ScicomP

Austin

Aug 2004

Memory Debugging with TotalView on AIX and Linux/Power

Chris Gottbrath

Memory Debugging in AIX and Linux-Power Clusters

- **Intro: Define the problem and terms**
  - What are Memory bugs?
  - Why are they hard to solve?
- **Tools: TotalView and the Heap Interposition Agent**
  - What does it mean to be a Parallel Debugger?
  - What can the HIA do?
  - What is the TV Roadmap for Memory Debugging?
- **Strategies: HIA Usage and Tips**
  - General Strategies
  - Filling up memory
  - Rank process crashing
- **Example: Plugging a leak**
- **Conclusion**

ETNUS TOTALVIEW

Intro: Memory

- **Four kinds of memory**
  - **Text** – Memory used to store your program's machine code instructions
  - **Data** – Memory used for storing uninitialized and initialized data
  - **Heap** – Memory used for data allocated at runtime
    - This is the kind of memory that requires the most intensive management and is the focus of the rest of this talk
  - **Stack** – Memory used by the currently executing routine and all the routines in its backtrace

Intro: Heap Memory

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

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

ETNUS TOTALVIEW

Intro: 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 definied 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
  - 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

Intro: Why are they hard?

- **Memory problems can lurk**
  - For a given scale, or platform or problem they may be non-fatal
  - Libraries could be source of problem
- **The mistake and fallout can be widely separated**
  - The mistake is rarely fatal in and of itself
  - The fallout can occur at any subsequent memory access through a pointer
- **Potentially 'racy'**
  - Memory allocation pattern non-local
  - Even the fallout is not always fatal. It can result in data corruption which may or may not result in a subsequent crash
- **May be caused by or cause of a 'classical' bug**

Intro: Memory Problem 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
- **The Cluster Environement 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

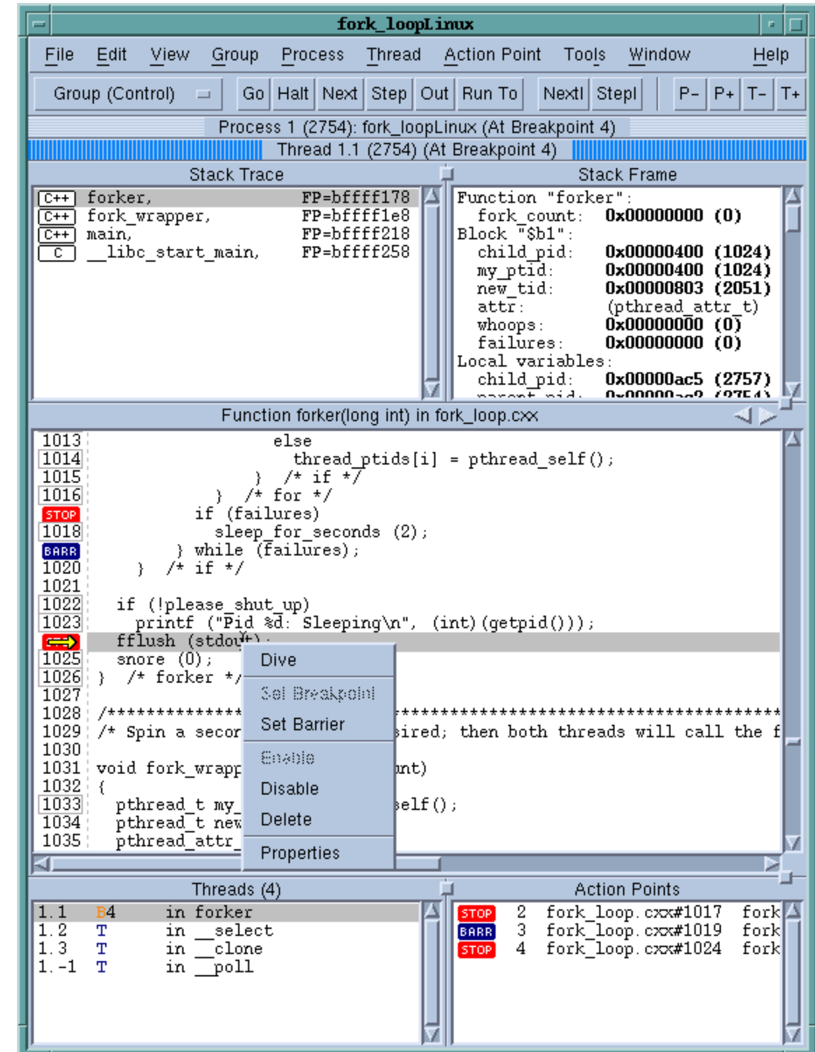
Intro: What is the solution?

- **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**
  - TotalView parallel and memory features
  - Strategies for successful debugging
  - An example debugging session

ETNUS TOTALVIEW

Tools: What is TotalView?

- **Source Code Debugger**
  - C, C++, Fortran, Fortran90
    - Complex language features
  - Wide compiler and platform support
  - Multithreaded debugging
    - Including OpenMP
  - Distributed Debugging
    - Cluster architecture
  - Memory Debugging Capabilities
    - Heap Interposition Agent
  - Powerful and Easy GUI
    - Visualization
  - Extensible via Scripting

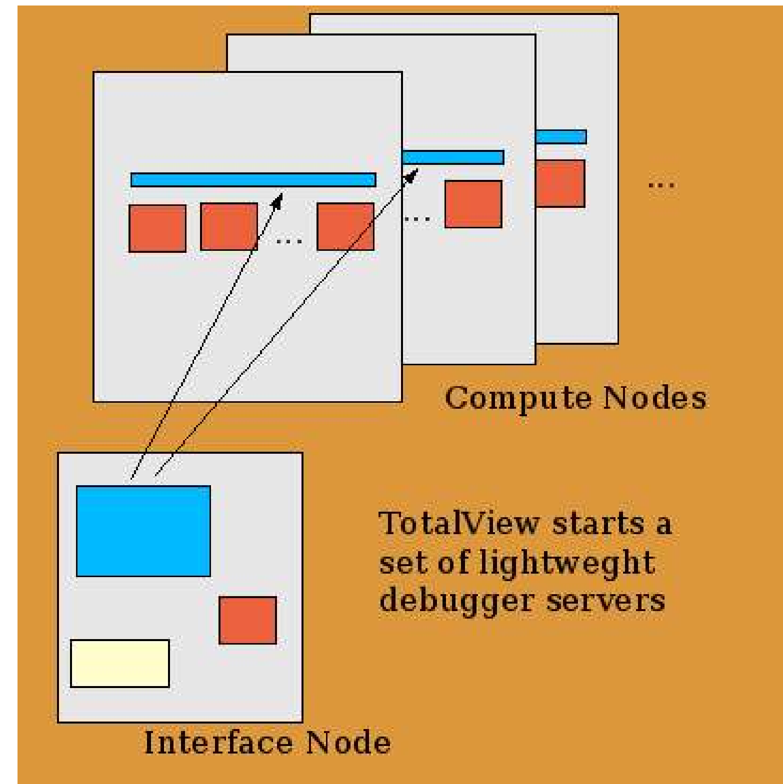


Tools: TotalView as Parallel Debugger

- **Cluster Architecture**
- **Process Aquisition**
- **Usability**
  - Status
  - Process Control
  - Data Exploration
- **MPI Message Queue Debugging**
- **Scalability**

Tools: Architecture for Cluster Debugging

- **Cluster Architecture**
  - Single Client (TotalView)
    - Heavy overhead
    - GUI and debug engine
  - Debugger Servers (tvdsvr)
    - Low overhead
    - 1 per node
    - Traces multiple rank processes
    - Runs as user
  - TotalView communicates directly with tvdsvrs
    - Not using MPI
    - Protocol optimization



- **Provides: Robust, Scalable, Minimal Interaction**

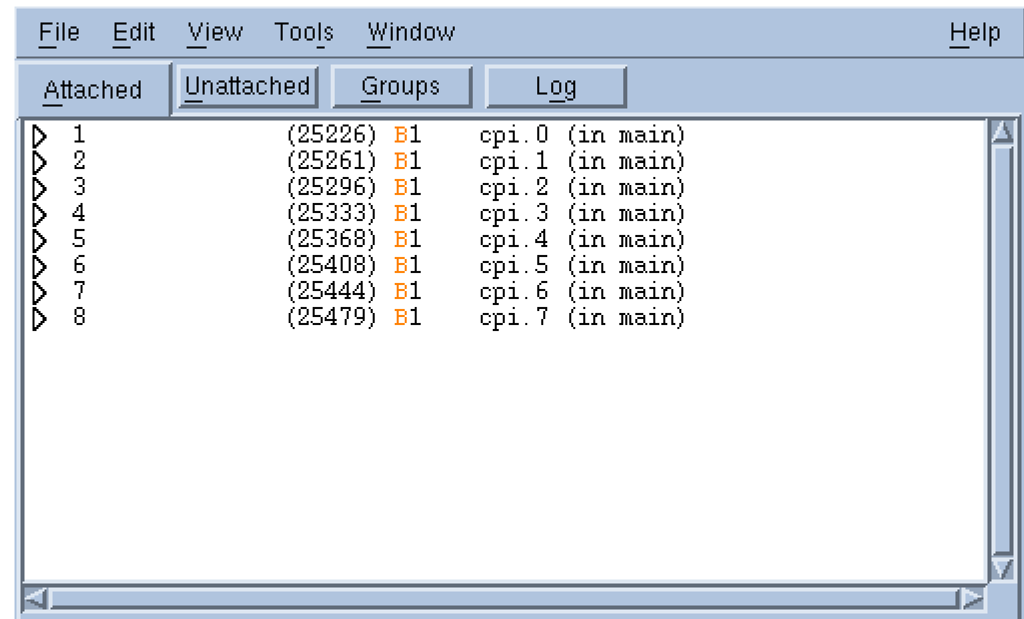
ETNUS TOTALVIEW

Tools: Process Aquisition

- **TotalView Process Aquisition**
  - Seamlessly attach to all the processes making up an MPI job
    - Jobs started via TotalView
    - Already running or hung job
  - Based on a public interface
    - Almost every MPI implementation provides support
  - Single Server Launch based on rsh
    - No special support needed
    - Drop in ssh as a secure replacement
  - Bulk Server Launch
    - Allows for faster launch if underlying support exists in the cluster environment (e.g. IBM POE)
  - Optionally attach to a subset

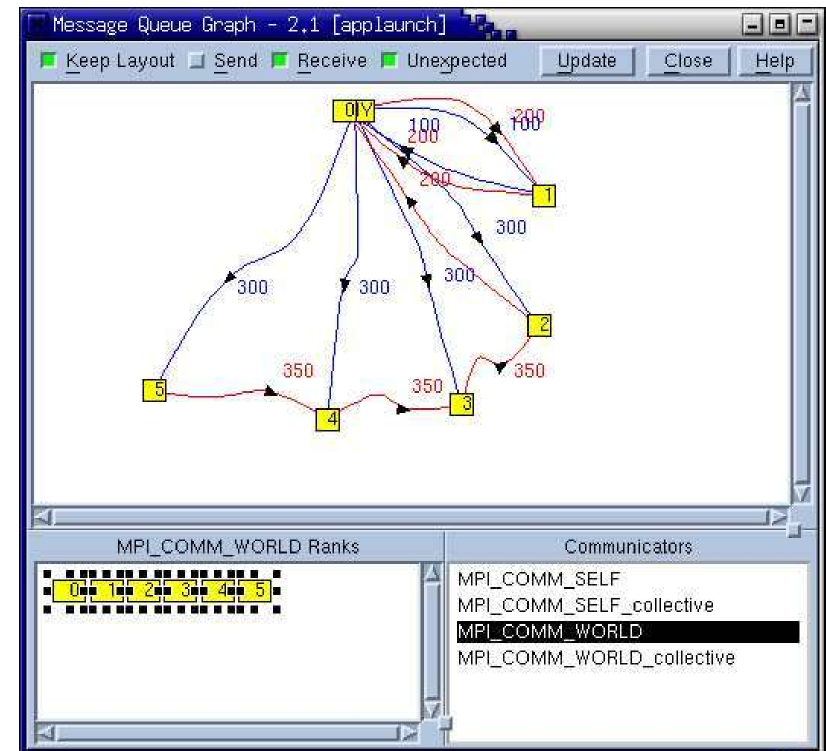
Tools: Core Parallel Functionality

- **The crucial thing in clusters is parallelism**
  - Parallelism touches the whole debugger interface
  - More states than just started and stopped
- **TotalView provides**
  - Automatic & manual process groups for process control
  - Root & Process window
    - Status information
    - Navgation
  - Rich set of action points
  - Parallel expression evaluation machanism
  - View SIMD data across all processes from one window
  - Asyncronous CLI



Tools: MPI Message Queue Information

- **Deadlocks**
  - MPI programs can suffer deadlocks
    - State information held in MPI library
  - TotalView can expose that information
    - Quickly debug deadlocks
    - Public interface that many MPI vendors support
  - Message Queue graph
    - Patterns easy to spot
    - Detail windows

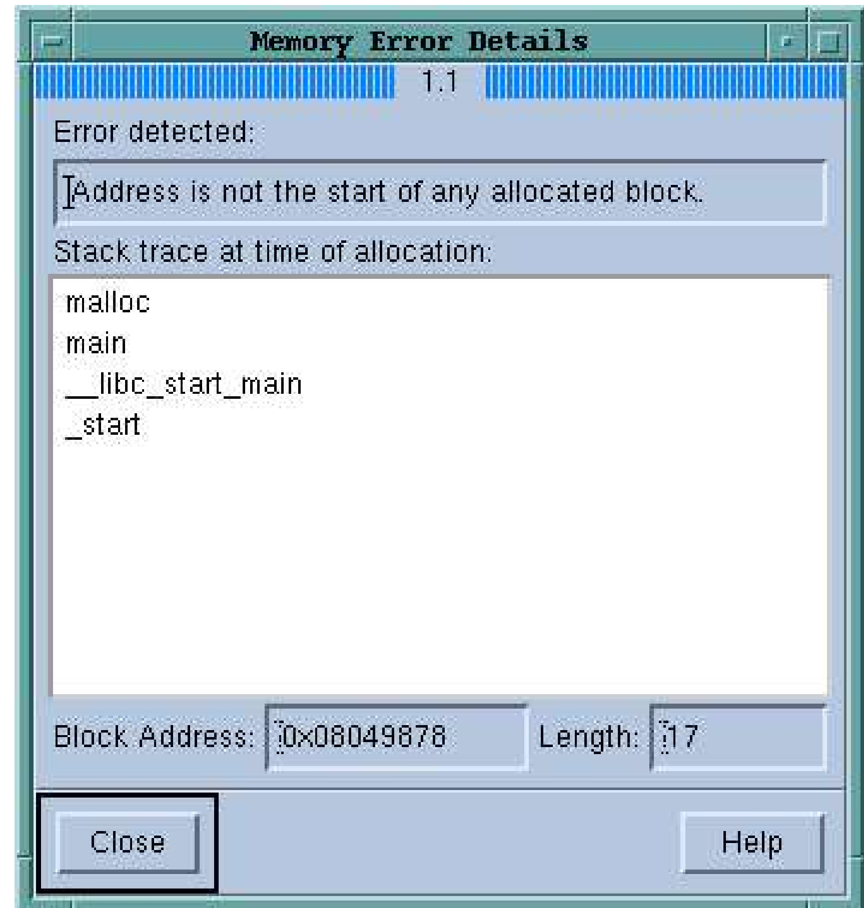


Tools: Scalability

- **Scalability means many things**
  - Startup and runtime performance / responsiveness
  - Memory usage
  - Status and data representation
  - Control Issues
  - Program size/complexity also grows
- **Practical scalability**
  - 10s of processes trivially
  - 100s of processes regularly
  - 1,000s of processes can be debugged currently with TotalView
  - More work on scalability as part of BG/L work
- **Features and strategies to work at scale**
  - Subset attach

Tools: TotalView as Memory Debugger

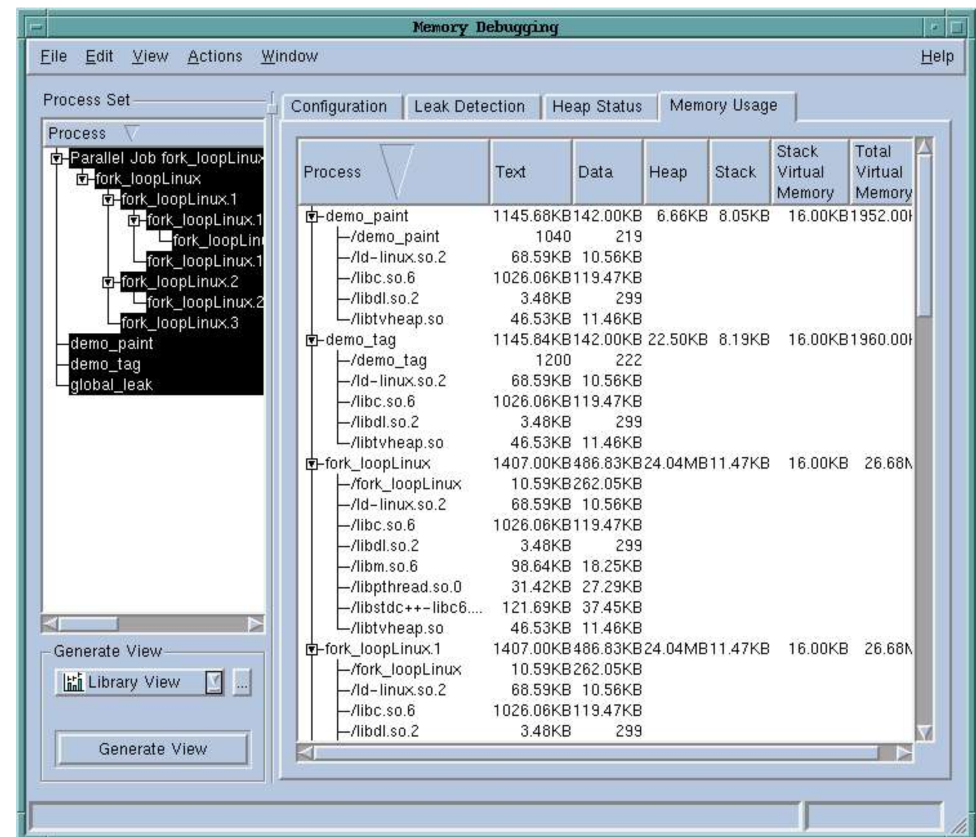
- **Parallel Memory Usage Statistics**
- **Heap Tracker**
  - Heap Interposition Technique
  - Capabilities
    - Protocol Violations Flagged at Runtime
    - Leak Detection
    - Dangling Pointer Annotation
    - Memory Painting
    - Event Notification
    - Memory Hording
  - Parallel and MPI Aware Interface



ETNUS TOTALVIEW

Tools: Memory Statistics

- **Memory Usage Statistics**
  - Gives overview of memory usage patterns
  - By process or library
  - Sortable
  - Filterable



Tools: Memory Tracker

- **The TotalView Memory Tracker**
  - Gets inserted into your program to provide instrumentation needed by TotalView
  - It maintains separate table of allocations that can be read by TotalView
  - Can take action at all points of allocation, re- and de-allocation
- **Interposed over malloc() calls**
  - Linked 'between' your program and malloc()
    - For parallel programs simple relinking
    - Can be used without relinking in many serial cases
  - Catches malloc() calls and return values in both your program and libraries
    - Checks values and builds table of allocations
  - If you have a custom malloc() you can continue to use it

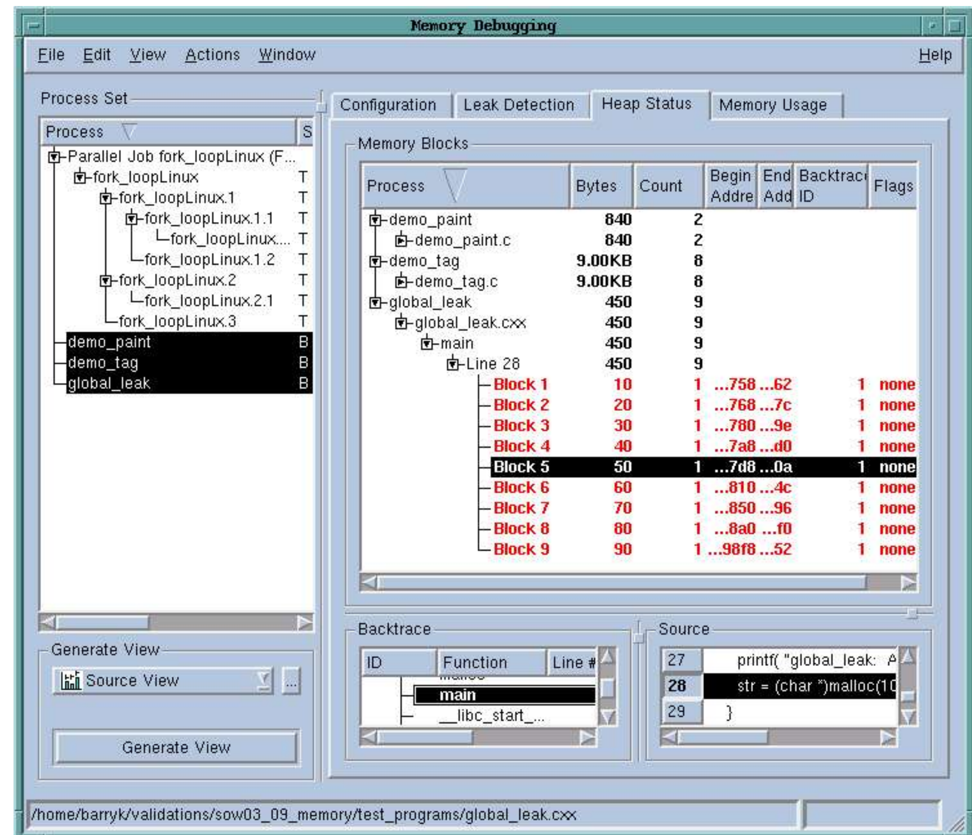
ETNUS TOTALVIEW

Tools: Heap Errors Flagged by Tracker

- **Example Heap allocation errors automatically detected**
  - **Free not allocated**
    - call to `free()` with an address that does not lie in any allocated block from the heap
  - **Realloc not allocated**
    - call to `realloc()` with an address that does not lie at any allocated block
  - **Address not at start of block**
    - `free()` or `realloc()` receive a heap address that does not lie at the start of any previously allocated block
  - **Double allocation**
    - An already allocated address is returned by a new request. Indicates a problem in the heap manager.
  - **Allocation request returns NULL**
    - A null value is returned by an allocation operation
- **Example Heap allocation errors not automatically detected**
  - Failure to call free()
    - No call site for error

Tools: Heap Information

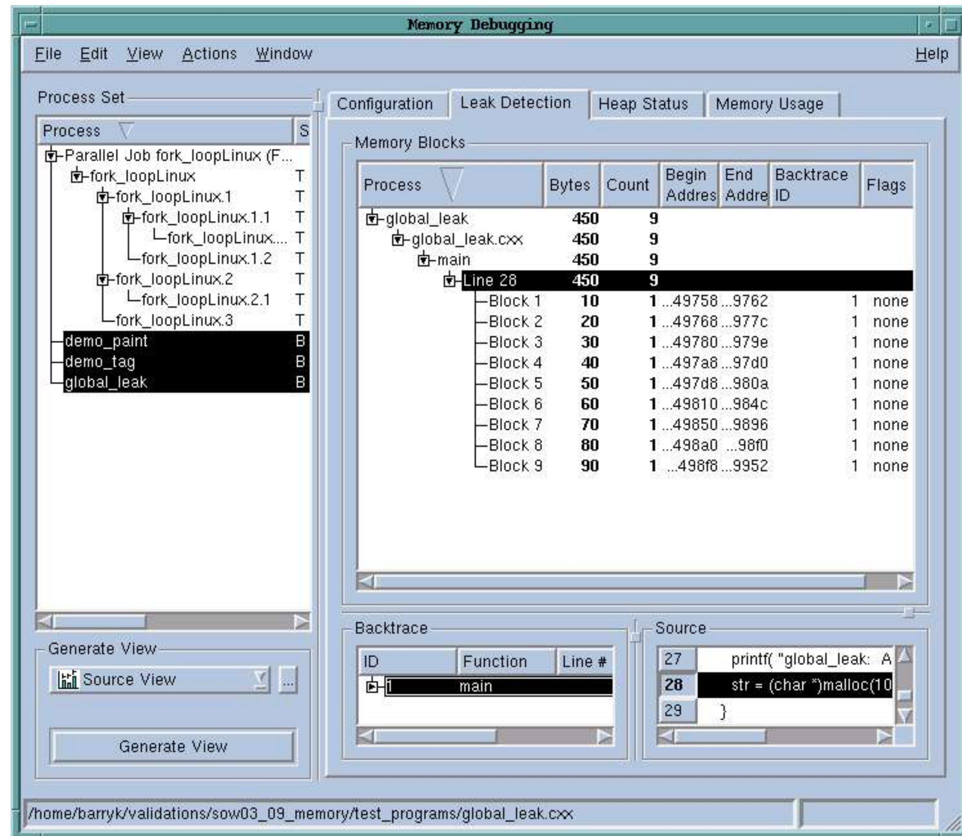
- **Shows all memory allocations in each process**
  - By source code location
  - By stack backtrace
- **Select processes**
- **Drill down by source structure**
- **For each block**
  - Stack and source code at point of allocation
- **If leak detection has been done leaks are highlighted**



ETNUS TOTALVIEW

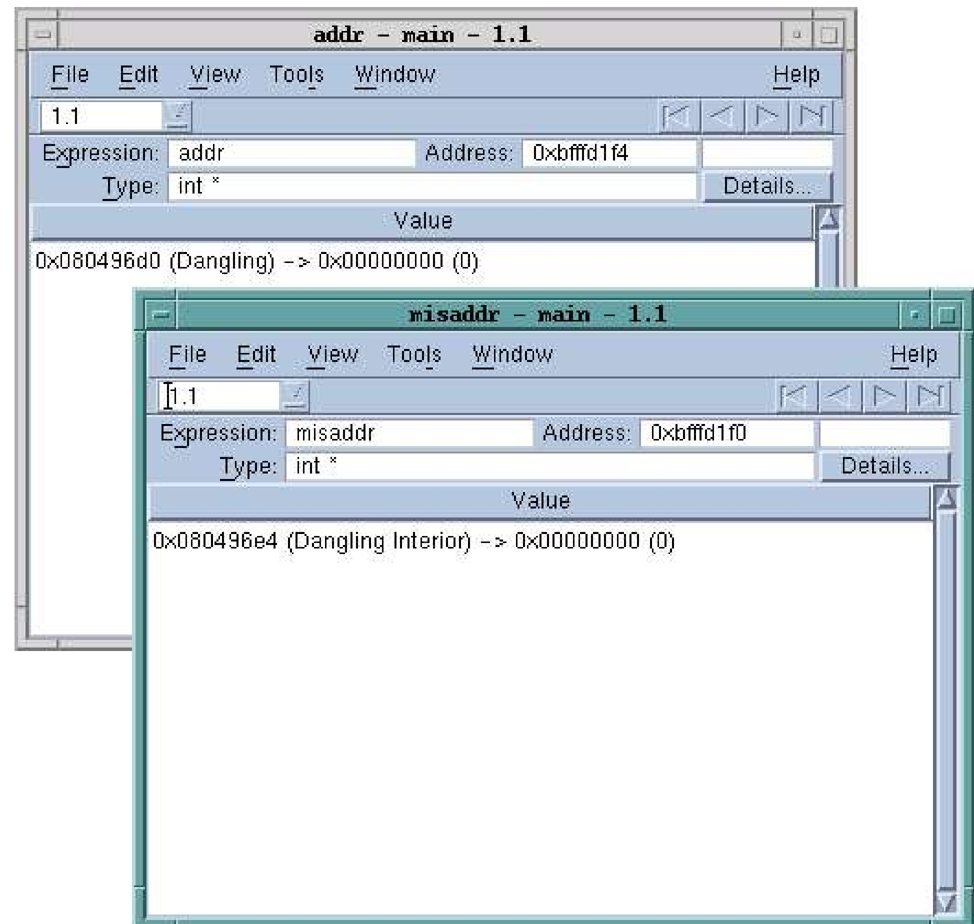
Tools: Leak detection

- **Leak : unreachable memory**
  - Garbage Collection algorithm
    - Examine all the pointers and registers in a program
    - Any memory allocations not reachable by any pointers is a leak
  - This is an expensive operation, initiated at user request
  - List of leaks is displayed just as the heap entries
  - False positives are possible



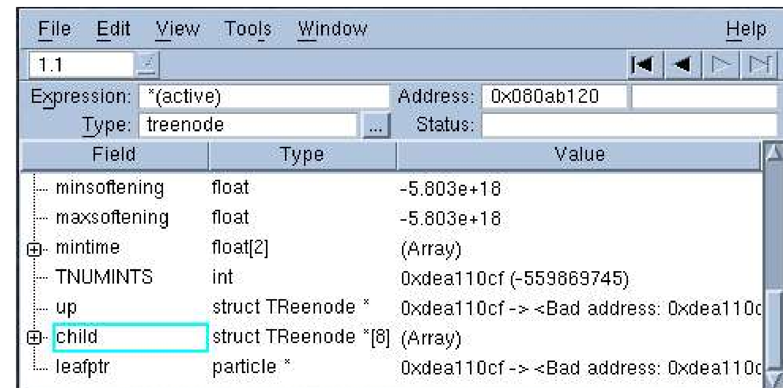
Tools: Dangling Pointer Detection

- **Dangling Pointer: pointer to unallocated memory**
  - TotalView annotates dangling pointers in the variable window when HIA is activated
    - May contain dangerously 'reasonable' looking data
  - Similarly, pointers are annotated “Allocated” and “Allocated Interior”



Tools: Memory Painting

- **The Heap Tracker can paint heap memory**
  - Allocated memory is normally returned with 'noise'
    - In some cases this noise looks like program data and can be hard to spot
  - Deallocated memory remains in the heap with old data intact
    - It will be marked dangling in TV but the program might still mistakenly operate on the data
  - Painting changes the data on allocation or deallocation
    - Easy to spot visually
    - Painted values point to invalid addresses
    - Painted values can be chosen to raise arithmetic errors
  - Change a subtle error into an obvious one



ETNUS TotalView

Tools: Event Notification and Hording

- **Notification of allocation events**
  - Request notification of heap allocation events related to a specific allocation
  - Allows a focused view of life cycle of a specific allocation
  - Conceptually similar to a watchpoint/breakpoint
- **Hoarding memory**
  - Prevents a certain bit of memory from being reallocated when it otherwise would
  - Preserves information about the allocation
  - Only function that changes allocation pattern

Tools: Using the Heap Tracker with AIX

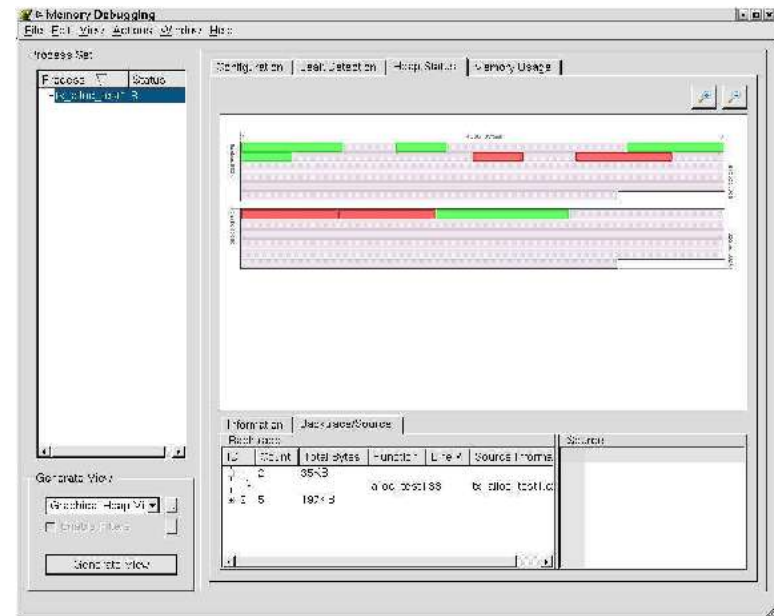
- **On AIX the HIA needs to be built against the system's C library**
  - AIX doesn't support pre-loading
  - The script aix_install_tvheap_mr in the TotalView installation makes this easy.
    - This needs to be run for each node in a cluster (use poe)
    - This needs to be rerun if the system library changes
- **Then your application needs to be linked with the HIA library**
  - For a 64 bit executable on AIX 5.X it is
    - mpcc_r -g $target.o -o $target -L $path_mr -L $path \ $path/aix_malloctype64_5.o
- **Then enable heap debugging in the TV GUI**
  - Turn on notification only for the poe task
    - Use the CLI and enter dheap -notify
- **There are other procedures, see the TV documentation.**
- **On Linux TotalView can use LD_PRELOAD interpose the HIA and relinking the executable is optional.**

Tools: TotalView Roadmap

- **TV 6.5.0 available now**
  - Available on AIX, x86, etc..
  - Parallel and Memory Debugging Features
- **Power-Linux Release Coming Soon**
  - Planned for a release later this year (4Q2004)
  - Support basic debugging features
    - Not Memory Debugging (initially)
    - No visualizer
- **Memory Debugging enhancements in 2005**
  - Added Power-Linux support
  - Enhancements to Memory Debugging for all platforms
    - Graphical view of heap allocations
    - Separately stored configuration files
    - Filters for memory debugging info
    - Pointer Allocation Information Enhancements
    - Heap API

Tools: Graphical View of Heap (future)

- **This will provide a visual representation of the Heap**
  - Overall heap usage and fragmentation visible at a glance
  - Leaked allocations would be marked
  - Image could be zoomed
  - Individual allocations could be selected as in the tree based report
  - Allocations matching some critera could be highlighted
- **The image to the right is a mock up**
  - The visual layout could change significantly



Tools: Memory Debugging Filters (future)

- **Allows the user to remove heap blocks matching certain critera from the heap status and leak report.**
  - Remove entries associated with a specific shared library
  - Remove entries based on block count or block size
  - Other critera like line number, pc, subroutine name
- **Multiple filters can be defined and toggeled on and off**
  - This allows the user to deal with large reports in an organized manner
  - Eliminate 'false positives' or leaks that have been understood

ETNUS TOTALVIEW

Tools: Pointer Allocation Info (future)

- **The additional information displayed for pointers in the heap in the data window will be extended**
  - Stack at point of allocation for an Allocated pointer
  - Stack at point of deallocation for a Dangling pointer
  - Status of notification for allocation and deallocation for the block being referenced
- **Similar information will be exposed in the dwhat command in the CLI**

Tools: API and Config Files (future)

- **HIA application program interface**
  - Allow target programs to use the information exposed by the HIA
    - The program can query HIA and perform checking based on heap status
      - Is this pointer allocated?
      - What is the current overall heap size?
  - The program can alter HIA settings
- **HIA Config Files**
  - More fine grained control of features
  - Will allow for the peristance of settings across sessions

Strategies: General Thoughts

- **Memory tracking is integrated with general debugging process**
  - Try to change a subtle error into a fatal one
    - The debugger will catch seg faults
    - Better for the fallout to be close to the error
  - Take advantage of the live process under the debugger
    - Look at context of error and/or fallout
  - The hypothesis testing cycle is vital
    - Use TotalView to steer problem and closely watch outcomes
    - Use painting and dangling pointer detection to confirm or rule out memory bug
  - CLI scripts can be used to monitor a long running application

Strategies: Filling up memory

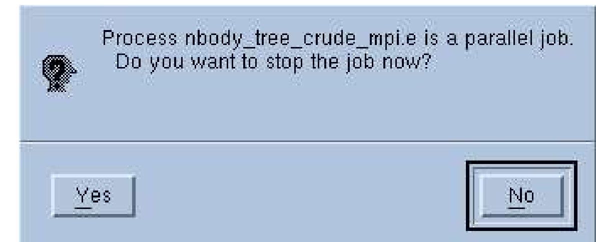
- **Scenario**
  - Processes in a parallel job are growing to fill available physical memory
- **Strategy**
  - Rebuild with tracker and rerun under TotalView
  - Watch heap usage with memory statistics window
  - Leak analysis with TotalView
- **Tips**
  - Leak report can only show point of allocation, you have to work out why they aren't getting deallocated
  - Heap table can be dumped (in the CLI) and compared before and after operations
  - Watch allocations with heap notification

Strategies: A rank process is crashing

- **Scenario:**
  - A rank process is crashing with a segv. Something is scribbling in the heap.
- **Strategy:**
  - Run the parallel job under TotalView with HIA
    - This will get you a stack trace
  - Examine the variable causing the segv
    - Is it dangling into a deallocated block?
  - Rerun to try to catch the scribbler in the act
    - Watchpoints on data locations being scribbled
    - Painting on allocation and deallocation
    - Painting and hoarding to change allocation pattern

Example: Patching a Leak

- **I have an bug in my MPICH program**
  - All of the rank processes grow to huge size
  - I'm going to show major steps in an example debugging session
- **First**
  - Rebuild the application (linux-x86 in this case) with
    - -L $tvlibdir -ltvheap -W1,-rpath,$tvlibdir
- **Next**
  - Run application with
    - mpirun $mpi_args -tv $programname
    - with poe this would be
      - totalview poe -a $poe_args $programname



Example: Launch and confirm leak

- **TotalView has automatically attached to rank processes**
  - 8 procs shown at right
- **Run to region of interest**
- **Start comparing memory statistics**

File Edit View Tools Window Help

Attached | Unattached | Groups | Log

1	(32746)	B1	nbody_tree_crude_mpi.e.0 (in MPIR_Br
2	(32752)	T	nbody_tree_crude_mpi.e.1 (in __GI_se
3	(32758)	T	nbody_tree_crude_mpi.e.2 (in __GI_se
4	(32764)	T	nbody_tree_crude_mpi.e.3 (in __GI_se
5	(302)	T	nbody_tree_crude_mpi.e.4 (in __GI_se
6	(308)	T	nbody_tree_crude_mpi.e.5 (in __GI_se
7	(314)	T	nbody_tree_crude_mpi.e.6 (in __GI_se
8	(320)	T	nbody_tree_crude_mpi.e.7 (in __GI_se

By Process | By Library

	Process	Text	Data	Heap	Stack	StackVm	
1	(32746)	1521.50K	314254	2560.05K	6980	28672	44
7	(314)	1431.00K	295697	2199.58K	20748	24576	39
5	(302)	1431.00K	295697	2191.58K	20784	24576	39
6	(308)	1431.00K	295697	2191.58K	20784	24576	39
8	(320)	1431.00K	295697	2191.58K	20784	24576	39
2	(32752)	1431.00K	295697	2135.58K	4316	24576	39
3	(32758)	1431.00K	295697	2127.58K	4940	24576	39
4	(32764)	1431.00K	295697	2127.58K	20784	24576	39

Focus: Group | Workers | 1.< | P/T Set:

Sort By: VmSize | Order: Descending | Auto Scale | Update | Cl

By Process | By Library

	Process	Text	Data	Heap	Stack	StackVm	VmSize
1	(32746)	1521.50K	314254	4856.05K	6784	28672	6392.00K
3	(32758)	1431.00K	295697	4431.58K	3392	24576	5848.00K
5	(302)	1431.00K	295697	4431.58K	3392	24576	5848.00K
6	(308)	1431.00K	295697	4431.58K	3392	24576	5848.00K
7	(314)	1431.00K	295697	4431.58K	3392	24576	5848.00K
8	(320)	1431.00K	295697	4431.58K	3392	24576	5848.00K
2	(32752)	1431.00K	295697	4359.58K	3392	24576	5812.00K
4	(32764)	1431.00K	295697	4359.58K	3392	24576	5812.00K

Focus: Group | Workers | 1.< | P/T Set:

Sort By: VmSize | Order: Descending | Auto Scale | Update | Close | Help

ETNUS
TOTALVIEW

Example: Heap Tracker Leak Detection

- **The leak report (for process number 5)**
  - Leaks classified according to stack when leaked memory was allocated
  - Many small leaks
  - Several groups
    - Same size
    - Same leaf function
    - Different called locations

```
d1.<>
d1.<>
d1.<> dfocus 5
d5.<
d5.<> dheap -leaks
process     5     (302):  total count 12083, total bytes 1449960
  * leak    1 -- total count 4102 (33.95%), total bytes 492240 (33.95%)
              -- smallest / largest / average leak:  120 / 120 / 120
        :   malloc            PC=0x40050c90 [/opt/toolworks/totalview.6.3.1-1/linux-x86/lib/libtvheap.
so]
        :   branch            PC=0x0804dc93 [oct_tree/tree.h#63]
        :   insert_to_tree    PC=0x0804e14c [oct_tree/tree.h#207]
        :   main              PC=0x080527e4 [nbody_tree_crude_mpi.c#552]
        :   __libc_start_main PC=0x4006ec64 [/lib/libc.so.6]
        :   _start            PC=0x08049e11 [/home/chrisg/work/ClusterWorld/src/etnus_demo_files/src/h
ybrid/Nbodycode/nbody_tree_crude_mpi.e]

  * leak    2 -- total count 3679 (30.45%), total bytes 441480 (30.45%)
              -- smallest / largest / average leak:  120 / 120 / 120
        :   malloc            PC=0x40050c90 [/opt/toolworks/totalview.6.3.1-1/linux-x86/lib/libtvheap.
so]
        :   branch            PC=0x0804dc93 [oct_tree/tree.h#63]
        :   insert_to_tree    PC=0x0804e1e1 [oct_tree/tree.h#220]
        :   main              PC=0x080527e4 [nbody_tree_crude_mpi.c#552]
        :   __libc_start_main PC=0x4006ec64 [/lib/libc.so.6]
        :   _start            PC=0x08049e11 [/home/chrisg/work/ClusterWorld/src/etnus_demo_files/src/h
ybrid/Nbodycode/nbody_tree_crude_mpi.e]

  * leak    3 -- total count 3461 (28.64%), total bytes 415320 (28.64%)
              -- smallest / largest / average leak:  120 / 120 / 120
```

Example: Examine Allocation

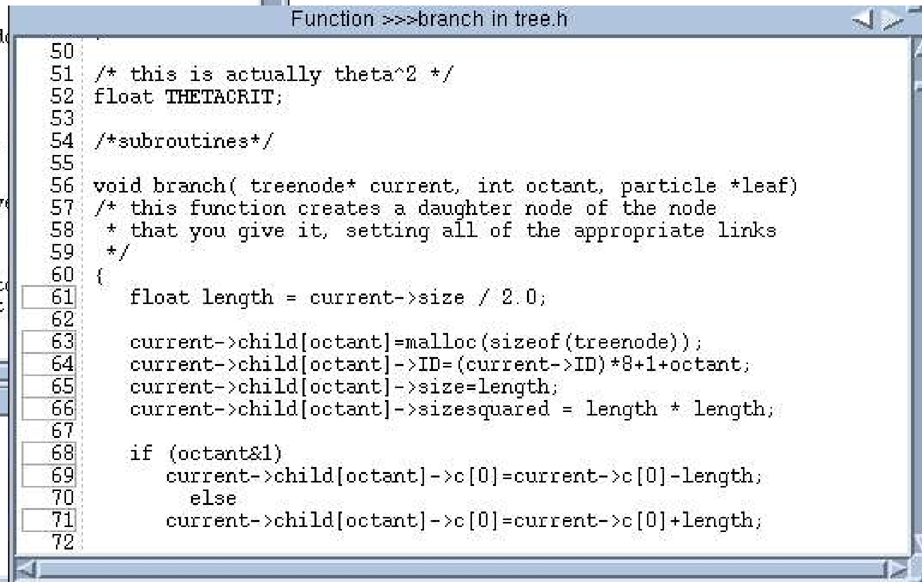
```
File  Edit  View  Group  Process  Thread  Action Point  Tools  Window  Help
Group (Control)  Go  Halt  Next  Step  Out  Run To  NextI  StepI  P-  P+  T-  T+
Process 5 (302): nbody_tree_crude_mpi.e.4 (At Breakpoint 2)
Thread 5.1 (302) (At Breakpoint 2)

Stack Trace
C  main,               FP=bffffc78
   __libc_start_main,  FP=bffffc98

Stack Frame
Function "main":
  argc:        0x00000001 (1)
  argv:        0x0809da18 (Alloc
Local variables:
  allparts:    0x080a97c0 (Alloc
  myparts:     0x080bce18 (Alloc
  green:       (tree)
  currtime:    0.54
  step:        0x00000036 (54)
  go_on:       0x00000001 (1)
  max:         3.0916

Function >>>>insert_to_tree in tree.h
199     {
200       active_oct = select_octant( current, toadd );
201
202       if ( current->child[active_oct] == NULL )
203       {
204         if ( slidingdown == NULL )
205         {
206           branch( current, active_oct, toadd
207           placed = 1;
208         }
209         else
210         {
211           if (impending_collision( current,
212           {
213             bud( current, active_oct );
214             current = current->child[active
215           }
216           else
217           {
218             branch( current, active_oct, t
219             branch( current, select_octant
220             placed = 1;
221           }

Thread (1)
5.1   B2   in main
BARR   2
```

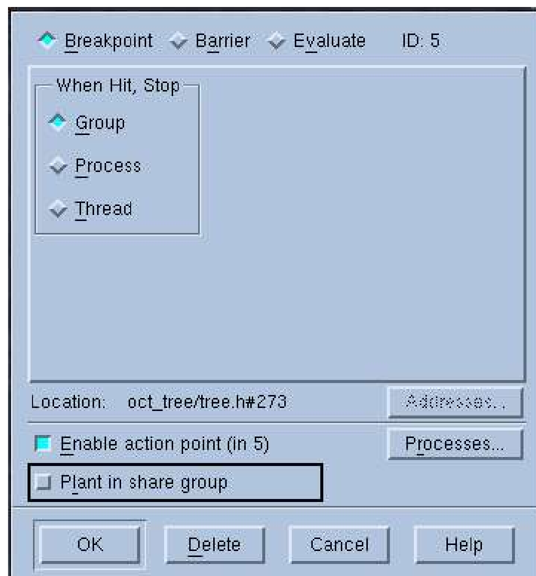


- **The allocations are occuring from various calls to branch() that occur in insert_to_tree()**
  - The leak seems to be occuring to all of these allocations
  - Where are these allocations dealloated?

Example: Examine point of deallocation

ETNUS TOTALVIEW

- **All deallocations occur here**
  - Recusive
  - Set breakpoint to watch what is happening
  - Focus on one process



File Edit View Group Process Thread Action Point Tools Window Help

Group (Control) Go Halt Next Step Out Run To NextI StepI P- P+ T- T+

Process 5 (302): nbody_tree_crude_mpi.e.4 (At Breakpoint 2)

Thread 5.1 (302) (At Breakpoint 2)

Stack Trace

```
C  main,               FP=bffffc78
   __libc_start_main,  FP=bffffc98
```

Stack Frame

```
Function "main":
  argc:       0x00000001 (1)
  argv:       0x0809da18 (Alloc
Local variables:
  allparts:   0x080a97c0 (Alloc
  myparts:    0x080bce18 (Alloc
  green:      (tree)
  currtime:   0.54
  step:       0x00000036 (54)
```

Function >>>>>trimbranch in tree.h

```
254  int trimbranch(treenode* active)
255  /* always clean up after yourself.. this
256   * gets rid of all of its children
257   */
258  {
259     int i;
260     int num = 0;
261
262     for(i=1;i<8;i++)
263     {
264        if( (active->child[i])!=NULL )
265           num+=trimbranch(active->child[i]);
266     }
267
268     #ifdef DBUG
269        printf("freeing %d:",active->ID);
270        printf("and %d children\n",num);
271     #endif
272
STOP    free(active);
274     num++;
275
276     return num;
277  }
278
```

Thread (1)

```
5.1   B2   in main
```

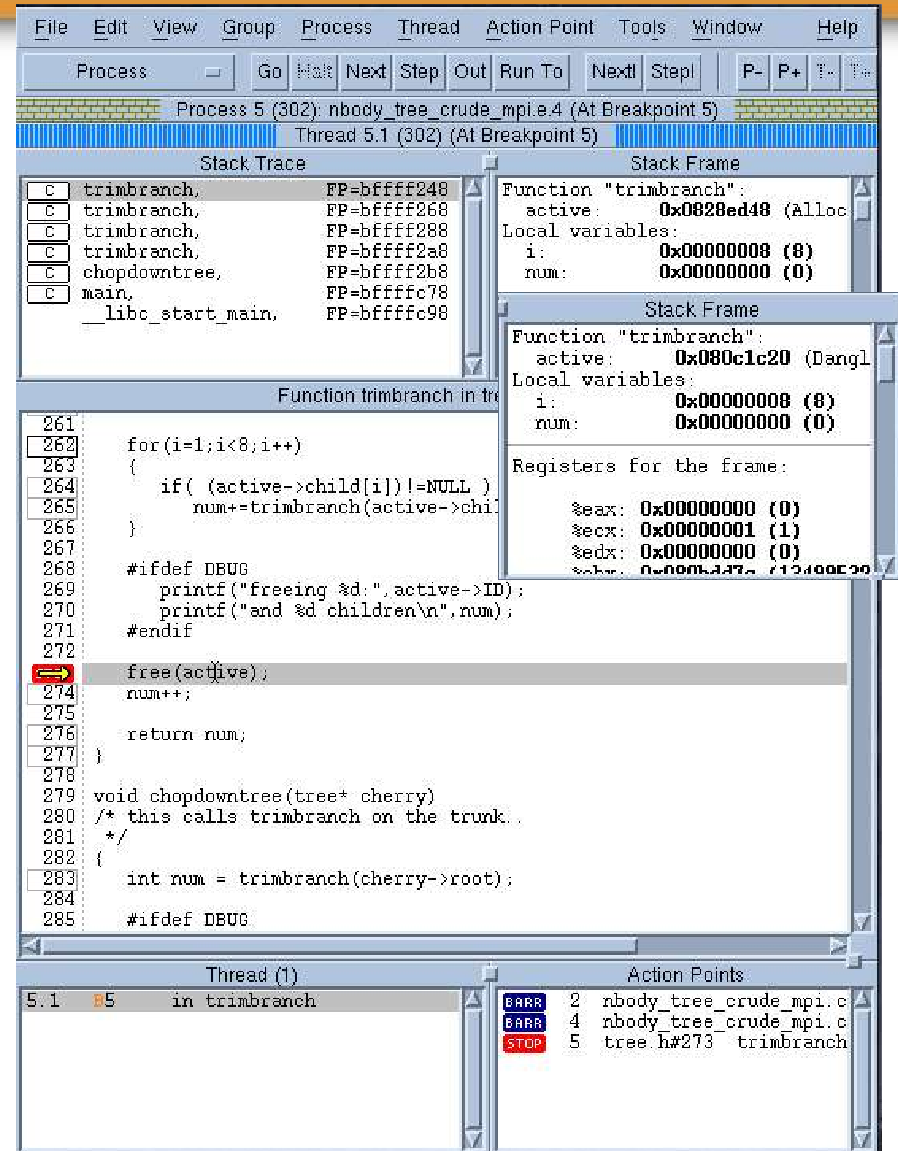
Action Points

```
BARR  2  nbody_tree_crude_mpi.c
STOP  3  tree.h#273  trimbranch
```

ETNUS TOTALVIEW

Example: Observe the deallocation

- **Watch variable 'active'**
  - Is it getting deallocated?
  - Are its children getting deallocated?
- **Watch several steps**
  - Ha!



tmass	float	7e-05
tke	float	1.31459e-05
tpot	float	-4.13274e-08
CRITDIST	float	3.73358
softdifference	float	0
softproduct	float	1
minsoftening	float	1
maxsoftening	float	1
mintime	float[2]	(Array)
[0]		54.8624
[1]		1.04278
TNUMINTS	int	0x00000239 (569)
up	struct TReenode *	0x08270d48 (Allocated) -> (
child	struct TReenode *[8]	(Array)
[0]		0x0828f7c8 (Allocated) -> (
[1]		0x0828ed48 (Dangling Interi
[2]		0x0828eac8 (Dangling Interi
[3]		0x0827c9c8 (Dangling Interi
[4]		0x08272ac8 (Dangling Interi
[5]		0x0828d848 (Dangling Interi
[6]		0x00000000
[7]		0x082847c8 (Dangling Interi
leafptr	particle *	0x00000000

Example: Confirmation

- **TotalView can test small changes without recompilation**

Breakpoint Barrier Evaluate ID: 6

Expression:

```
i=0;
if((active->child[i])!=0)
  num=num+trimbranch(active->child[i]);
```

C Fortran Assembler

Location: oct_tree/tree.h#262 Addresses...

Enable action point Processes...

Plant in share group

OK Delete

File Edit View Group Process Thread Action Point Tools Window Help

Group (Control) Go Halt Next Step Out Run To NextI StepI P- P+ T- T+

Process 18 (1248): nbody_tree_crude_mpi.e.3 (Stopped)

Thread 18.1 (1248) (Stopped) <Trace Trap>

Stack Trace

```
C  trimbranch,        FP=bffff230
   PC: bffff250,      FP=bffff248
C  trimbranch,        FP=bffff26c
   PC: bffff28c,      FP=bffff284
C  trimbranch,        FP=bffff2a8
C  chopdowntree,      FP=bffff2b8
C  main,              FP=bffffc78
   __libc_start_main, FP=bffffc98
```

Stack Frame

```
Function "trimbranch":
  active:      0x080c8b18 (Alloc
Local variables:
  i:           0x00000000 (0)
  num:         0x00000000 (0)
Registers for the frame:
  %eax:  0xbffff22c (-1073745
```

Function trimbranch in tree.h

```
250          }
251      }
252  }
253
254  int trimbranch(treenode* active)
255  /* always clean up after yourself.. this
256   * gets rid of all of its children
257   */
258  {
259      int i;
260      int num = 0;
261
262      for(i=1;i<8;i++)
263      {
264         if( (active->child[i])!=NULL )
265            num+=trimbranch(active->child[i]);
266      }
267
268      #ifdef DBUG
269         printf("freeing %d:",active->ID);
270         printf("and %d children\n",num);
271      #endif
272
273      free(active);
274      num++;
```

Thread (1)

18.1 T in trimbranch

Action Points

BARR	2	nbody_tree_crude_mpi.c	
BARR	4	nbody_tree_crude_mpi.c	
EVAL	6	tree.h#262	trimbranch
STOP	5	tree.h#273	trimbranch

```
d5.<> f 18
d18.<
d18.<> dheap -leaks
process    18    (1248):  total count 0, total bytes 0

d18.<>
```

Conclusion

- **Reviewed the characteristics of memory problems**
- **Proposed interactive debugging approach**
  - Integrate memory debugging with general debugging practice
- **Discussed the capabilities of TotalView**
  - Parallel Debugger
  - Memory Debugger
- **Suggested strategies for tackling memory bugs with TotalView**
- **Looked closely at tracking down a leaky MPI program**
- **For more information see www.etnus.com**
- **If you are interested in being a beta tester for TotalView on Linux-Power and/or for the upcoming memory debugging enhancements contact us at support@etnus.com**