SP ScicomP
July, 2006

Boulder, CO

Memory Debugging on AIX BGL and Power/Linux with TotalView

Chris Gottbrath
Product Manager, Etnus

Chris.Gottbrath@etnus.com

About Etnus TotalView

- **Most advanced debugger on AIX, BlueGene/L, Power/Linux, Intel/Linux**
  - Memory Debugger
  - Parallel Debugger
- **Customers include:**
  - Major government research and academic labs worldwide
  - Software development companies
  - Companies in Finance, Entertainment, Telecommunications, Energy, Aerospace, Climate Modelling, Automotive
- **TotalView has been in development for 19 years under BBN, Dolphin, and Etnus**
- **Partner with IBM to provide C/C++ and Fortran debugging solution for complex codes on AIX, BGL, and Linux**
- **Collaboration with other vendors provides consistent debugging solution across most UNIX and Linux platforms**

Heap Memory

- **Heap is managed by the program**
  - C: Malloc() and free()
  - C++: New and Delete
  - Fortran90: Allocatable arrays
- **Malloc usage is something like:**

```
int * vp;
vp=malloc(sizeof(int)*number);
if (vp == 0){ /*malloc must have failed*/ }
/* use vp */
free(vp);
vp=0;
```

What is a Memory Bug?

- **A Memory Bug is a mistake in the management of heap memory**
  - Mistake: The program fails to follow the procedure defined in the heap allocation API
    - Failure to check for error conditions
    - Relying on nonstandard behavior
    - Leaking: Failing to free memory
    - Dangling references: Failing to clear pointers
    - Memory Corruption: Writing to memory not owned / Over running array bounds
  - Fallout: The program may then operate on an address in the heap based on an incorrect assumption about the allocation state of that address
    - Write/Read to a pointer pointing to a deallocated block
    - Read/Write to a pointer pointing to a block that has been deallocated and then reallocated (for a new purpose)
    - Leaked memory consumes a limited resource

Memory Problems in Clusters

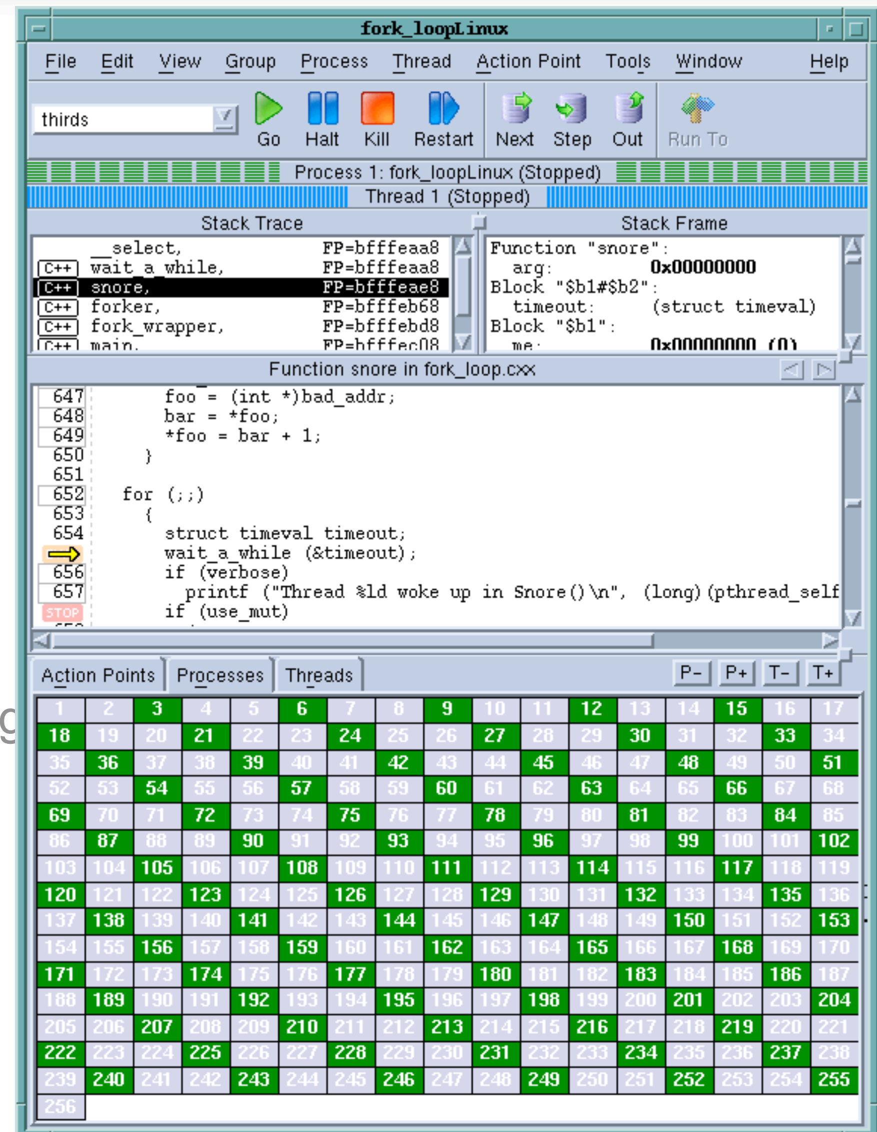
- **Moving an application to a cluster increases the problem complexity**
  - Distributed algorithms are more complex
  - Application data set size may push available memory even when everything is functioning correctly
  - Porting to cluster may involve moving to a new architecture/OS
- **Cluster Environment is different**
  - Many potentially useful memory tools aren't designed for use in a cluster
    - May simply fail
    - May require extreme 'workarounds'
  - Report based tools need cluster-aware filtering mechanisms

What is the solution?

- **Interactive debugging style**
  - Integrate memory debugging with general debugging practices
- **Tackle parallel memory problems in clusters with**
  - The Right Tools -- used together
    - Parallel Debugger
    - Memory Debugger
  - Experience to use tools effectively
- **The remainder of this talk covers Etnus's solution: TotalView and TotalView's integrated memory debugger.**

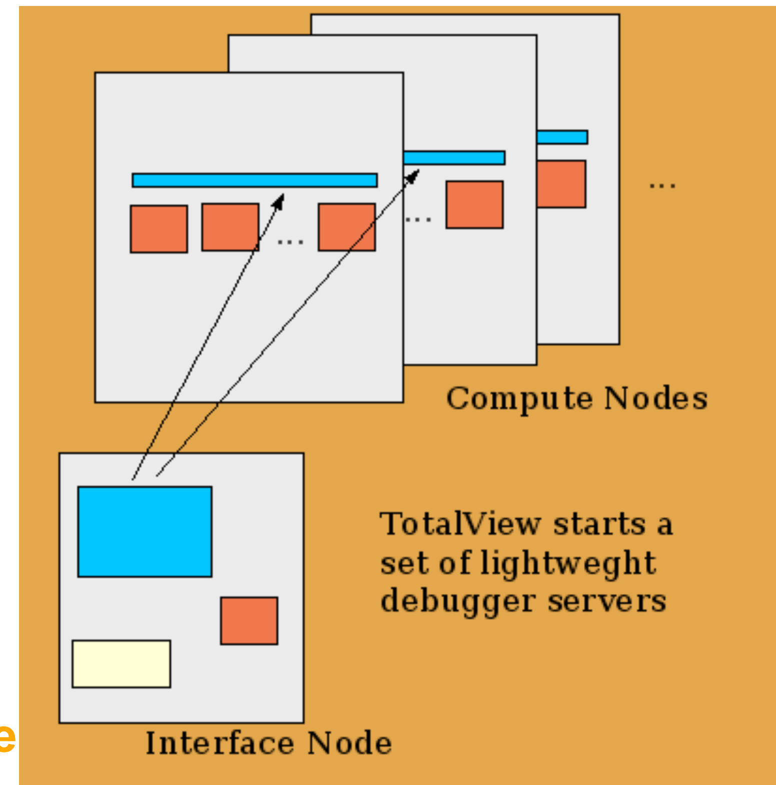
What is TotalView?

- **Source Code Debugger**
  - C, C++, Fortran 77, Fortran90
    - Complex language features
  - Wide compiler and platform support
    - BGL
    - Linux Power & Intel
    - AIX
  - Multi-threaded Debugging
  - Distributed (MPI) & Remote Debugg
  - Memory Debugging Capabilities
    - Integrated into the Debugger
  - Powerful and Easy GUI
    - Visualization



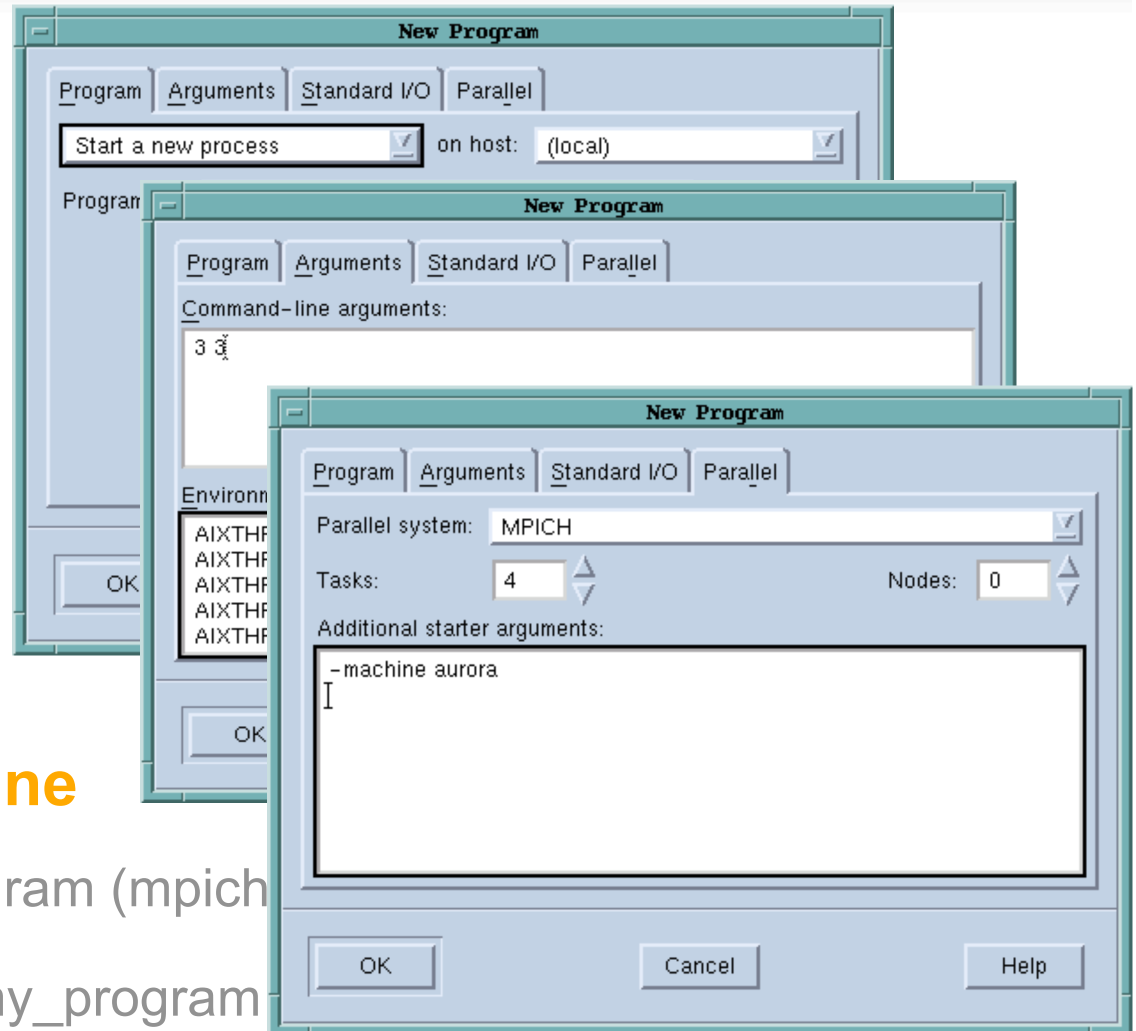
Architecture for Cluster Debugging

- **Cluster Architecture**
  - Single Front End (TotalView)
    - GUI and debug engine
  - Debugger Agents (tvdsvr)
    - Low overhead, 1 per node
    - Traces multiple rank processes
  - TotalView communicates directly with tvdsvrs
    - Not using MPI
    - Protocol optimization
- **Provides: Robust, Scalable, Minimal Interaction**
- **Memory Debugging fits into same mode**
  - HIA exists with in rank process
  - TotalView acquires data via tvdsvr



Starting MPI job within a debugger

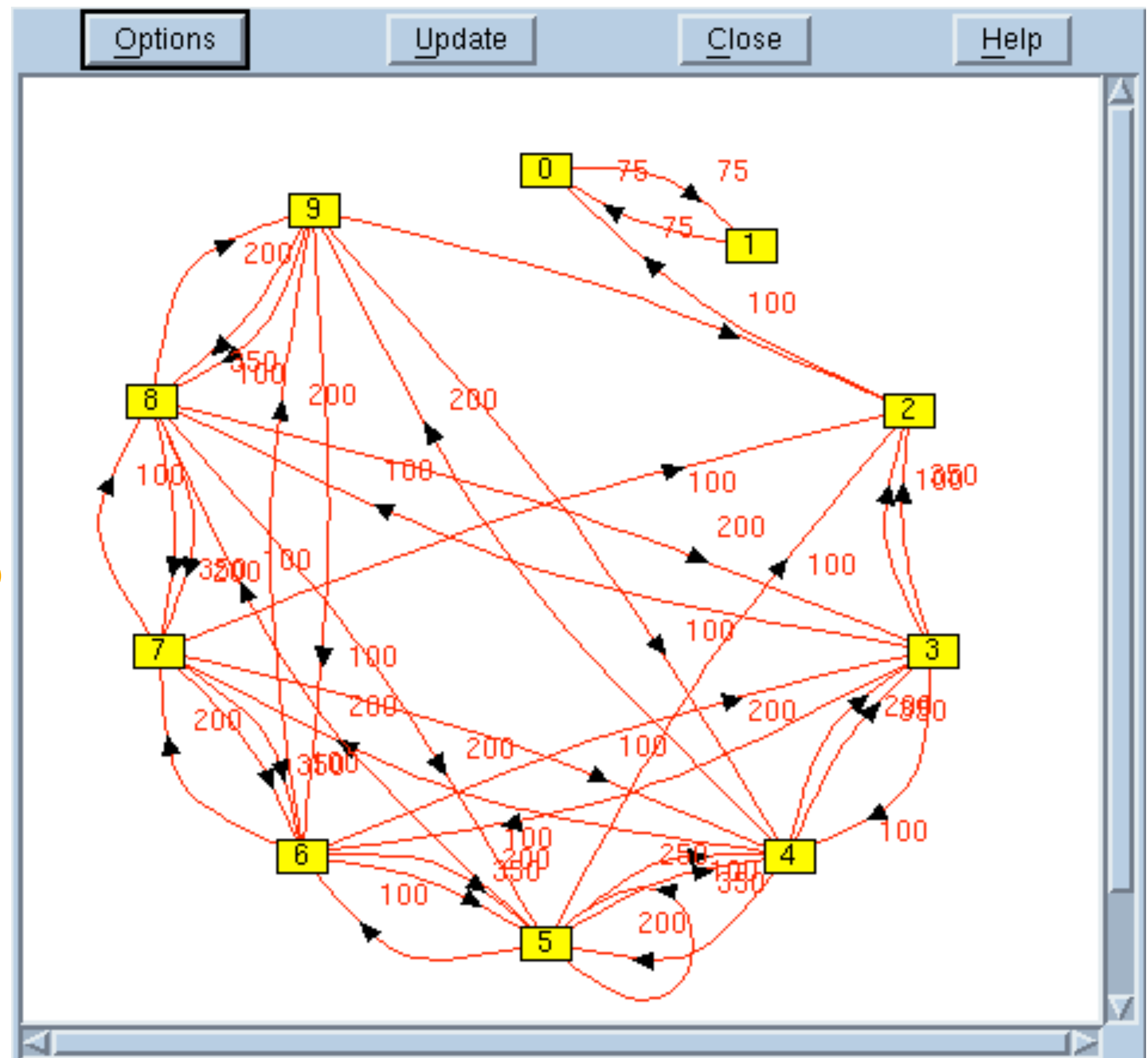
- **Indirect Launch**
  - Choose MPI implementation
  - Set parameters
  - Enable Memory Debugging
  - Add your own MPI
- **Start from command line**
  - mpirun -tv -np 4 my_program (mpich
  - totalview mpirun -a -np my_program



ETNUS TOTALVIEW

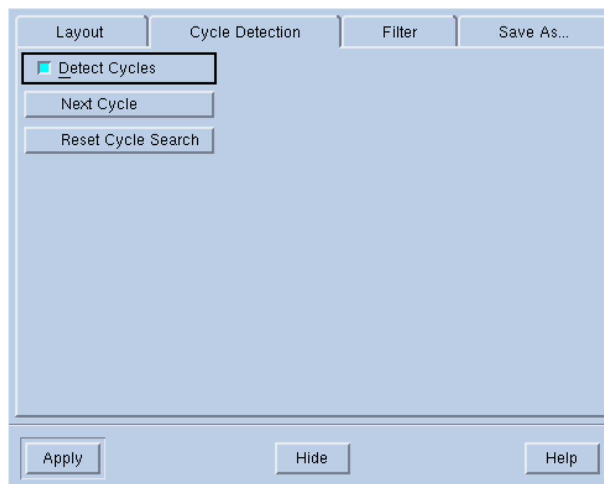
Message Queue Graph

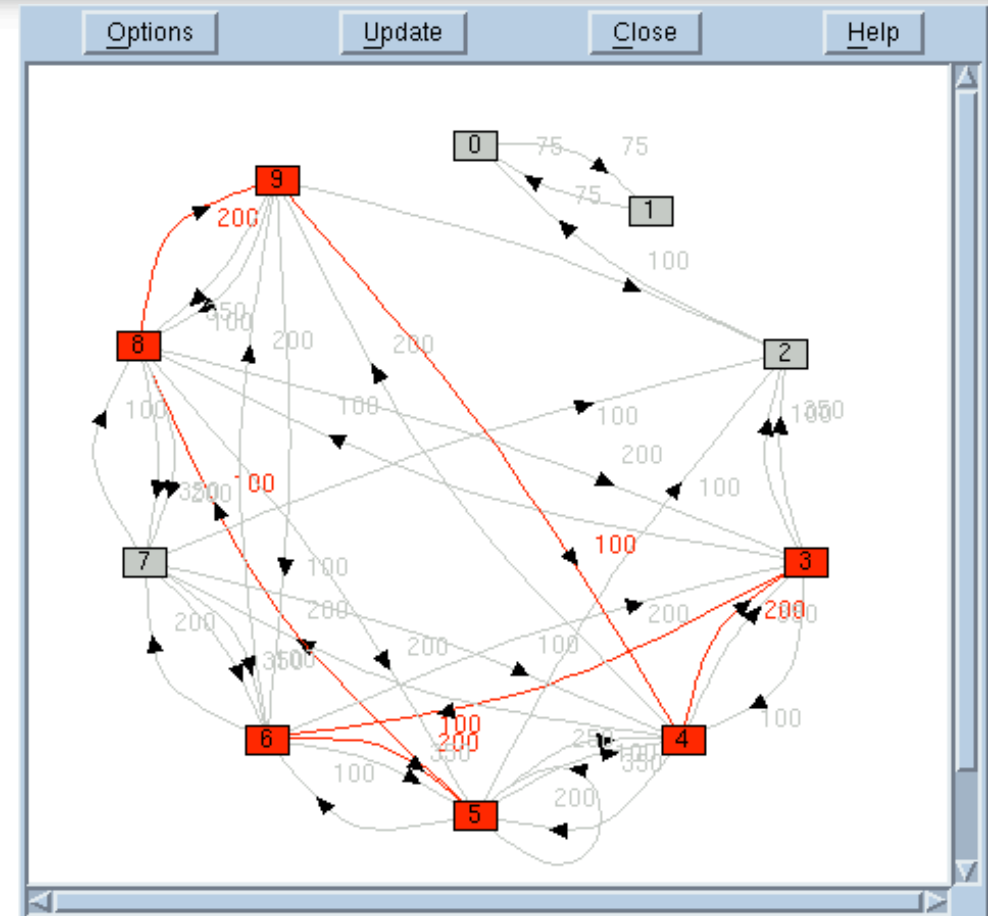
- **All pending messages**
  - Receives
  - Sends
  - Unexpected
- **Patterns are easy to spot.**



Message Queue Debugging

- **Filtering**
  - Tags
  - MPI Communicators
- **Cycle detection**
  - Find deadlocks





TotalView has an Integrated Memory Debugger

- **Built into the TotalView debugger**
- **Interactively investigate Memory State**
  - Look at memory issues in context of your running application
  - Solve memory problems during debug phase
- **Detect and solve**
  - Memory Leaks
  - Dangling Pointers
  - Malloc API usage errors (ie. double free)
  - Heap Corruption (dynamic arrays)

The Agent and Interposition

TotalView

Process

User Code and Libraries

Allocation Table

Heap Interposition Agent (HIA)

Deallocation Table

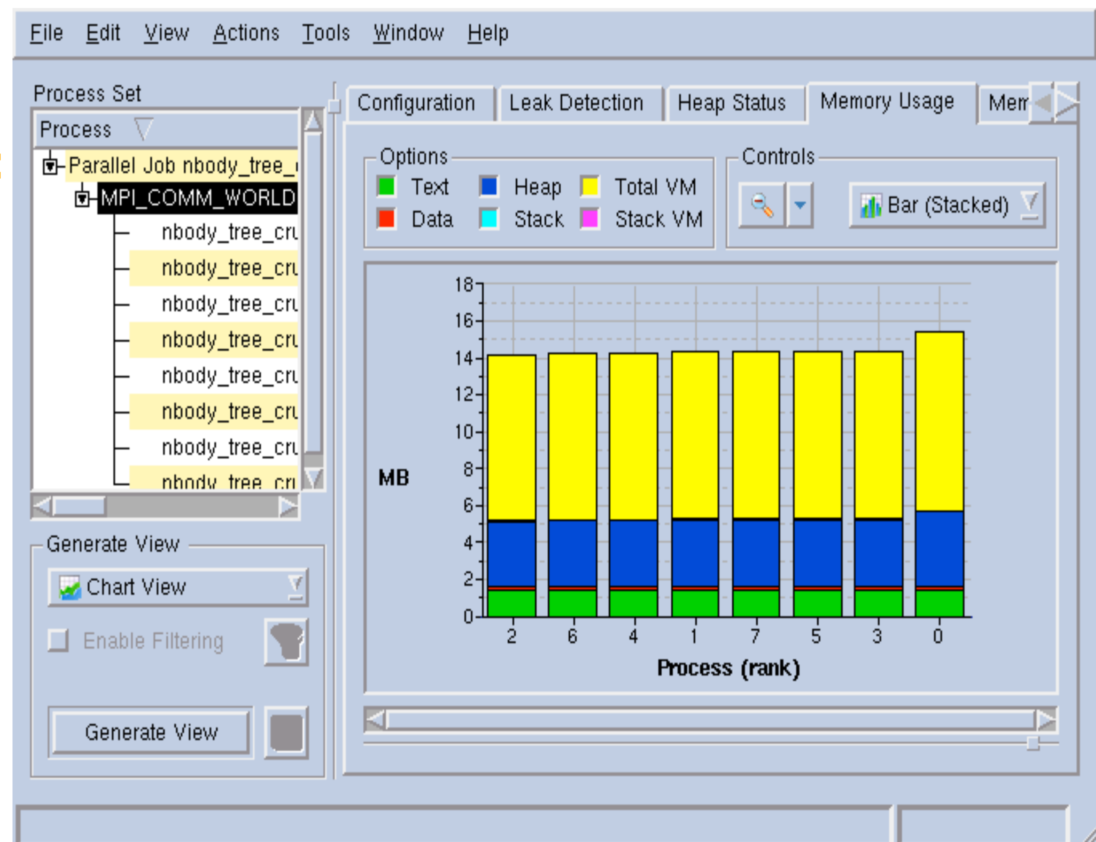
Malloc API

Preparing for Memory Debugging

- **Preparing the Cluster**
  - On AIX a special library must be created on each node
    - During installation and any time that libc is updated
    - AIX doesn't support 'preloading'
- **Launching the job**
  - Single process, MPICH MPI:
    - Just compile -g
  - Other MPI implementations (including poe) require that you link with the HIA.
- **Attaching to a process/job**
  - Requires linking or preloading the HIA.

Overall Memory Usage

- **Does not require enabling memory debugging**
- **View memory segment sizes:**
  - Text
  - Data
  - Stack
  - Heap
  - Total VM
  - Total Stack



Look at Memory Heap Patterns

- **Get a visual representation**
  - Overall heap usage and fragmentation visible at a glance
  - Available at request during life cycle of app
  - Leaked allocations can be marked
  - Select a block to get more information about that block and about related blocks

Interactive Memory Error Notification

- Sanity check calls to malloc(), free()
- TotalView detects the following errors:
  - Double **free()**
  - **free()** but not allocated
  - **realloc()** but not allocated
  - Address not at the start of a block
  - Guard corruptions on free
  - Malloc API errors

Dealing with heap and leak information

- Structured organization
- See the backtrace and source code for each allocation
- Interactive in live process or postmortem
- Leaks listed in red
- Sort by Address, Bytes or Counts
- Save to a file

Configuration | Leak Detection | Heap Status | Memory Usage | Memory Compare

Data Source: Allocations, Deallocations, Hoard

Options: Leaks, Baseline

Process	Bytes	Count	Begin Address	End Address
filterapp	164.32KB	793		
myClassB.cxx	151.50KB	771		
myClassB::init	131.00KB	259		
Line 36	128.00KB	256		
Line 33	3.00KB	3		
Block 3	1024	1	0x08071f00	0x080
Block 2	1024	1	0x08071af8	0x0807
Block 1	1024	1	0x08050ef0	0x0805
myClassB::myCla...	20.50KB	512		

Backtrace

ID	Function	Line #	Source Informatic
12			
	malloc	149	malloc_wrappers_
	myClassB::init	33	myClassB.cxx
	myClassB::myClassB	11	myClassB.cxx
	main	19	main.cxx
	__libc_start_main		libc.so.6
	_start		filterapp

Source: ./myClassB.cxx

```
29  }
30
31  void myClassB::init(void) {
32
33    b_pp = (int **) malloc (size * si
34
```

Painting and Hoarding for dangling pointer problems.

- **Painting**
  - Create arithmetic errors
  - Invalid values with in deallocated memory
- **Hoarding**
  - Delay freeing memory
  - Memory remains viable for a longer period of time

red_balls - main - 1.1

File Edit View Tools Window Help

1.1 More Less

Expression: red_balls Address: 0xbfffeb54

Type: snooker_ball_t *

Value
0x08049978 (Allocated) -> (snooker_ball_t)

red_balls - main - 1.1

File Edit View Tools Window Help

1.1 More Less

Expression: *(red_balls) Address: 0x08049978

Type: snooker_ball_t

Field	Type	Value
value	int	0xa110ca7f (-1592735105
x	double	-2.05181867705792e-149
y	double	-2.05181867705792e-149
spare	int	0xa110ca7f (-1592735105
colour	$string *	0xa110ca7f -> <Bad addre

red_balls - main - 1.1

File Edit View Tools Window Help

1.1 More Less

Expression: (red_balls)->x Address: 0x0804997c

Slice: [:] Filter:

Type: int[2]

Field	Value
[0]	0xa110ca7f (-1592735105)
[1]	0xa110ca7f (-1592735105)

Catch Pointers Writing Out of Bounds

- **Guard Blocks**
  - Catches writes past the allocation
  - Checked at
    - Deallocation
    - User Request
  - Lightweight
  - Turn on/off

The guard areas around a block have been overwritten, suggesting a bounds error

Block Information

0x0804c068 — 64 bytes — 0x0804c0a7

Status: Allocated

Flags: Operation in Progress

Block Backtrace Information

Select the desired tab below to see the block allocation or deallocation backtrace. Backtrace information may not always be available. Examine the Process Window to see the point at which the application stopped due to the event.

Backtrace

ID	Function	Line #	Source Information
4			
	malloc	149	malloc_wrappers_dlopen.c
	corrupt_data	77	main.cxx
	main	126	main.cxx
	__libc_start_main		libc.so.6
	_start		filterapp

Source /home/demouser/memory/main.cxx

```
73    size = 16;
74
75    // Allocate some arrays.
76    p0 = (int *) malloc( size * sizeof( int ) );
77    p1 = (int *) malloc( size * sizeof( int ) );
78    p2 = (int *) malloc( size * sizeof( int ) );
```

Point of Allocation | Point of Deallocation

Close | View in Block Properties window | Help

Memory Comparisons

- **"Diff" live processes or postmortem processes.**
- **Postmortem**
  - Core files
  - Saved memory debug files
    - See changes from point A to point B.

Memory Debugging

File Edit View Actions Tools Window Help

Process Set

Process

- Parallel Job fork_loopLinux (
  - fork_loopLinux
    - fork_loopLinux.1
      - fork_loopLinux
        - fork_loopLi
      - fork_loopLinux
    - fork_loopLinux.2
      - fork_loopLinux
    - fork_loopLinux.3
- filterapp(10/10/05 04:59
- filterapp(31163)
- free_doubleLinux(31161)

/fork_loopLinux.2.1
PID: 31131

Configuration | Leak Detection | Heap Status | Memory Usage | Memory Compare

Data Source: Allocations, Deallocations, Leaks, Hoard

Process Comparisons
Session 1: app(10/10/05 04:59 PM)
Session 2: filterapp(31163)
Reverse Diff

Process	Bytes Session 2	Bytes Session 1	Bytes Difference	Count Session 2	Count Session 1	Count Difference
filterapp	325.40KB	1612	323.83KB	1588	5	1583
assB.cxx	305.00KB	0	305.00KB	1544	0	1544
assA.cxx	9.00KB	0	9.00KB	18	0	18
stl_alloc.h	6.69KB	0	6.69KB	3	0	3
main.cxx	4.71KB	0	4.71KB	23	0	23
main.cxx	0	588	-588	0	3	-3
myClassA.cxx	0	1024	-1024	0	2	-2

Session 1 Source | Session 2 Source

Generate View

Source View

Enable Filtering

Generate View

Recent TotalView Improvemets

- **Memory**
  - Memory Debugging on BGL
  - Guard Blocks
  - Diff
- **General**
  - Support for Apple Macs (Power & Intel)
  - Raw Data Display Mode
  - Limited Integration with Eclipse
- **MPI**
  - Simplified MPI Launch Process
  - MPI Message Filtering and Cycle Detection
  - Improvements in memory debugging with MPI

Future Plans

- **TotalView Individual**
  - For laptops & small workstations
  - Easy to purchase & install
  - No memory debugging
- **TotalView 8 Program**
  - New licensing options
  - Web Store & Customer Portal
  - Support Forums
  - Contact me for details
- **Next Feature Release**
  - Breakpoint Improvements
  - MPI Improvements

Try TotalView with Your Code Today

- **Visit: www.etnus.com**
  - Get a free fully-functional 15 day demo
  - Online documentation and tips
- **Webcasts**
  - see www.etnus.com
- **Training**
  - contact sales@etnus.com
- **My contact info:**
  - chris.gottbrath@etnus.com