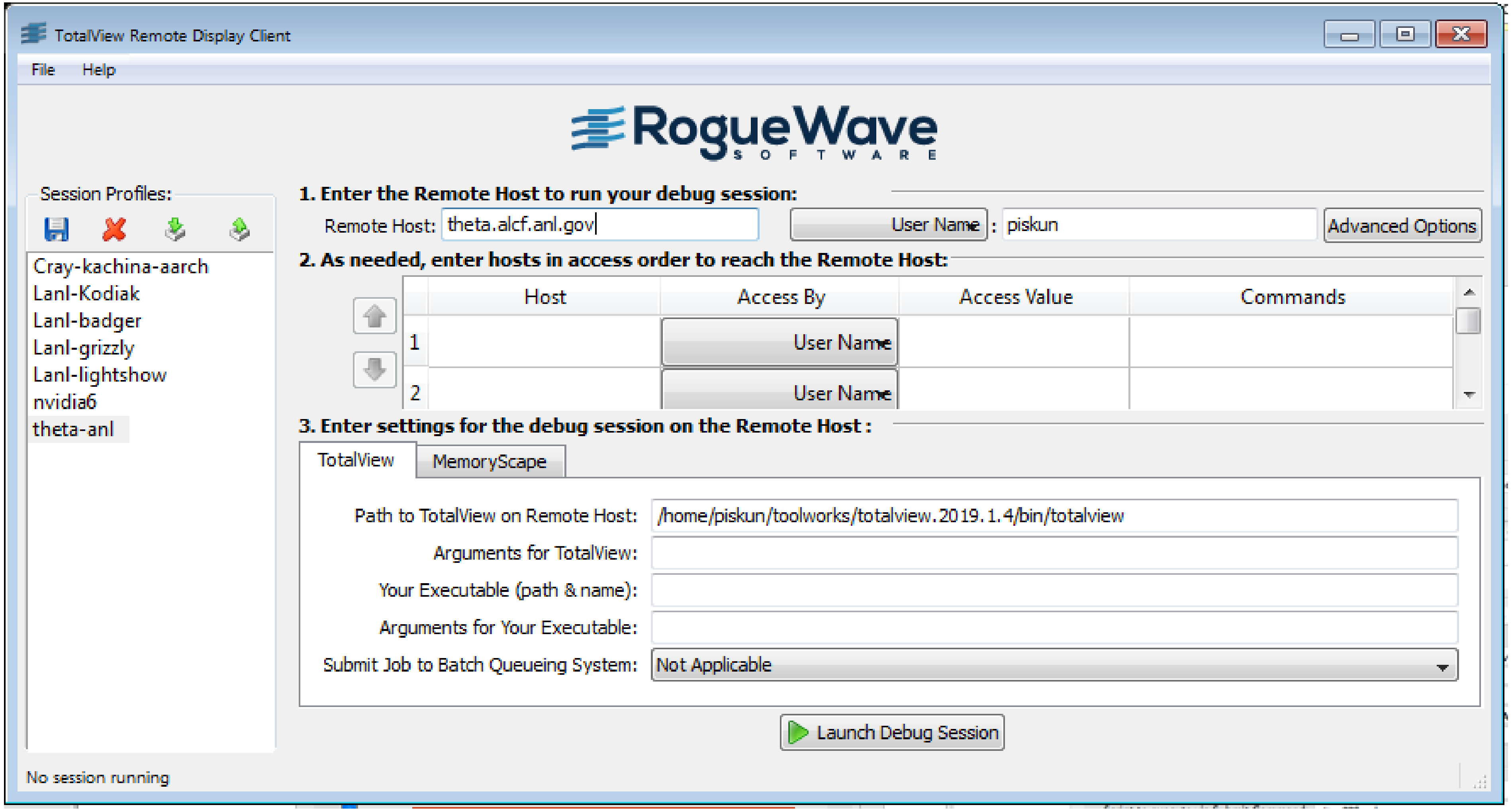
Commands: module load totalview

Font Path:

Color Location:

Window Manager:

# Remote Display Client (Argonne NL)



Cray-kachina-aarch
Lanl-Kodiak
Lanl-badger
Lanl-grizzly
Lanl-lightshow
nvidia6
theta-anl

1. Enter the Remote Host to run your debug session:

Remote Host: theta.alcf.anl.gov

User Name : piskun

2. As needed, enter hosts in access order to reach the Remote Host:

|  | Host | Access By | Access Value | Commands |
|---|---|---|---|---|
| 1 |  | User Name |  |  |
| 2 |  | User Name |  |  |

3. Enter settings for the debug session on the Remote Host :

Path to TotalView on Remote Host: /home/piskun/toolworks/totalview.2019.1.4/bin/totalview

Arguments for TotalView:

Your Executable (path & name):

Arguments for Your Executable:

Submit Job to Batch Queueing System: Not Applicable

No session running

# Summary

- Use of modern debugger saves you time.

- TotalView can help you because:
  - It’s **cross-platform** (the only debugger you ever need)
  - Allow you to debug accelerators (GPU) and CPU in **one session**
  - Allow you to debug **multiple languages** (C++/Python/Fortran)

# Using TotalView for Parallel Debugging on ANL

# Starting a MPI job – method 1

For HPC we have two methods to start the debugger

The ‘classic’ method

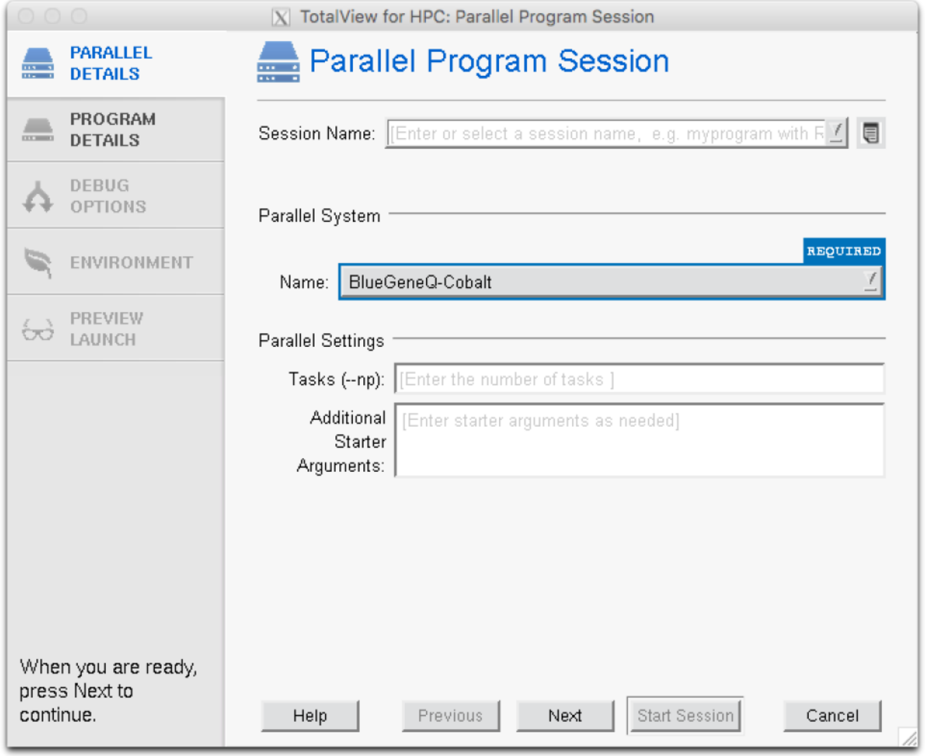
  - **`totalview -args mpiexec -np 512 ./myMPIprog myarg1 myarg2`**
  - This will start up TotalView on the parallel starter (mpiexec, srun, runjob, etc) and when you hit ‘Go’ the job will start up and the processes will be automatically attached.  At that point you will see your source and can set breakpoints.

- Some points to consider…
  - You don’t see your source at first, since we’re ‘debugging’ the mpi starter
  - Some MPI’s don’t support the process acquistion method (most do, but might be stripped of symbols we need when packaging)
  - In general more scalable than the next method...

# Starting a MPI job – method 2

The ‘indirect’ method

- Simply ‘totalview’ or ‘totalview myMPIprog’ and then you can choose a parallel system, number of tasks, nodes, and arguments to the program.



TotalView for HPC: Parallel Program Session

Parallel Program Session

Session Name: [Enter or select a session name, e.g. myprogram with F

Parallel System

Name: BlueGeneQ-Cobalt

Parallel Settings

Tasks (--np): [Enter the number of tasks ]

Additional Starter Arguments: [Enter starter arguments as needed]

When you are ready, press Next to continue.

- With this method the program source is available immediately
- Less dependent on MPI starter symbols
- May not be as scalable as some ‘indirect’ methods launch a debug server per process

# Using TotalView at Argonne

- TotalView available on Theta, Vesta, Mira, Cooley
  - Installed at: /soft/debuggers/totalview-2019-08-01/toolworks/totalview.2019T.2.7/bin/totalview
  - module load totalview
- Download and install RDC from https://www.roguewave.com/products-services/features/remote-display-client
- Connect to Theta
- Get allocation first
  - On Theta : qsub -A ATPESC2019 –n <N> –q debug-flat-quad –I
  - Module load totalview
  - totalview -args aprun –np <N> ….

# TotalView Resources and Documentation

# TotalView Resources & Documentation

- TotalView documentation:
  - https://support.roguewave.com/documentation/tvdocs/en/current/
  - User Guides:  Debugging, Memory Debugging and Reverse Debugging
  - Reference Guides:  Using the CLI, Transformations, Running TotalView
- TotalView online HTML doc:
  - http://docs.roguewave.com/totalview/current/html/index.html
- Other Resources (Blogs, videos, white papers, etc):
  - https://www.roguewave.com/resources?tagid=18
- New UI resources:
  - Reference CodeDynamics Help
    https://www.roguewave.com/help-support/documentation/codedynamics
- New UI videos:
  - https://www.roguewave.com/products-services/codedynamics/videos
- Python Debugging blog:
  - http://blog.klocwork.com/dynamic-analysis/the-challenge-debugging-python-and-cc-applications/

# Questions/Comments

- Any questions or comments?
  - Don’t hesitate to reach out to me directly with any problems or suggestions!
  - **Email:** nikolay.piskun@roguewave.com

- **Thank you for your time today!**

RogueWave
by Perforce

# THANK YOU