Advanced Technologies and Techniques for Debugging CUDA GPU HPC Applications

RogueWave SOFTWARE

Innovate with Confidence

Nikolay Piskun
Director of Continuing Engineering , TotalView products
GTC March, 2019

Agenda

- What is debugging and why TotalView?
- Overview of TotalView
- GPU debugging
- Python debugging
- Advanced C++ and Data debugging
- TotalView resources and documentation
- Questions/Comments

What is Debugging and Why do you need TotalView?

What is Debugging?

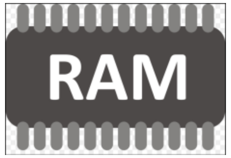
- Debugging is the process of finding and resolving defects or problems within a computer program or a system.
  - Algorithm correctness
  - Data correctness
  - Scaling/Porting correctness

TotalView debugger enables you to do:

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

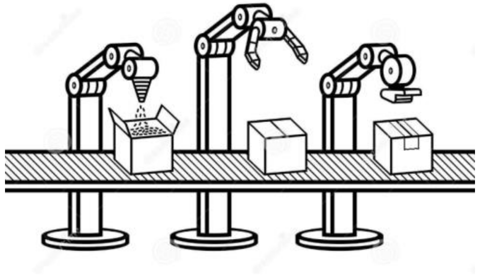
  » Debug a program running on another computer

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



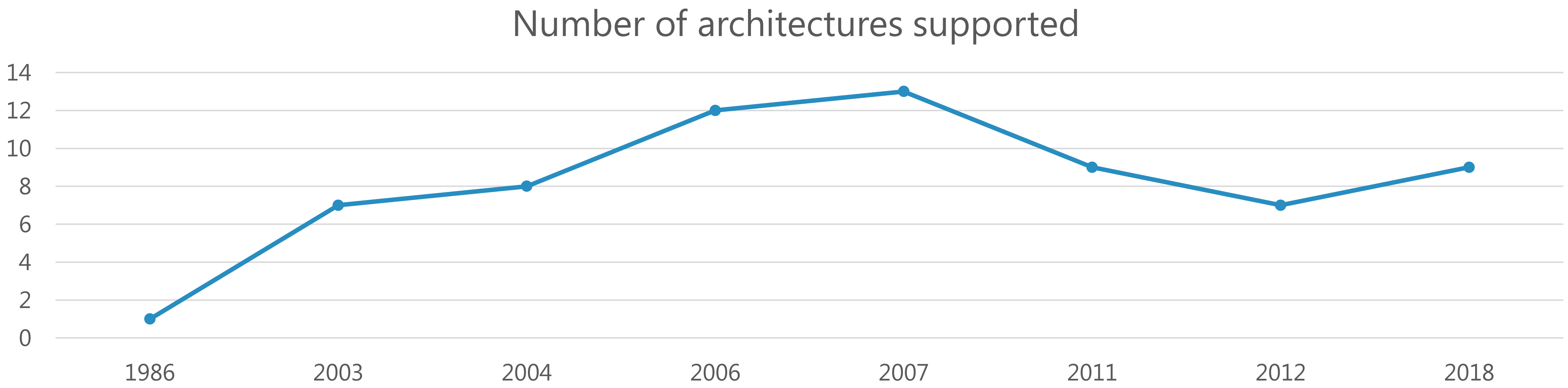
  » Find memory management problems (leaks, corruption …)

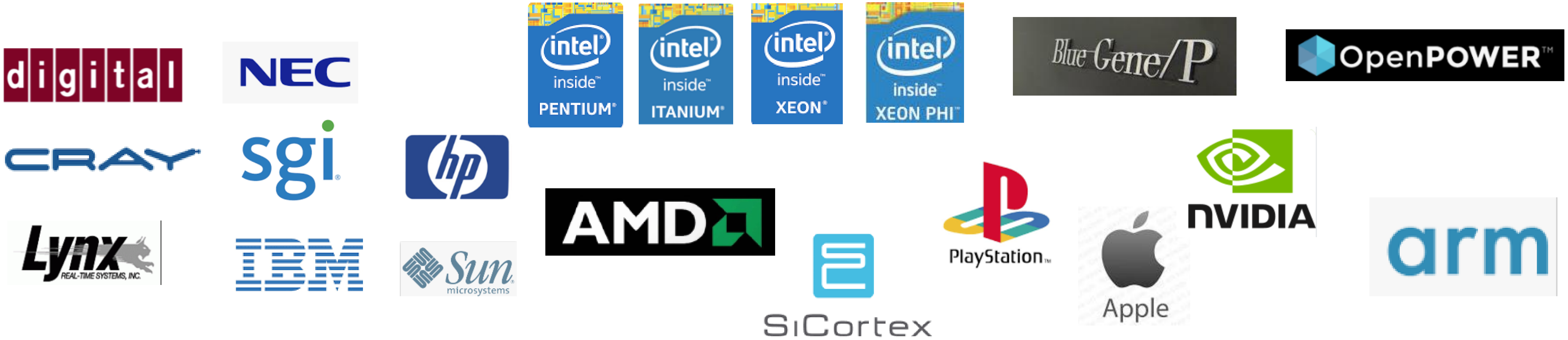
  » Comparing results between executions



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

History of TotalView

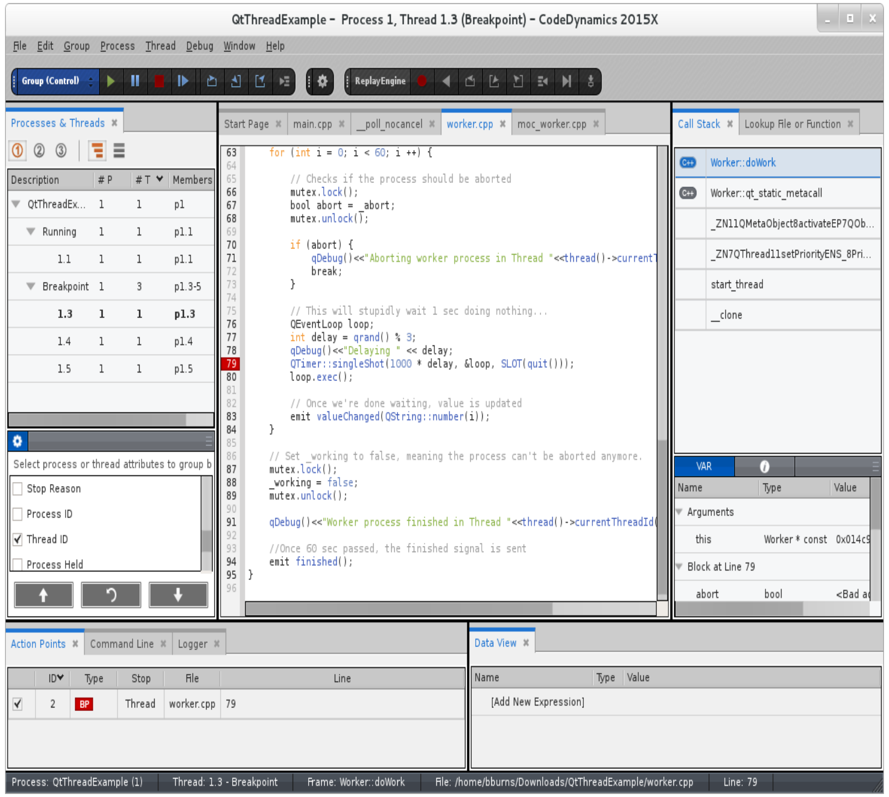




TotalView for HPC and for All

- 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
- Integrated Reverse debugging
- Batch non-interactive debugging.

- Allowing the business to have
  - <u>Predictable development schedules</u>
  - <u>Less time spent debugging</u>



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();
}
```

Worker::doWork
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

Nvidia software partner

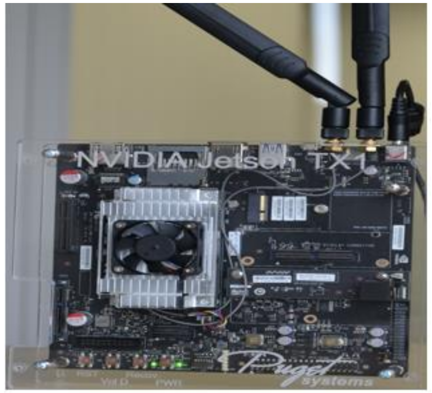
High Performance Computing Applications

- HPC Applications cover many different computing areas including:
  - Healthcare and Medicine
  - Modeling and simulation
  - Security
  - Bioinformatics
  - Molecular Dynamics
  - Environment (earthquake/tsunami)/Weather
  - Machine Learning/Artificial Intelligence

GPU Debugging

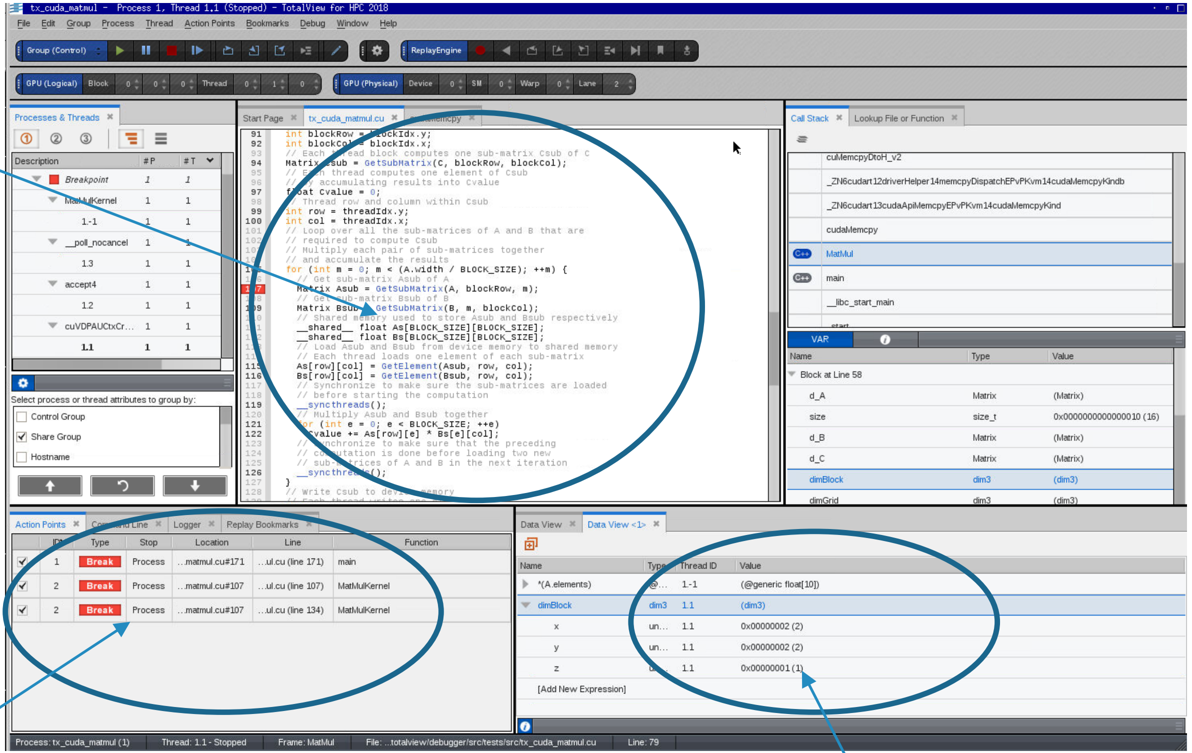
GPU debugging with TotalView

- NVIDIA CUDA support
    - Multiple platforms : X86-64, PowerLE, ARM64 (in beta)
    - Multiple cards: from Jetson to Volta (Turing testing)
- 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



CUDA Debugging Model Improvements

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



tx_cuda_matmul - Process 1, Thread 1.1 (Stopped) - TotalView for HPC 2018
File Edit Group Process Thread Action Points Bookmarks Debug Window Help
Group (Control)
ReplayEngine
GPU (Logical) Block 0 0 0 Thread 0 1 0
GPU (Physical) Device 0 SM 0 Warp 0 Lane 2
Processes & Threads
Start Page
tx_cuda_matmul.cu
cudaMemcpy
Call Stack
Lookup File or Function
VAR
Action Points
Command Line
Logger
Replay Bookmarks
Data View
Data View <1>

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
```

Call Stack:

- 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

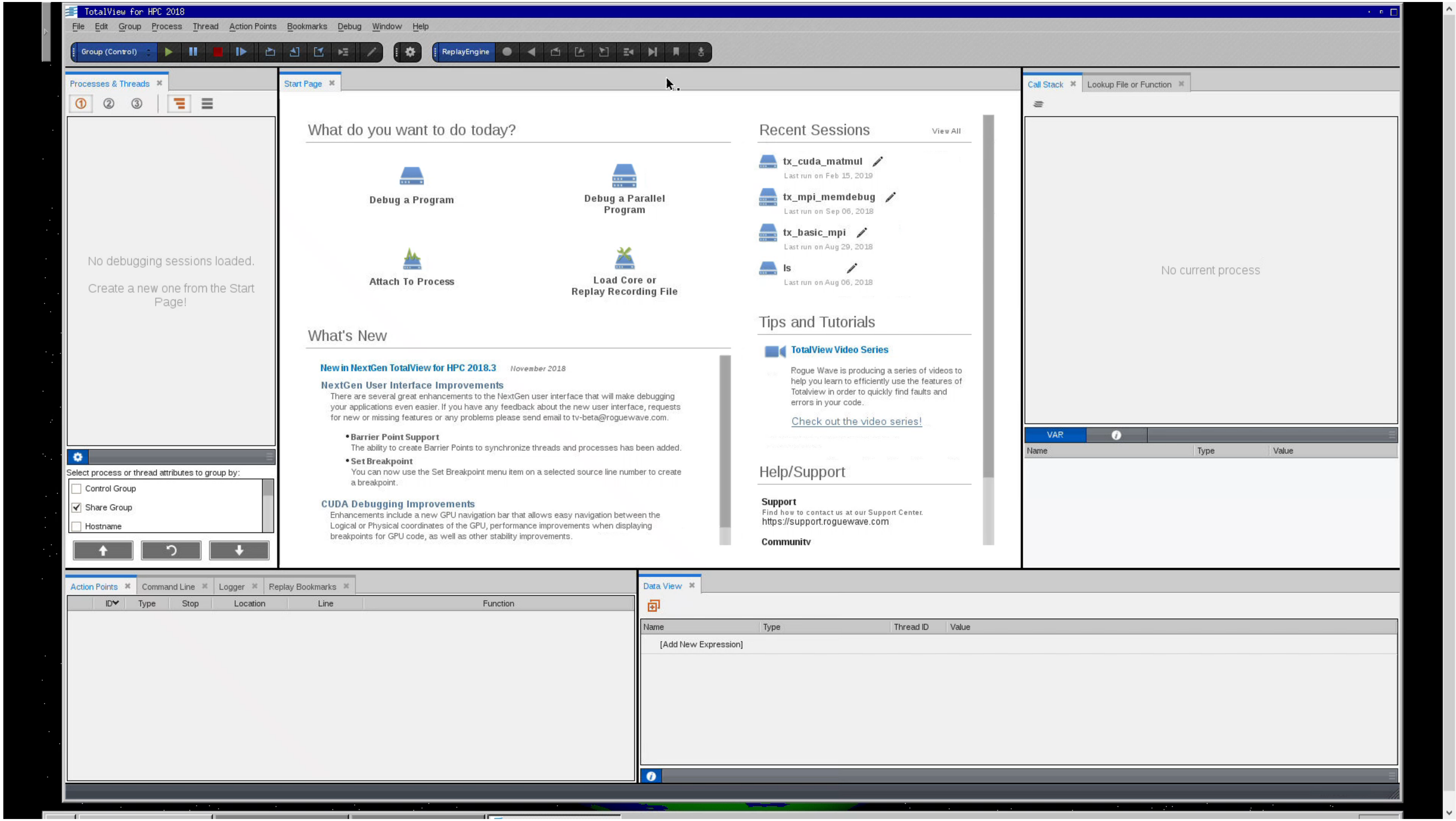
Name	Type	Thread ID	Value
*(A.elements)	@…	1.-1	(@generic float[10])
dimBlock	dim3	1.1	(dim3)
x	un…	1.1	0x00000002 (2)
y	un…	1.1	0x00000002 (2)
z	un…	1.1	0x00000001 (1)
[Add New Expression]			

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

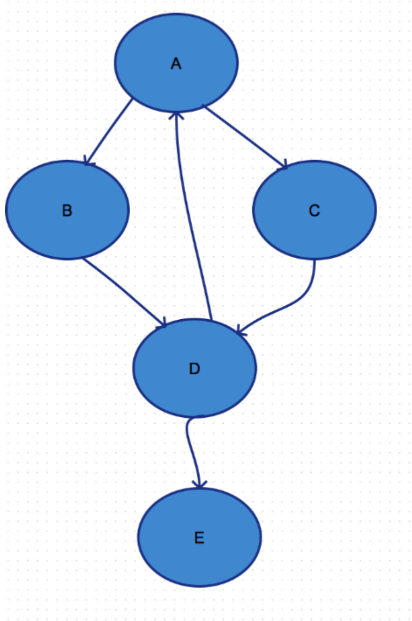
Python/C++ Debugging

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



Why Python ?

- Use Python to build applications that call out to C++
- Provides access to
  - High-performance routines
  - Leverage existing algorithms and libraries
  - Utilize advanced multi-threaded capabilities

- Calling between languages easily enabled using technologies such as SWIG, ctypes, Cython, CFFI, et al

- Debugging mixed language applications is not easy
  - Good for debugger developers ☺

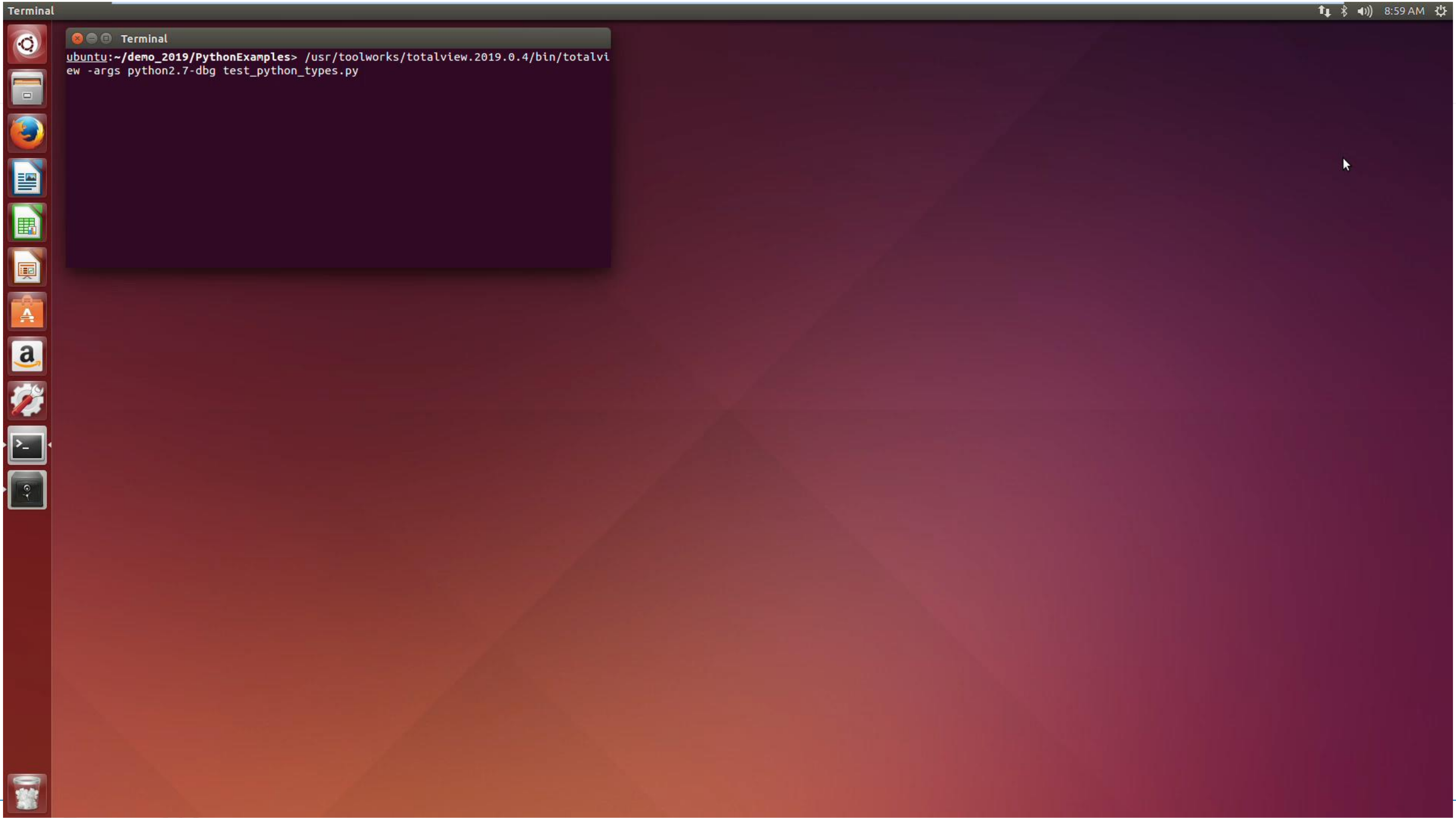
Python debugging with TotalView

- 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

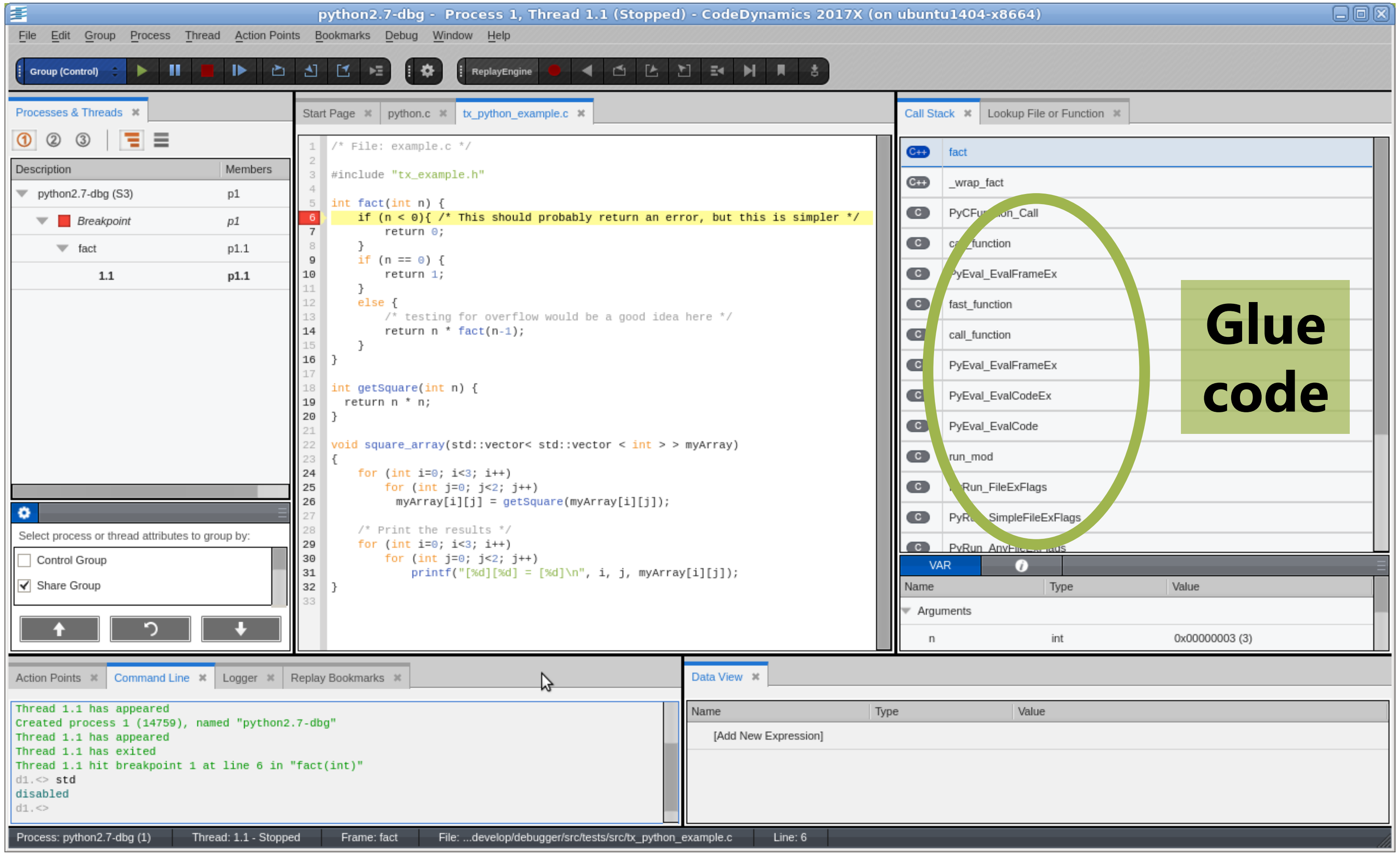
```
#!/usr/bin/python

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

```c
/* 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]);
}
```

C++	fact
C++	_wrap_fact
C	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)

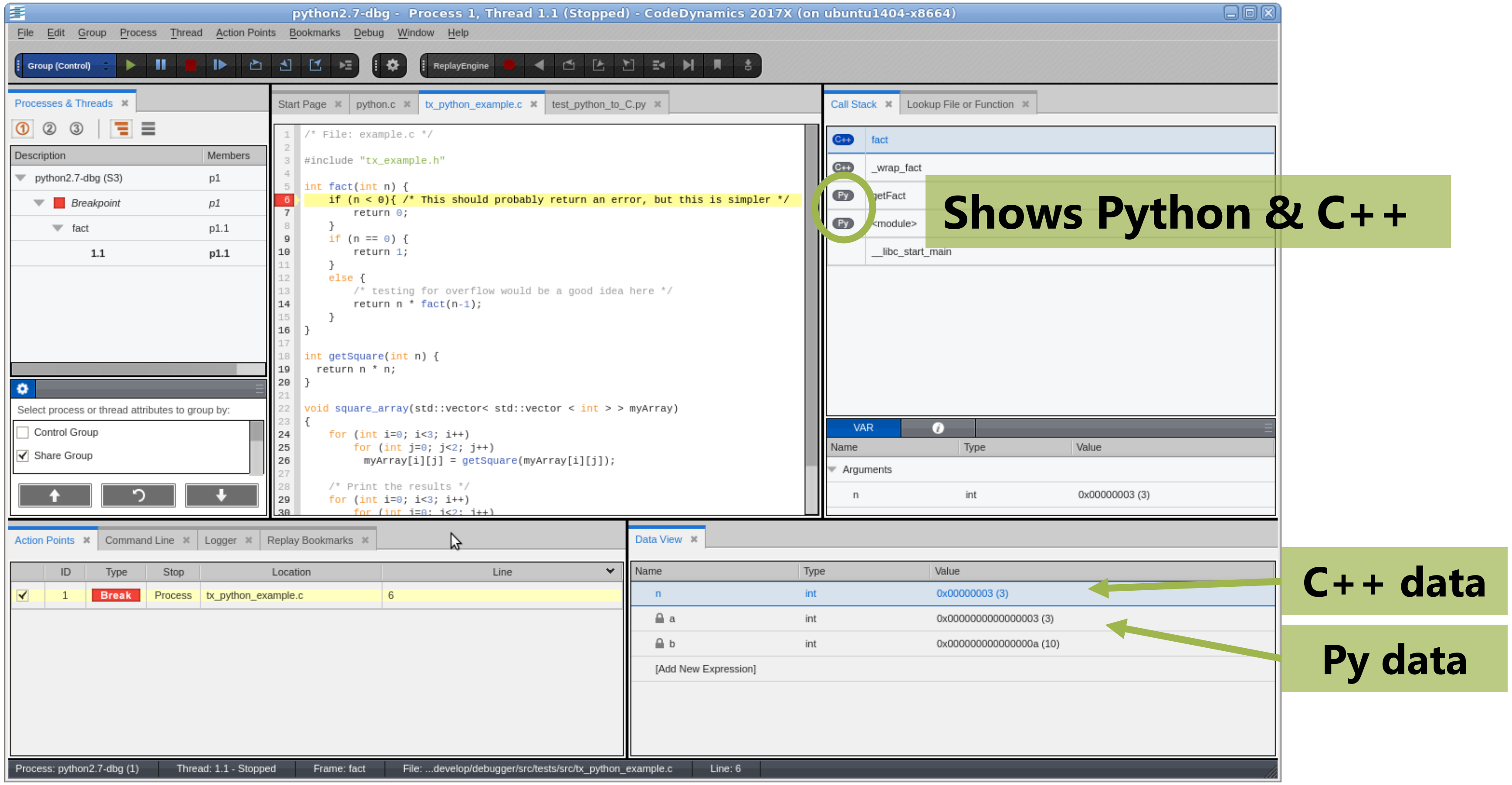
```
Thread 1.1 has appeared
Created process 1 (14759), named "python2.7-dbg"
Thread 1.1 has appeared
Thread 1.1 has exited
Thread 1.1 hit breakpoint 1 at line 6 in "fact(int)"
d1.<> std
disabled
d1.<>
```

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]		

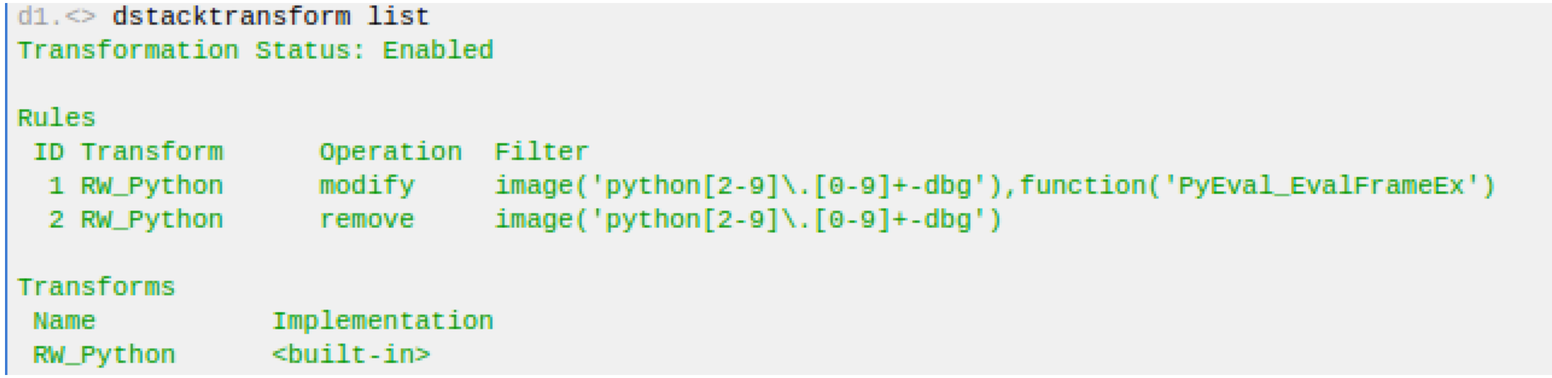
C++ data

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

Stack Transformation Facility

- Hides stack frames
- Transforms stack frames
- Backbone for:
  - Python support
  - OpenMP support
- **Useful for any glue code you want to hide**
  - **Language differences**
  - **Wrapper code**



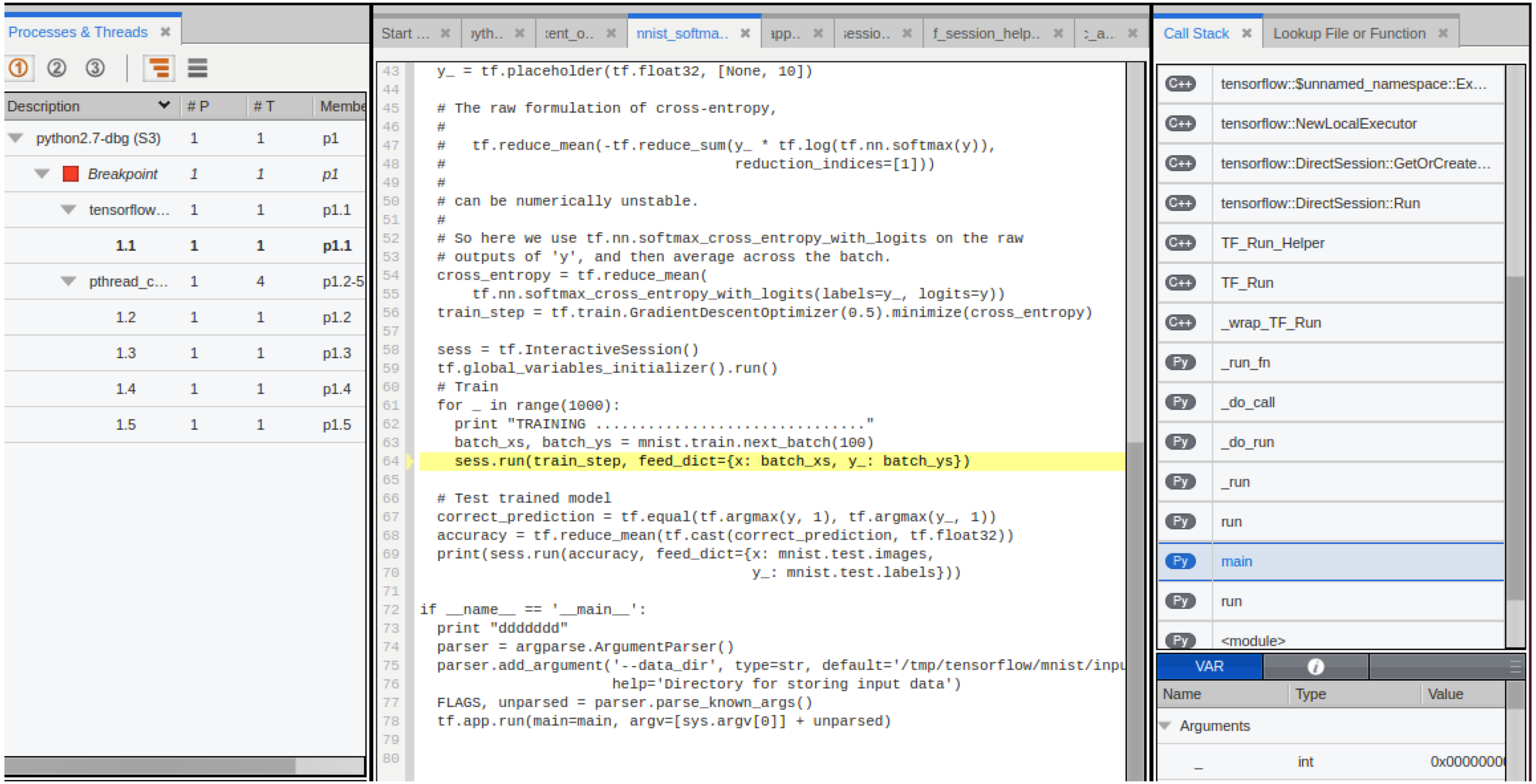
```
d1.<> dstacktransform list
Transformation Status: Enabled

Rules
 ID Transform      Operation  Filter
  1 RW_Python      modify     image('python[2-9]\.[0-9]+-dbg'),function('PyEval_EvalFrameEx')
  2 RW_Python      remove     image('python[2-9]\.[0-9]+-dbg')

Transforms
 Name           Implementation
 RW_Python      <built-in>
```

TensorFlow basics

- Open source
- Numerical computation
- Usage in machine learning
- Written in C++
  - Called from Python

TensorFlow

- Multi-threaded application
  - Glue code removed
  - Added a rule for wrappers



Description	# P	# T	Membe
python2.7-dbg (S3)	1	1	p1
Breakpoint	*1*	*1*	*p1*
tensorflow...	1	1	p1.1
1.1	**1**	**1**	**p1.1**
pthread_c...	1	4	p1.2-5
1.2	1	1	p1.2
1.3	1	1	p1.3
1.4	1	1	p1.4
1.5	1	1	p1.5

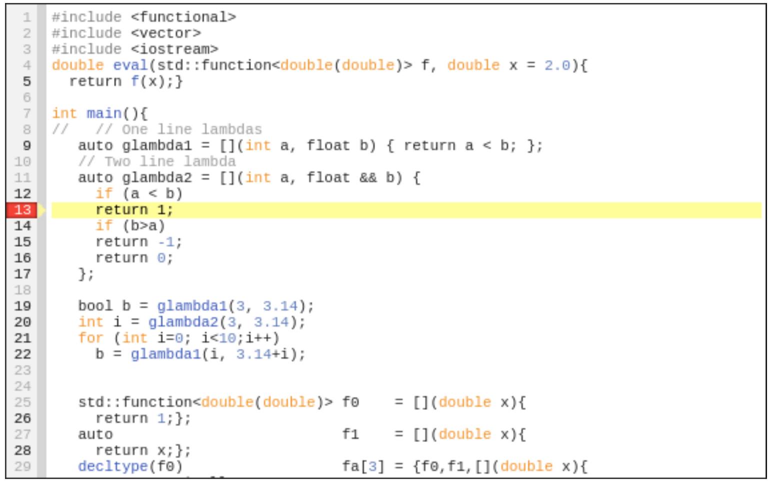
```
  y_ = tf.placeholder(tf.float32, [None, 10])

  # The raw formulation of cross-entropy,
  #
  #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
  #                                 reduction_indices=[1]))
  #
  # can be numerically unstable.
  #
  # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
  # outputs of 'y', and then average across the batch.
  cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

  sess = tf.InteractiveSession()
  tf.global_variables_initializer().run()
  # Train
  for _ in range(1000):
    print "TRAINING .............................."
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

  # Test trained model
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  print(sess.run(accuracy, feed_dict={x: mnist.test.images,
                                      y_: mnist.test.labels}))

if __name__ == '__main__':
  print "ddddddd"
  parser = argparse.ArgumentParser()
  parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/mnist/inpu
                      help='Directory for storing input data')
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

C++	tensorflow::$unnamed_namespace::Ex...
C++	tensorflow::NewLocalExecutor
C++	tensorflow::DirectSession::GetOrCreate...
C++	tensorflow::DirectSession::Run
C++	TF_Run_Helper
C++	TF_Run
C++	_wrap_TF_Run
Py	_run_fn
Py	_do_call
Py	_do_run
Py	_run
Py	run
Py	main
Py	run
Py	<module>

Name	Type	Value
Arguments		
_	int	0x0000000

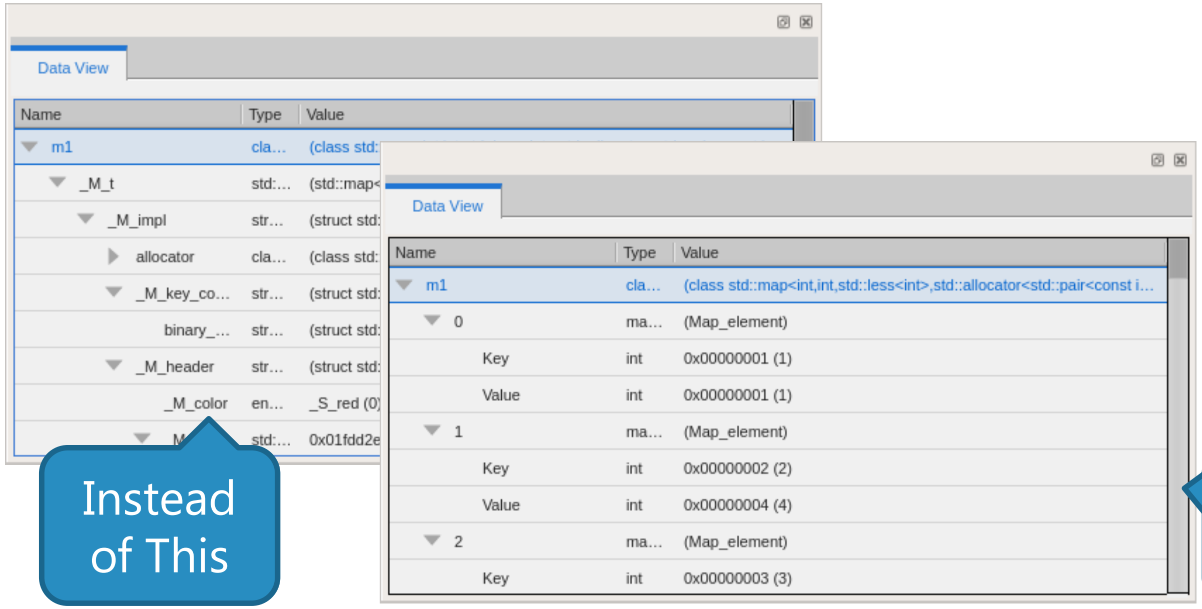
Advanced C++ and Data Debugging

Advanced C++ and Data Debugging



```cpp
#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

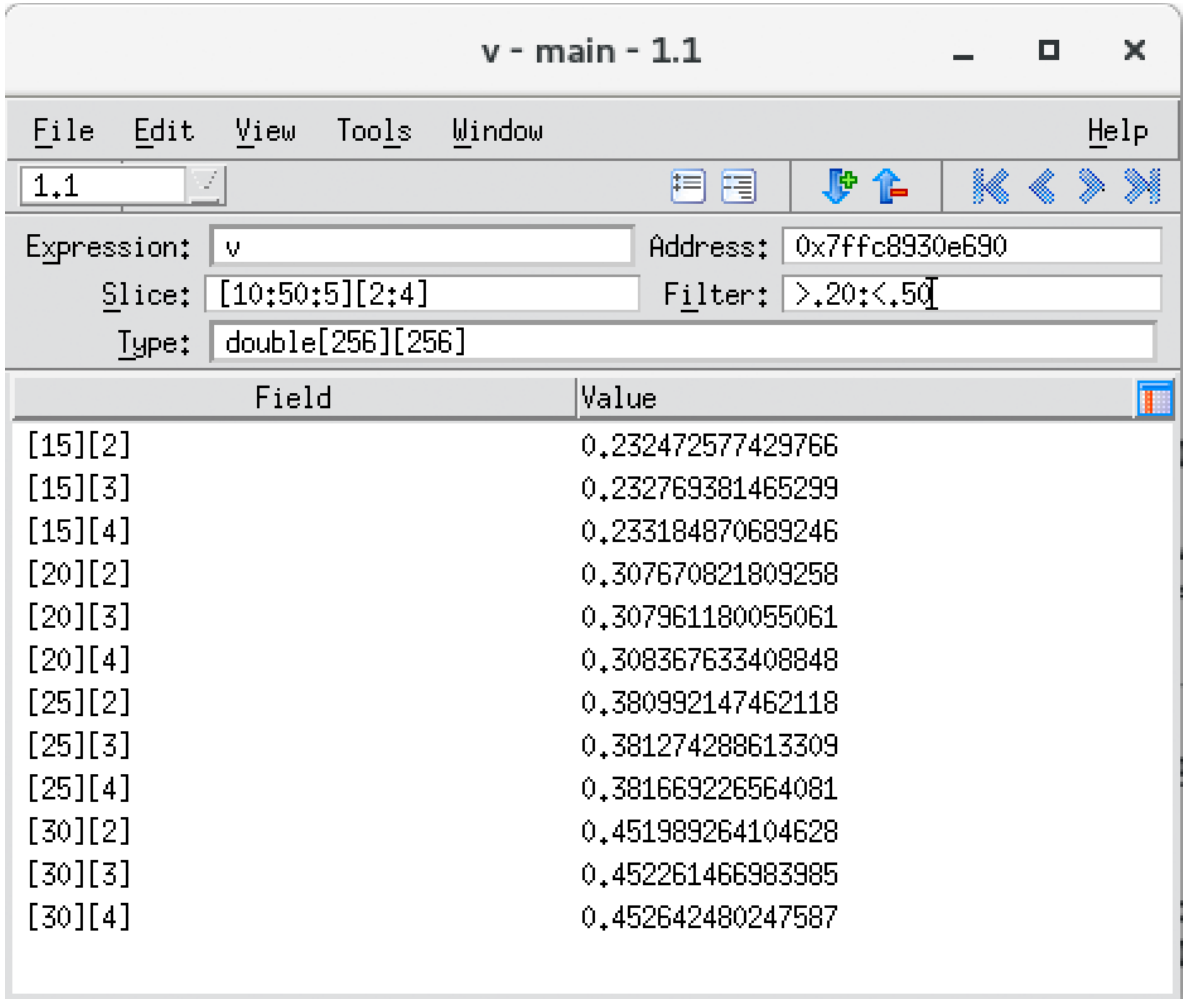
Name	Type	Value
m1	cla…	(class std::map<int,int,std::less<int>,std::allocator<std::pair<const i…
0	ma…	(Map_element)
Key	int	0x00000001 (1)
Value	int	0x00000001 (1)
1	ma…	(Map_element)
Key	int	0x00000002 (2)
Value	int	0x00000004 (4)
2	ma…	(Map_element)
Key	int	0x00000003 (3)

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



1.1

Expression: v

Address: 0x7ffc8930e690

Slice: [10:50:5][2:4]

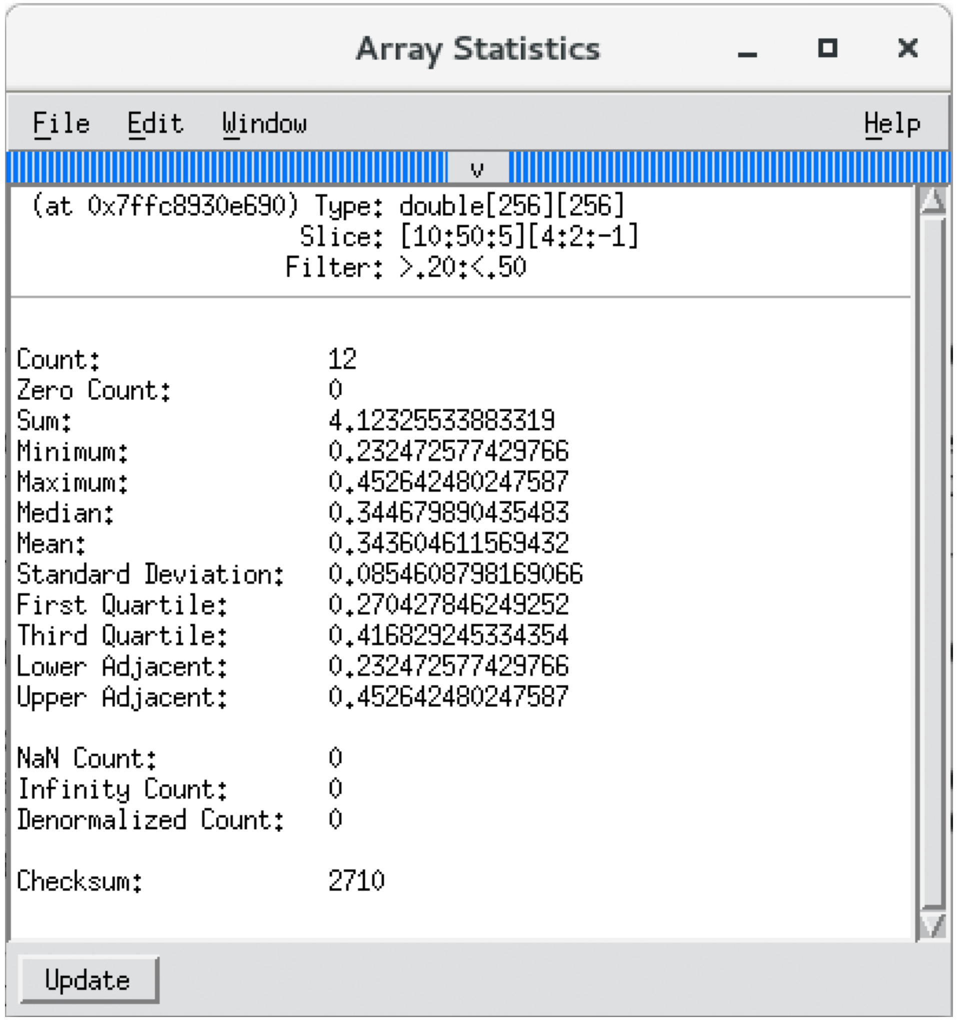
Filter: >.20:<.50

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

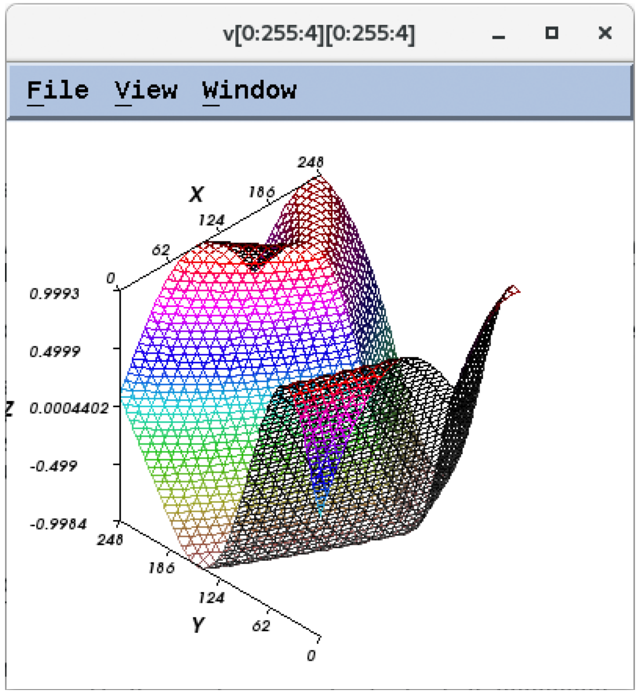
- Easily display a set of statistics for the filtered portion of your array

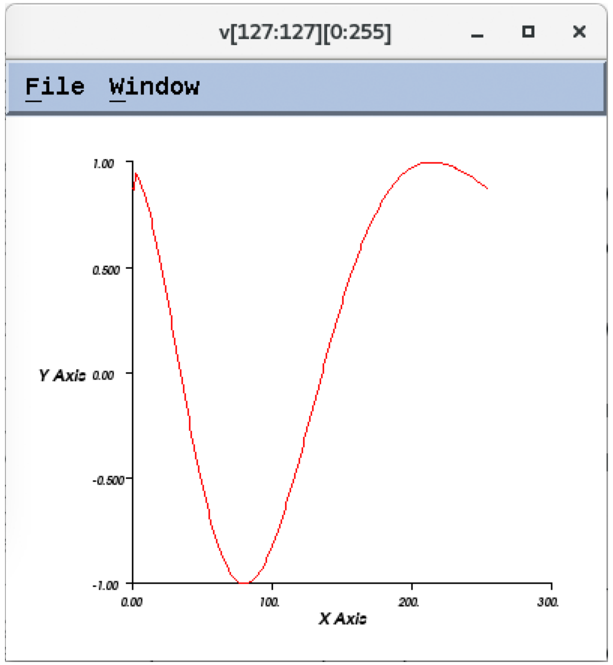


```
 (at 0x7ffc8930e690) Type: double[256][256]
                    Slice: [10:50:5][4:2:-1]
                   Filter: >.20:<.50

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





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)

Presenter

Nikolay Piskun

Director of CE, TotalView products

Rogue Wave Software/Perforce Software

nikolay.piskun@roguewave.com

RogueWave
SOFTWARE

Accelerating Great Code