Parallel Debugging Techniques & Introduction to Totalview

Le Yan
Louisiana Optical Network Initiative

LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

Outline

- Overview of parallel debugging
  - Challenges
  - Tools
  - Strategies
- Get familiar with TotalView through hands-on exercises

Outline

- Overview of parallel debugging
  - Challenges
  - Tools
  - Strategies
- Get familiar with TotalView through hands-on exercises

CENTER FOR COMPUTATION
& TECHNOLOGY

Bugs in Parallel Programming

- Parallel programs are prone to the usual bugs found in sequential programs
  - Improper pointer usage
  - Stepping over array bounds
  - Infinite loops
  - …
- Plus…

CENTER FOR COMPUTATION
& TECHNOLOGY

Common Types of Bugs in Parallel Programming

- Erroneous use of language features
  - Mismatched parameters, missing mandatory calls etc.
- Defective space decomposition
- Incorrect/improper synchronization
- Hidden serialization
- ……

CENTER FOR COMPUTATION & TECHNOLOGY

http://www.hpcbugbase.org/index.php/Main_Page

Debugging Essentials

- Reproducibility
  - Find the scenario where the error is reproducible
- Reduction
  - Reduce the problem to its essence
- Deduction
  - Form hypotheses on what the problem might be
- Experimentation
  - Filter out invalid hypotheses

CENTER FOR COMPUTATION
& TECHNOLOGY

Terrence Parr, *Learn The Essentials of Debugging*
http://www.ibm.com/developerworks/web/library/wa-debug.html?ca=dgr-lnxw03Dbug

Challenges in Parallel Debugging

- Reproducibility
  - Many problems cannot be easily reproduced
- Reduction
  - Smallest scale might still be too large and complex to handle
- Deduction
  - Need to consider concurrent and interdependent program instances
- Experimentation
  - Cyclic debugging might be very expensive

Bugs: A Little Example

```
…
integer*4 :: i,ista,iend
integer*4 :: chunksize=1024*1024
…
call MPI_Comm_Rank(MPI_COMM_WORLD, &
   myrank,error)
…
ista=myrank*chunksize+1
iend=(myrank+1)*chunksize
do i = ista,iend
   …
enddo
…
```

- What is the potential problem with large core count?

CENTER FOR COMPUTATION
& TECHNOLOGY

Bugs: A Little Example

```
…
integer*4 :: i,ista,iend
integer*4 :: chunksize=1024*1024
…
call MPI_Comm_Rank(MPI_COMM_WORLD, &
   myrank,error)
…
ista=myrank*chunksize+1
iend=(myrank+1)*chunksize
do i = ista,iend
   …
enddo
…
```

Integer overflow if myrank ≥ 4096

- A bug that shows up only when running with more than 4096 cores

CENTER FOR COMPUTATION & TECHNOLOGY

Debugging with write/printf

- Very easy to use and most portable, but…
  - Need to edit, recompile and rerun when additional information is desired
  - May change program behavior
  - Only capable of displaying a subset of the program's state
  - Output size grows rapidly with increasing core count and harder to comprehend
- Not recommended

LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

Compilers Can Help

- Most compilers can (at runtime)
  - Check array bounds
  - Trap floating operation errors
  - Provide traceback information
- Relatively scalable, but…
  - Overhead added
  - Limited capability
  - Non-interactive

CENTER FOR COMPUTATION
& TECHNOLOGY

Parallel Debuggers

- Capable of what serials debuggers can do
  – Control program execution
  – Set action points
  – View/edit values of variables
- More importantly
  – Control program execution at various levels
    - Group/process/thread
  – Display communication status between processes

CENTER FOR COMPUTATION
& TECHNOLOGY

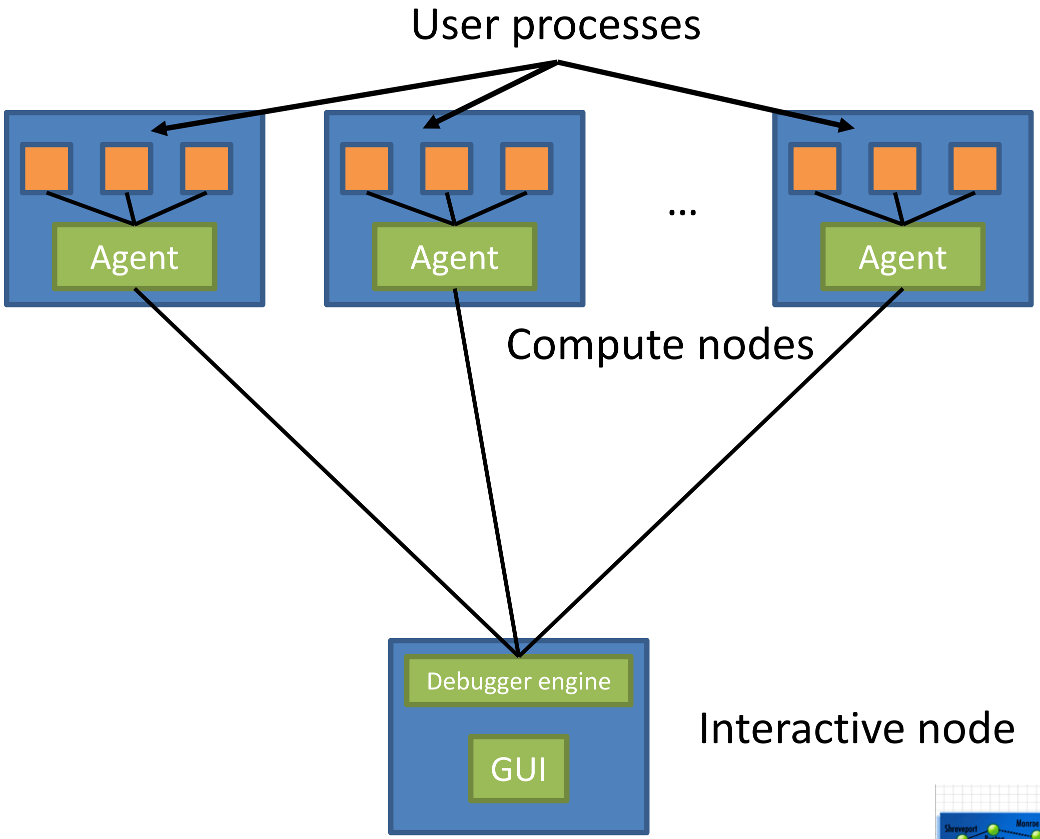
An Ideal Parallel Debugger

- Should allow easy process/thread control and navigation
- Should support multiple high performance computing platforms
- Should not limit the number of processes being debugged and should allow it to vary at runtime

CENTER FOR COMPUTATION
& TECHNOLOGY

How Parallel Debuggers Work

- Frontend
  - GUI
  - Debugger engine
- Debugger Agents
  - Control application processes
  - Send data back to the debugger engine to analyze



Debugging at Very Large Scale

- The debugger itself becomes a large parallel application
- Bottlenecks
  - Debugger framework startup cost
  - Communication between frontend and agents
  - Access to shared resources, e.g. file system

Validation Is Crucial

- Have a solid validation procedure to check the correctness
- Test smaller components before putting them together

CENTER FOR COMPUTATION
& TECHNOLOGY

General Parallel Debugging Strategy

- Incremental debugging
  - Downscale if possible
    - Participating processes, problem size and/or number of iterations
    - Example: run with one single thread to detect scope errors in OpenMP programs
  - Add more instances to reveal other issues
    - Example: run MPI programs on more than one node to detect problems introduced by network delays

Strategy at Large Scale

- Again, downscale if possible
- Reduce the number of processes to which the debugger is attached
  - Reduces overhead
  - Reduces the required number of license seats as well
- Focus on one or a small number of processes/threads
  - Analyze call path and message queues to find problematic processes
  - Control the execution of as few processes/threads as possible while keeping others running
    - Provides the context where the error occurs

Trends in Debugging Technology

- Lightweight trace analysis tools
  - Help to identify processes/threads that have similar behavior and reduce the search space
  - Complementary to full feature debuggers
  - Example: Stack Trace Analysis Tool (STAT)
- Replay/Reverse execution
  - ReplayEngine now available from TotalView
- Post-mortem statistical analysis
  - Detect anomalies by analyzing profile dissimilarity of multiple runs

Outline

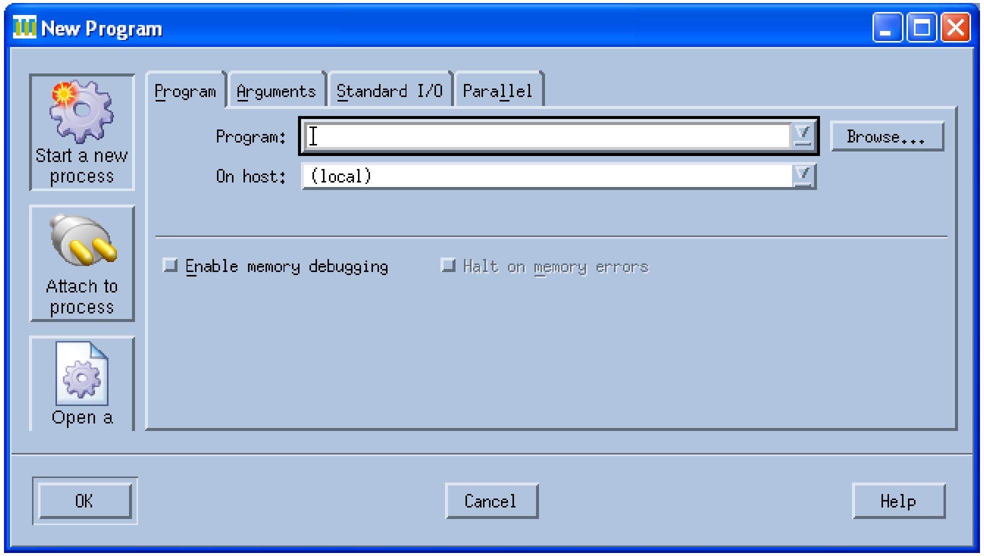
- Overview of parallel debugging
  - Challenges
  - Tools
  - Strategies
- Get familiar with TotalView through hands-on exercises

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

What Is TotalView

- A powerful debugger for both serial and parallel programs
  - Support Fortran, C/C++ and Assembler
  - Supported on most platforms
  - Both graphic and command line interface
- Features
  - Common debugging functions such as execution control and breakpoints
  - Memory debugging
  - Reverse debugging
  - Batch mode debugging
  - Remote debugging client
  - …

Three Ways to Start TotalView



Program:

On host: (local)

□ Enable memory debugging

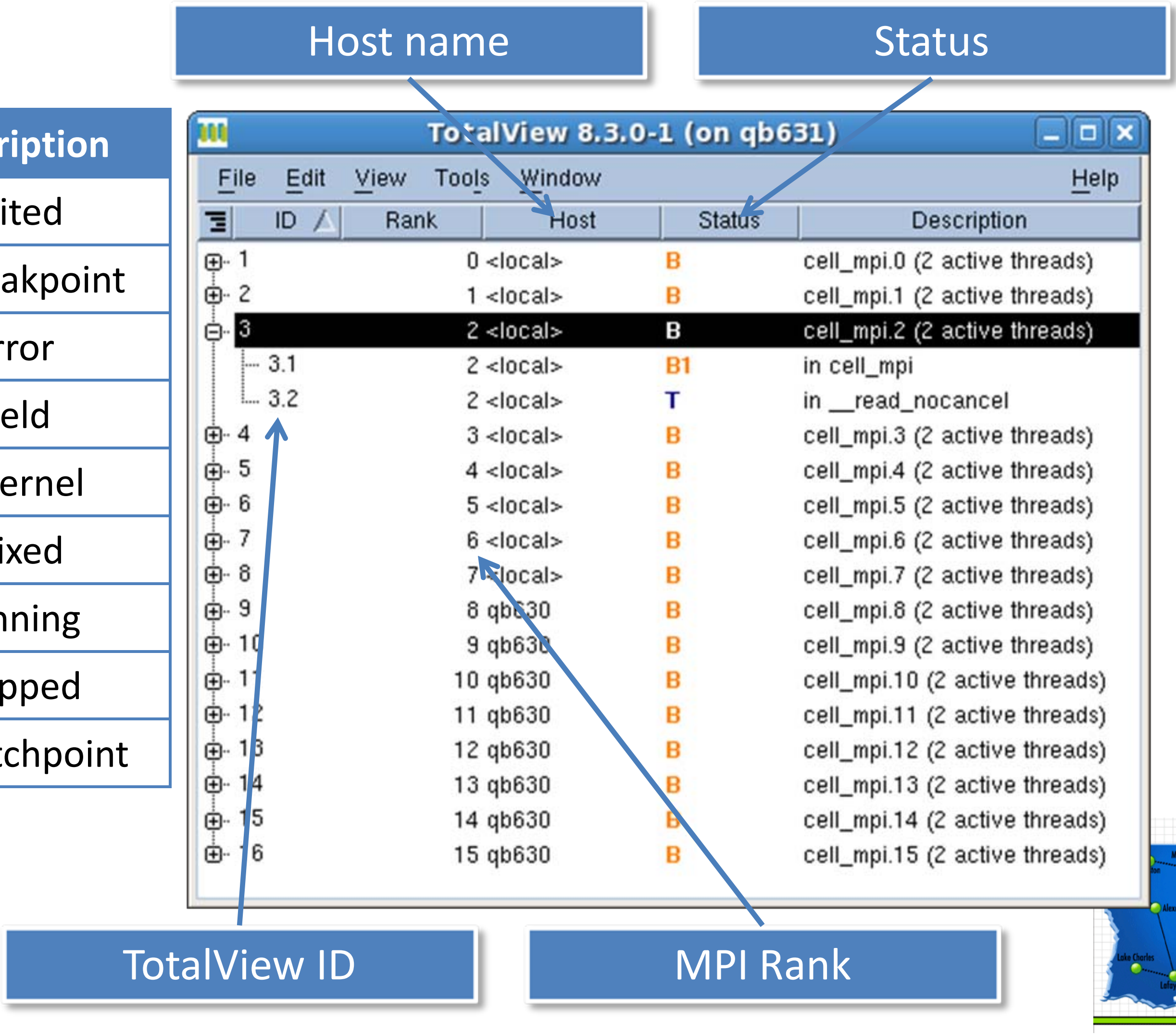
□ Halt on memory errors

- Start with core dumps
- Start by attaching to one or more running processes
- Start the executable within TotalView

CENTER FOR COMPUTATION
& TECHNOLOGY

User Interface - Root Window

Status Code	Description
Blank	Exited
B	At breakpoint
E	Error
H	Held
K	In kernel
M	Mixed
R	Running
T	Stopped
W	At watchpoint

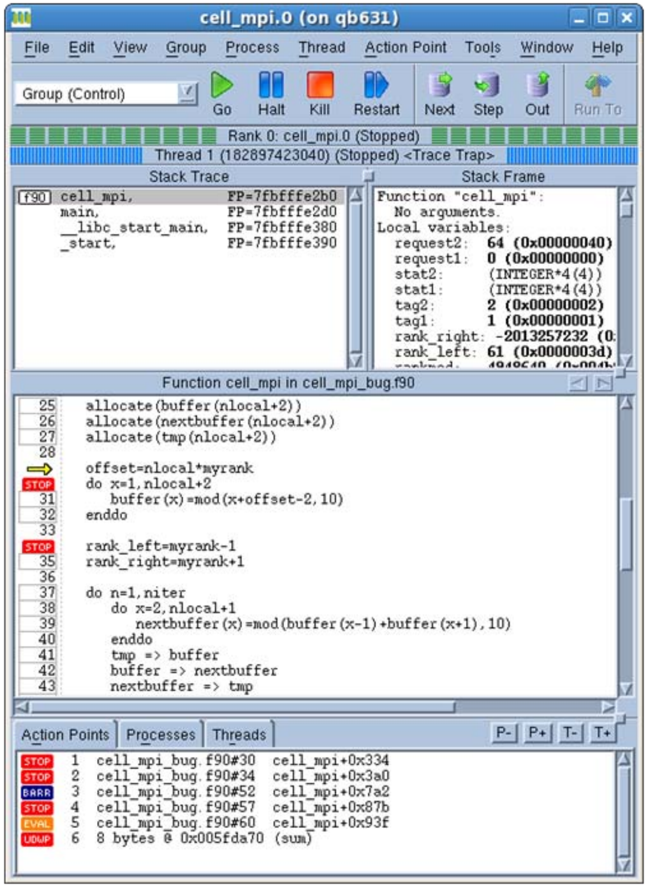


TotalView 8.3.0-1 (on qb631)

ID	Rank	Host	Status	Description
1	0	<local>	B	cell_mpi.0 (2 active threads)
2	1	<local>	B	cell_mpi.1 (2 active threads)
3	2	<local>	B	cell_mpi.2 (2 active threads)
3.1	2	<local>	B1	in cell_mpi
3.2	2	<local>	T	in __read_nocancel
4	3	<local>	B	cell_mpi.3 (2 active threads)
5	4	<local>	B	cell_mpi.4 (2 active threads)
6	5	<local>	B	cell_mpi.5 (2 active threads)
7	6	<local>	B	cell_mpi.6 (2 active threads)
8	7	<local>	B	cell_mpi.7 (2 active threads)
9	8	qb630	B	cell_mpi.8 (2 active threads)
10	9	qb630	B	cell_mpi.9 (2 active threads)
11	10	qb630	B	cell_mpi.10 (2 active threads)
12	11	qb630	B	cell_mpi.11 (2 active threads)
13	12	qb630	B	cell_mpi.12 (2 active threads)
14	13	qb630	B	cell_mpi.13 (2 active threads)
15	14	qb630	B	cell_mpi.14 (2 active threads)
16	15	qb630	B	cell_mpi.15 (2 active threads)

LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

User Interface – Process Window



Rank 0: cell_mpi.0 (Stopped)

Thread 1 (182897423040) (Stopped) <Trace Trap>

Stack Trace

```
f90  cell_mpi,             FP=7fbfffe2b0
     main,                 FP=7fbfffe2d0
     __libc_start_main,    FP=7fbfffe380
     _start,               FP=7fbfffe390
```

Stack Frame

```
Function "cell_mpi":
  No arguments.
Local variables:
  request2:    64 (0x00000040)
  request1:    0 (0x00000000)
  stat2:       (INTEGER*4(4))
  stat1:       (INTEGER*4(4))
  tag2:        2 (0x00000002)
  tag1:        1 (0x00000001)
  rank_right: -2013257232 (0:
  rank_left:  61 (0x0000003d)
```

Function cell_mpi in cell_mpi_bug.f90

```
   allocate(buffer(nlocal+2))
   allocate(nextbuffer(nlocal+2))
   allocate(tmp(nlocal+2))

⇨  offset=nlocal*myrank
STOP  do x=1,nlocal+2
         buffer(x)=mod(x+offset-2,10)
      enddo

STOP  rank_left=myrank-1
      rank_right=myrank+1

      do n=1,niter
         do x=2,nlocal+1
            nextbuffer(x)=mod(buffer(x-1)+buffer(x+1),10)
         enddo
         tmp => buffer
         buffer => nextbuffer
         nextbuffer => tmp
```

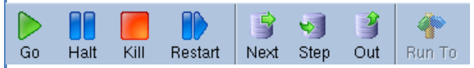
Action Points

```
STOP  1  cell_mpi_bug.f90#30  cell_mpi+0x334
STOP  2  cell_mpi_bug.f90#34  cell_mpi+0x3a0
BARR  3  cell_mpi_bug.f90#52  cell_mpi+0x7a2
STOP  4  cell_mpi_bug.f90#57  cell_mpi+0x87b
EVAL  5  cell_mpi_bug.f90#60  cell_mpi+0x93f
UDWP  6  8 bytes @ 0x005fda70 (sum)
```

- Stack trace pane
  - Call stack of routines
- Stack frame pane
  - Local variables, registers and function parameters
- Source pane
  - Source code
- Action points, processes, threads pane
  - Manage action points, processes and threads

CENTER FOR COMPUTATION
& TECHNOLOGY

Control Commands

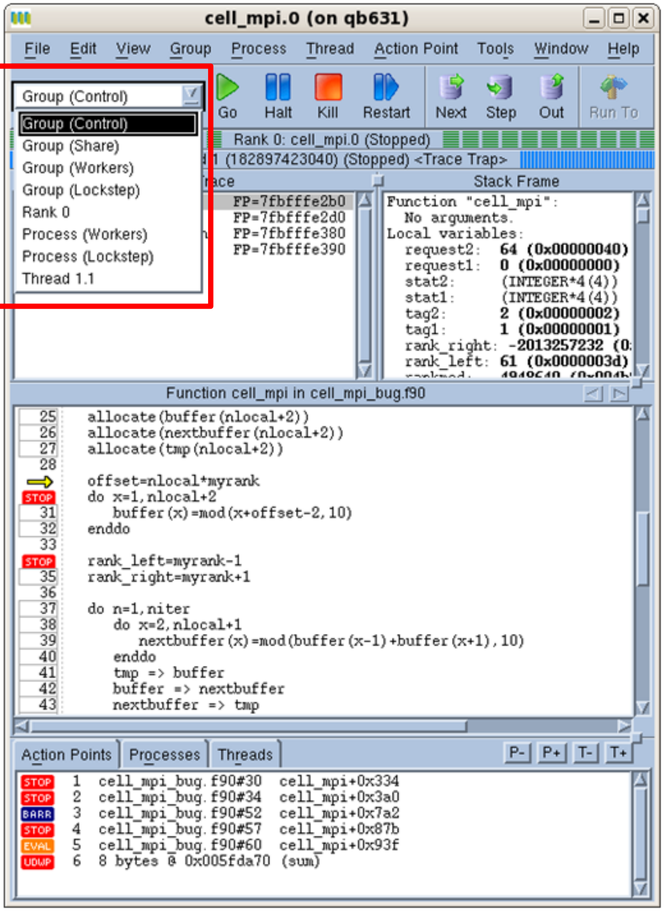
TotalView	Description
 Go Halt Kill Restart Next Step Out Run To	
Go	Start/resume execution
Halt	Stop execution
Kill	Terminate the job
Restart	Restarts a running program
Next	Run to the next source line without stepping into another function
Step	Run to next source line
Out	Run to the completion of current function
Run to	Run to the indicated location

CENTER FOR COMPUTATION
& TECHNOLOGY

Controlling Execution

- The process window always focuses on one process/thread
- Switch between processes/threads
  - p+/p-, t+/t-, double click in root window, process/thread tab
- Need to set the appropriate scope when
  - Giving control commands
  - Setting action points

Process/Thread Groups



Rank 0: cell_mpi.0 (Stopped)

1 (182897423040) (Stopped) <Trace Trap>

```
FP=7fbfffe2b0
FP=7fbfffe2d0
FP=7fbfffe380
FP=7fbfffe390
```

```
Function "cell_mpi":
  No arguments.
Local variables:
  request2:   64 (0x00000040)
  request1:   0 (0x00000000)
  stat2:      (INTEGER*4(4))
  stat1:      (INTEGER*4(4))
  tag2:       2 (0x00000002)
  tag1:       1 (0x00000001)
  rank_right: -2013257232 (0:
  rank_left:  61 (0x0000003d)
```

Function cell_mpi in cell_mpi_bug.f90

```
allocate(buffer(nlocal+2))
allocate(nextbuffer(nlocal+2))
allocate(tmp(nlocal+2))

offset=nlocal*myrank
do x=1,nlocal+2
   buffer(x)=mod(x+offset-2,10)
enddo

rank_left=myrank-1
rank_right=myrank+1

do n=1,niter
   do x=2,nlocal+1
      nextbuffer(x)=mod(buffer(x-1)+buffer(x+1),10)
   enddo
   tmp => buffer
   buffer => nextbuffer
   nextbuffer => tmp
```

```
STOP  1  cell_mpi_bug.f90#30  cell_mpi+0x334
STOP  2  cell_mpi_bug.f90#34  cell_mpi+0x3a0
BARR  3  cell_mpi_bug.f90#52  cell_mpi+0x7a2
STOP  4  cell_mpi_bug.f90#57  cell_mpi+0x87b
EVAL  5  cell_mpi_bug.f90#60  cell_mpi+0x93f
UDWP  6  8 bytes @ 0x005fda70 (sum)
```

- Scope of commands and action points
  - Group(control)
    - All processes and threads
  - Group(workers)
    - All threads that are executing user code
  - Rank X
    - Current process and its threads
  - Process(workers)
    - User threads in the current process
  - Thread X.Y
    - Current thread
  - User defined group
    - Group -> Custom Groups, or
    - Create in call graph

CENTER FOR COMPUTATION
& TECHNOLOGY

Types of Action Points

- **Breakpoints** stop the execution of the processes and threads that reach it
- **Evaluation points**: stop and execute a code fragment when reached
  - Useful when testing small patches
- **Process barrier points** synchronize a set of processes or threads
- **Watchpoints** monitor a location in memory and stop execution when its value changes
  - Unconditional
  - Conditional

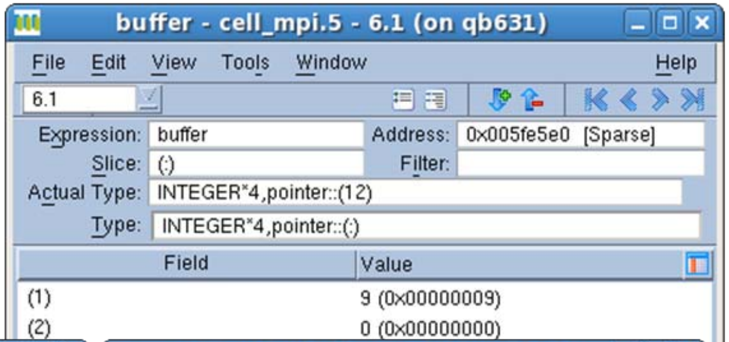
CENTER FOR COMPUTATION
& TECHNOLOGY

Setting Action Points

- Breakpoints
  - Right click on a source line -> Set breakpoint
  - Click on the line number
- Watch points
  - Right click on a variable -> Create watchpoint
- Barrier points
  - Right click on a source line -> Set barrier
- Edit action point property
  - Right click on a action point in the Action Points tab -> Properties

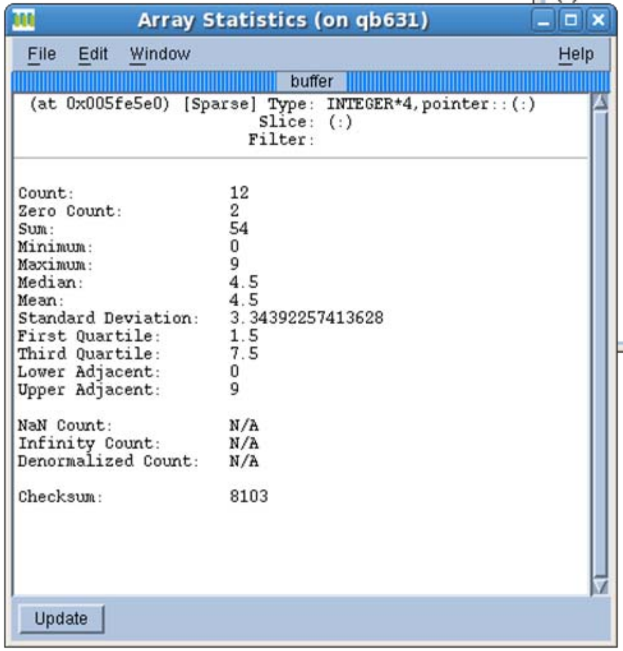
Viewing/Editing Data

- View values and types of variables
  - At one process/thread
  - Across all processes/threads
- Edit variable value and type
- Array Data
  - Slicing
  - Filtering
  - Visualization
  - Statistics



```
Expression: buffer                        Address: 0x005fe5e0 [Sparse]
     Slice: (:)                            Filter:
Actual Type: INTEGER*4,pointer::(12)
      Type: INTEGER*4,pointer::(:)

Field                 Value
(1)                   9 (0x00000009)
(2)                   0 (0x00000000)
```

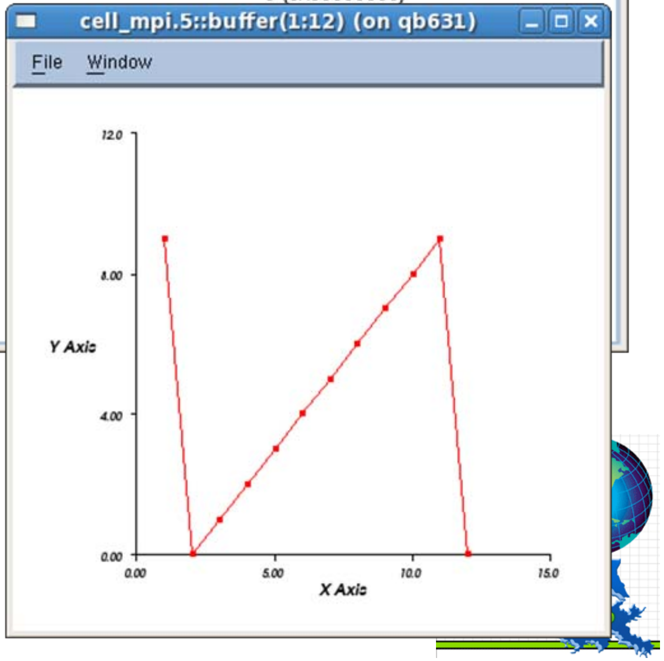


```
(at 0x005fe5e0) [Sparse] Type: INTEGER*4,pointer::(:)
                         Slice: (:)
                        Filter:

Count:                    12
Zero Count:               2
Sum:                      54
Minimum:                  0
Maximum:                  9
Median:                   4.5
Mean:                     4.5
Standard Deviation:       3.34392257413628
First Quartile:           1.5
Third Quartile:           7.5
Lower Adjacent:           0
Upper Adjacent:           9

NaN Count:                N/A
Infinity Count:           N/A
Denormalized Count:       N/A

Checksum:                 8103
```

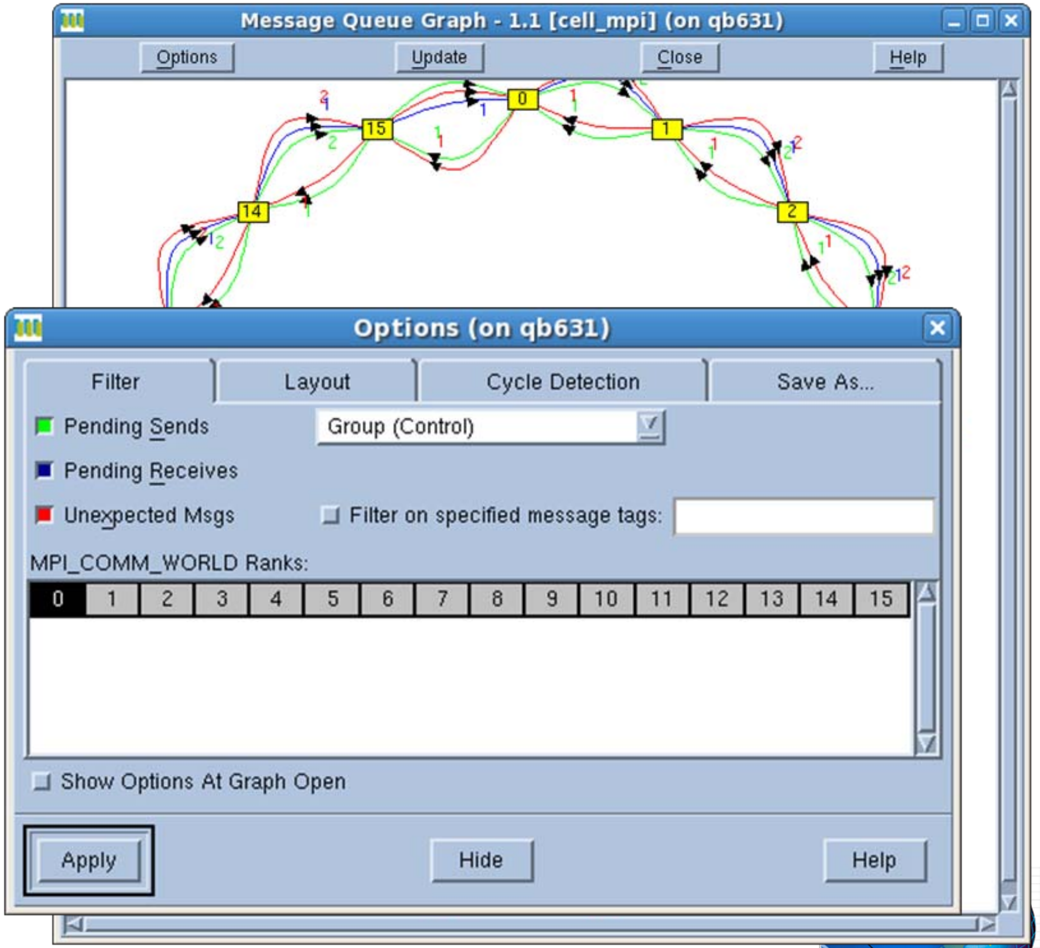


Viewing Dynamic Arrays in C/C++

- Edit “type” in the variable window
- Tell TotalView how to access the memory from a starting location
- Example
  - To view an array of 100 integers
    - Change “**Int ***” to “**int[100]***”

MPI Message Queues

- Detect
  - Deadlocks
  - Load balancing issues
- Tools -> Message Queue Graph
  - More options available



Message Queue Graph - 1.1 [cell_mpi] (on qb631)

Options (on qb631)

🟩 Pending Sends

Group (Control)

🟦 Pending Receives

🟥 Unexpected Msgs

□ Filter on specified message tags:

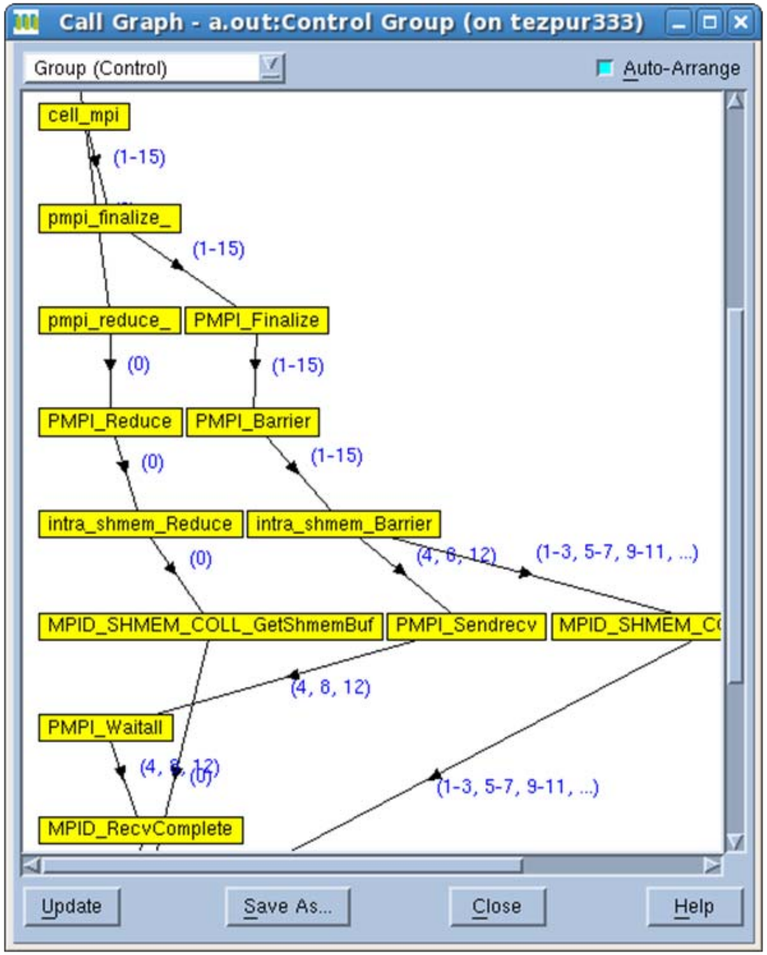
MPI_COMM_WORLD Ranks:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

□ Show Options At Graph Open

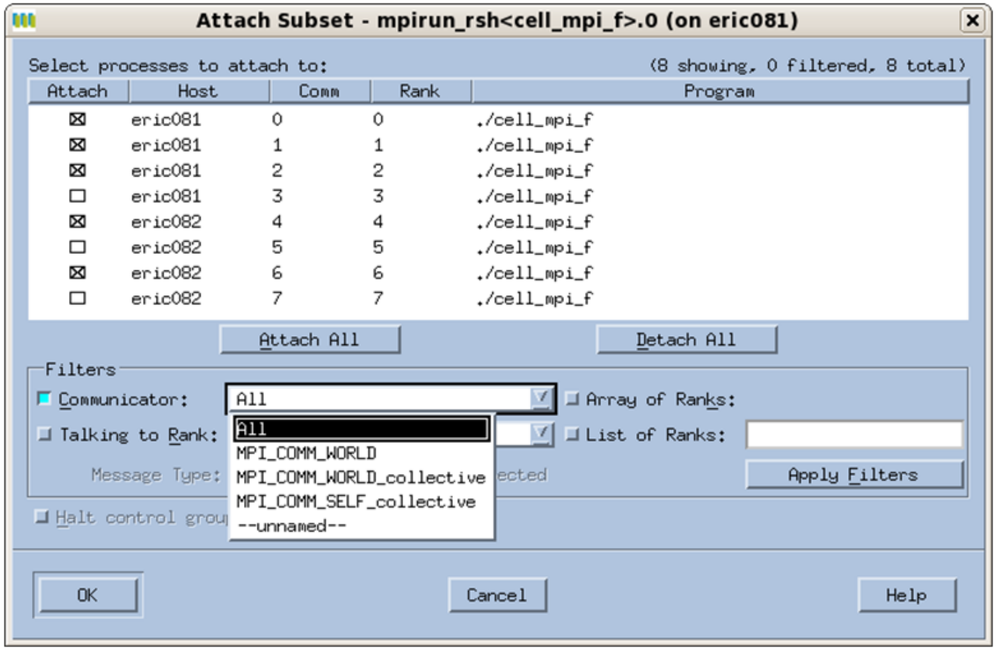
Call Graph

- Tools -> Call graph
- Quick view of program state
  - Nodes: functions
  - Edges: calls
- Look for outliers



Attaching to/Detaching from Processes

- You can
  - Attach to one or more running processes after launching TotalView
  - Launch the program within TotalView and detach from/reattach to any subset of processes later on



Attach Subset - mpirun_rsh<cell_mpi_f>.0 (on eric081)

Select processes to attach to: (8 showing, 0 filtered, 8 total)

Attach	Host	Comm	Rank	Program
☒	eric081	0	0	./cell_mpi_f
☒	eric081	1	1	./cell_mpi_f
☒	eric081	2	2	./cell_mpi_f
□	eric081	3	3	./cell_mpi_f
☒	eric082	4	4	./cell_mpi_f
□	eric082	5	5	./cell_mpi_f
☒	eric082	6	6	./cell_mpi_f
□	eric082	7	7	./cell_mpi_f

Filters

☑ Communicator: All

□ Talking to Rank:

Message Type:

□ Array of Ranks:

□ List of Ranks:

□ Halt control grou

Memory Debugging

- Features
  - Memory usage report
  - Error detection
    - Memory leak
    - Dangling pointer
    - Memory corruption
  - Event notification
    - Deallocation/reallocation
  - Memory comparison between processes

CENTER FOR COMPUTATION
& TECHNOLOGY

Memory Debugging - Usage

- Need to link to the TotalView heap library to monitor heap status
  - The name of the library is platform dependent
- To access memory debugging functions
  - Prior to 8.7
    - Tools -> Memory debugging
  - Since 8.7
    - Debug -> Open MemoryScape

CENTER FOR COMPUTATION
& TECHNOLOGY

References

- TotalView user manual
  - http://www.totalviewtech.com/support/documentation/totalview/index.html
- LLNL TotalView tutorial
  - https://computing.llnl.gov/tutorials/totalview
- NCSA Cyberinfrastructure Tutor
  - “Debugging Serial and Parallel Codes” course
- HPCBugBase
  - http://hpcbugbase.org/index.php/Main_Page

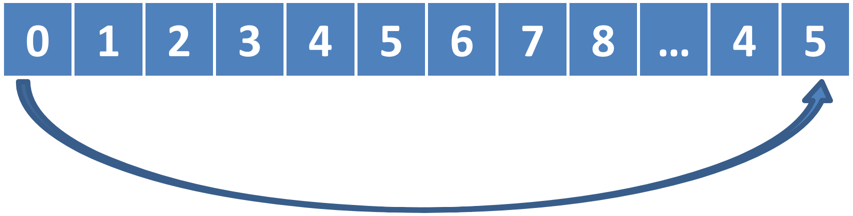
CENTER FOR COMPUTATION
& TECHNOLOGY

Hands-on Exercise

- Debug MPI and OpenMP programs that solve a simple problem to get familiar with
  - Basic functionalities of parallel debuggers
    - TotalView: BigRed, Kraken, Steele and Queen Bee
    - DDT: BigRed, Kraken, Ranger and Lonestar
  - Some common types of bugs in parallel programming
- Programs and instructions can be found at http://www.cct.lsu.edu/~lyan1/summerschool10

LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

Problem



0	1	2	3	4	5	6	7	8	…	4	5

- A 1-D periodic array with *N* elements
- Initial value
  - C: *cell(x)=x%10*
  - Fortran: *cell(x)=mod(x-1,10)*
- In each iteration, all elements are updated with the value of two adjacent elements:
  - $cell(x)_{i+1}=[cell(x-1)_i+cell(x+1)_i]\%10$
- Execute N_{iter} iterations
- The final outputs are the global maximum and average

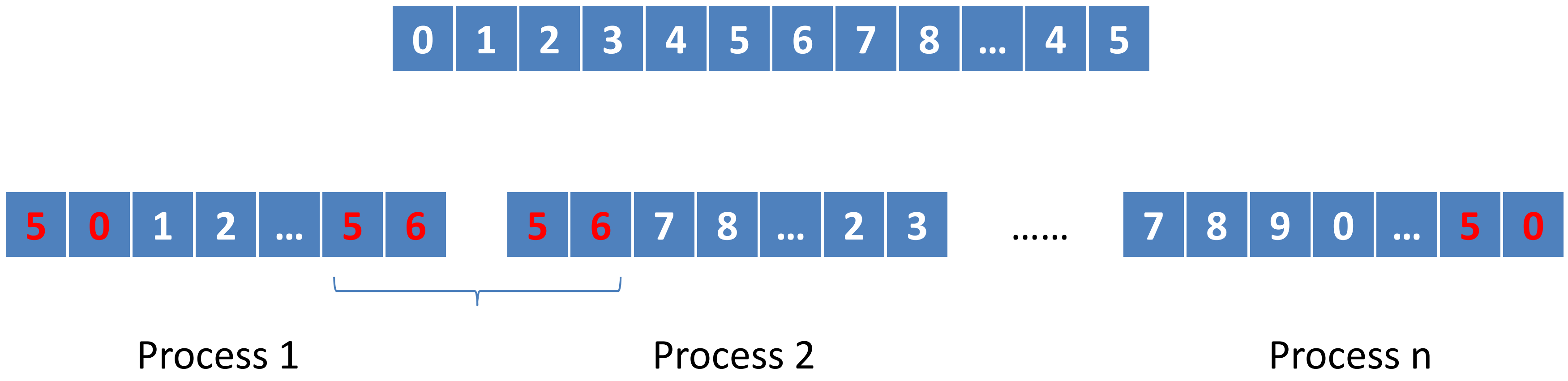
CENTER FOR COMPUTATION & TECHNOLOGY

http://www.hpcbugbase.org/index.php/Main_Page

Sequential Program

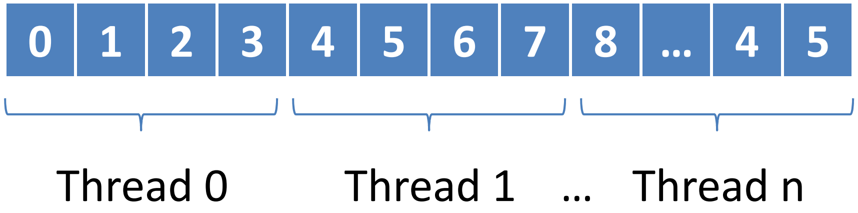
- Use an integer array to hold current values
- Use another integer array to hold the calculated values
- Swap the pointers at the end of each iteration
- The result is used to check the correctness of the parallel programs
  - Chances are that we will not have such a luxury for large jobs

MPI Program



- Divide the array among *n* processes
- Each process works on its local array
- Exchange boundary data with neighbor processes at the end of each iteration
- Ring topology

OpenMP Program



0	1	2	3	4	5	6	7	8	…	4	5

Thread 0 Thread 1 … Thread n

- Each thread works on its own part of the global array
- All threads have access to the entire array, so no data exchange is necessary

CENTER FOR COMPUTATION
& TECHNOLOGY