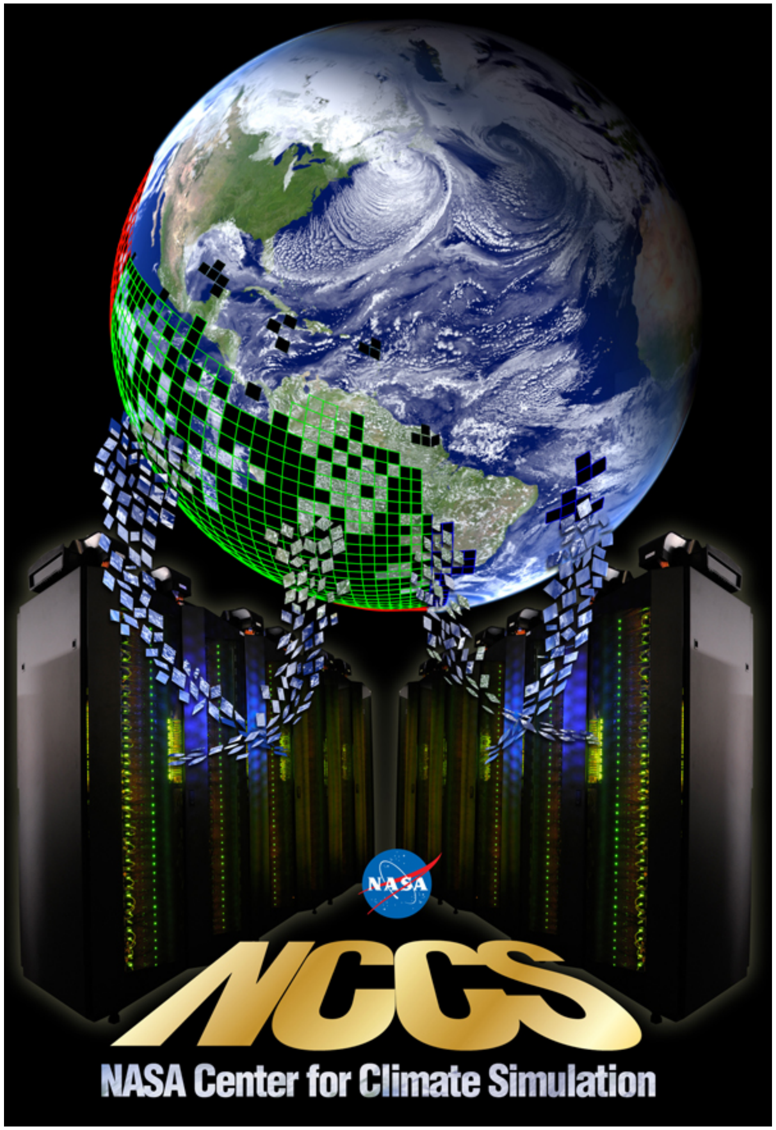


NCCS Brown Bag Series

TotalView on Discover:

Part 2

ReplayEngine and MemoryScape

Chongxun (Doris) Pan
doris.pan@nasa.gov
June 14, 2012

Agenda For Part 1 (see previous tutorial)

- Overview
- Basic Navigation and Control
- Demo 1: Basic Navigation and Control
  - ❖ Using an OpenMP Space weather application
- Debugging MPI Applications
- Demo 2: MPI Debugging:
  - ❖ Using GEOS5-AGCM

Agenda For Part 2

- Overview
- Reverse Debugging with ReplayEngine
- Demo 1: Replay Engine
  - Using an OpenMP Space weather application
- Memory Debugging with MemoryScape
- Demo 2: MemoryScape
  - Using GEOS AGCM

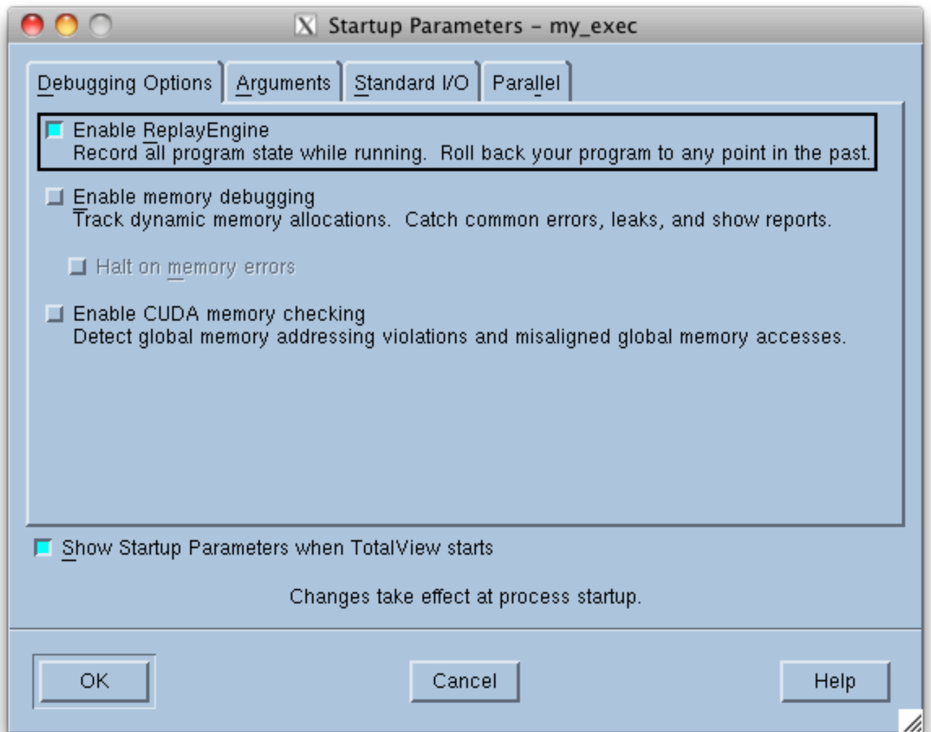
What is ReplayEngine ?

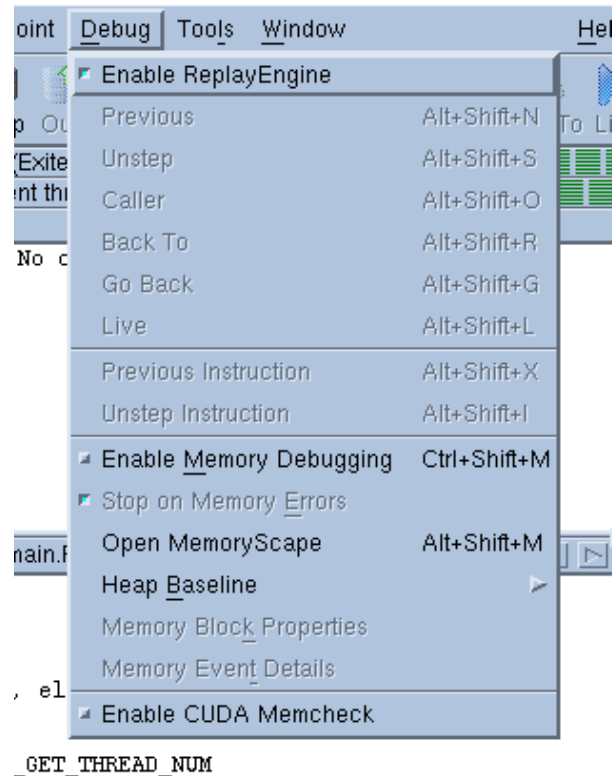
- Reverse Debugging
  - Let you step backward by function, line, or instruction
  - Capture and deterministically replay executions
- Major features
  - No recompilation
  - Designed to be used for parallel applications
- Why use ReplayEngine?
  - Eliminate restart cycle and hard-to-reproduce bugs

ReplayEngine

- **Record Mode** (saving execution history)
  - ❖ Capture the order of executions and changes to the data
- **Replay Mode** (when using a ReplayEngine command)
  - ❖ Like a “rewind” button on a DVR
- **Implications**
  - ❖ To ensure deterministic execution trajectory, it forces single thread execution at a time
  - ❖ During Replay mode, some operations (i.e., setting variables or full asynchronous thread control) are not available
  - ❖ Record mode could incur significant overhead (both in execution time and memory usage)

Enable ReplayEngine



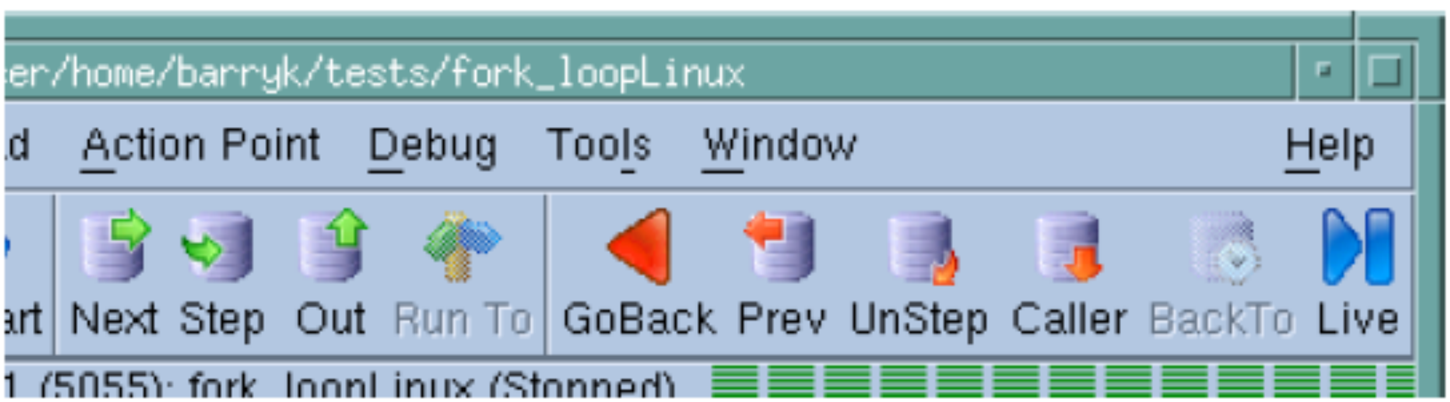


1. Starting Totalview with ReplayEngine enabled

2. After launching totalview, click the "Enable ReplayEngine" Button. ReplayEngine will be enabled after "Restart"

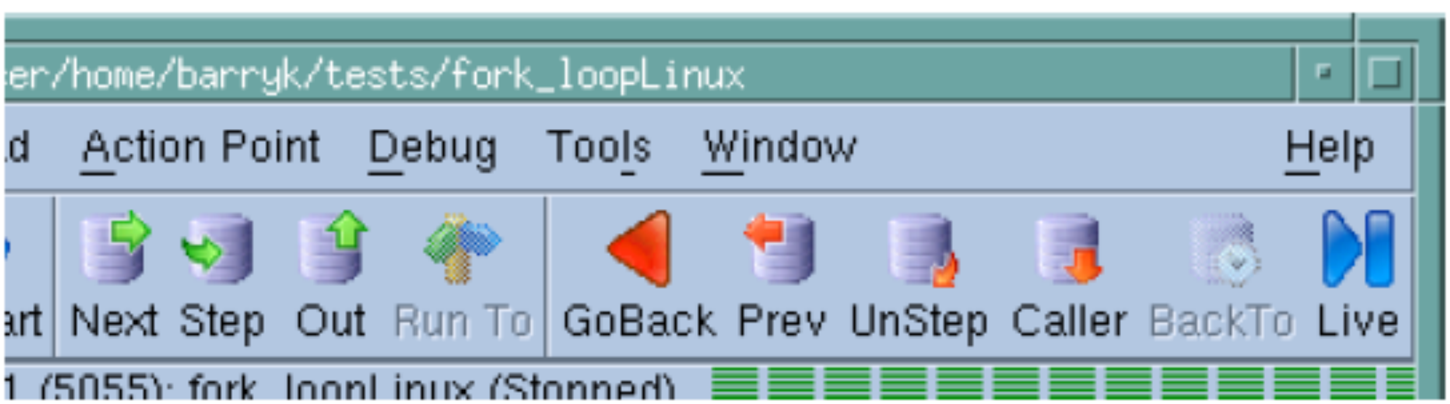
ReplayEngine Usage Models

- **GoBack** (*vs.* Go): back to the last action point or the start of the recorded history
- **Prev** (*vs.* Next): back to previous statement executed. If the line has a function call, it skips over the call
- **Unstep** (*vs.* Step): back to previous statement executed. If the line has a function call, it moves to the last statement in the call
- **Caller** (*vs.* Out): back to before the current routine was called.

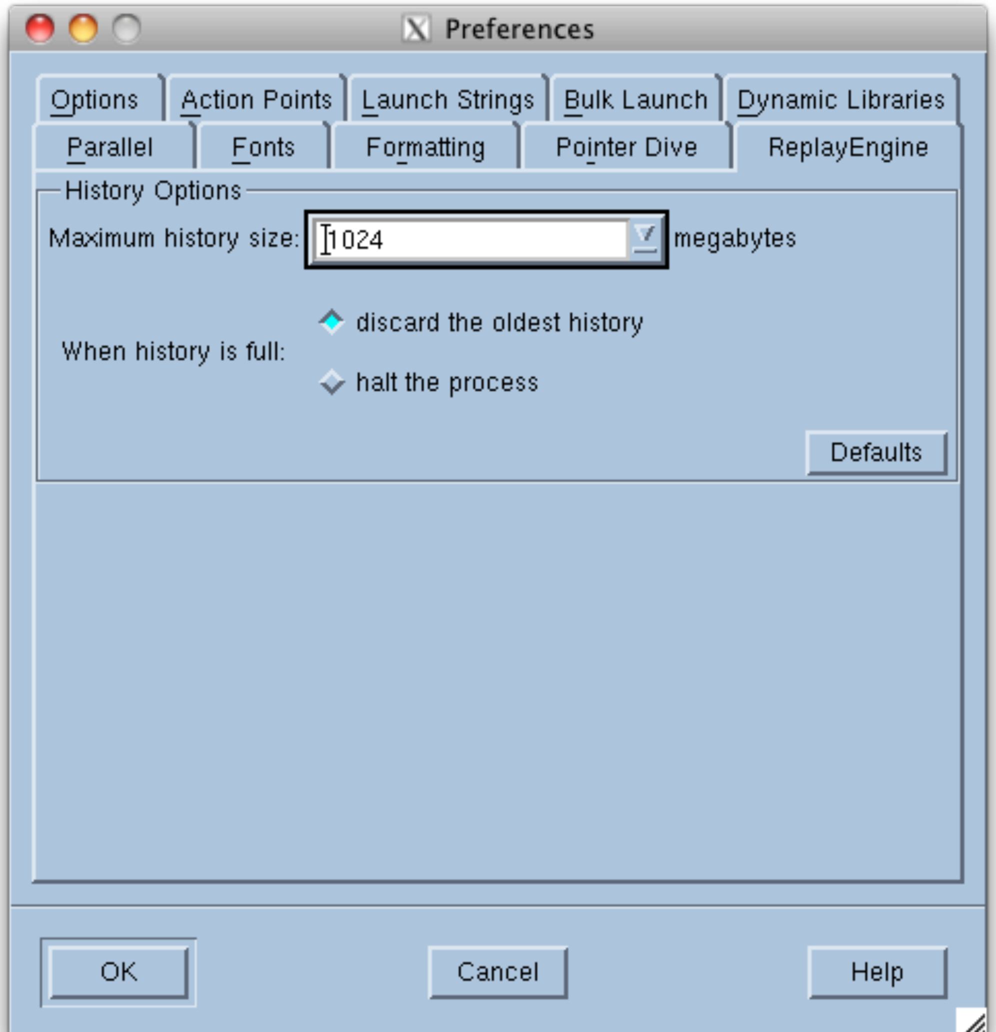


ReplayEngine Usage Models (Cont'd)

- **BackTo** (*vs*. Run To): back to the line you select
- **Live**: to shift from Replay mode to Record mode. It displays the statement that would have executed if you had not moved into Replay mode.



ReplayEngine Preferences



- Setting Maximum History size (default is unlimited)
- setenv TVD_REPLAY_TMPDIR to control the directory of saving the history information (default is /tmp)

Demo 1: ReplayEngine

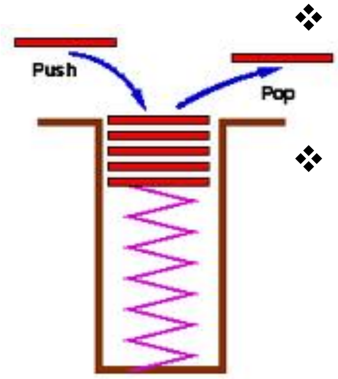
- Debugging an OpenMP Spaceweather application with ReplayEngine enabled
- Toolbar commands
- Preferences
- TVD_REPLAY_TMPDIR

Agenda For Part 2

- Overview
- Reverse Debugging with ReplayEngine
- Demo 1: Replay Engine
  - Using an OpenMP Space weather application
- Memory Debugging with MemoryScape
- Demo 2: MemoryScape
  - Using GEOS AGCM

Your Program data

- Your program data is organized in four segments:
  1. Text (code): machine code instructions
  2. Data section: static and global variables
  3. Stack: local (automatic) variables
     - Memory set aside for each thread of execution. It remains during program execution.
     - When a function is called, a block is reserved on the top of the stack for local (automatic) variables. It will go out of scope when the function returns and automatically deallocate.
     - Accessing stack in LIFO order – the most recently reserved block is the next block to be freed.
     - Easy to keep track of the stack -- Stack typically maps to the cache, so it is faster than the heap but smaller and expensive

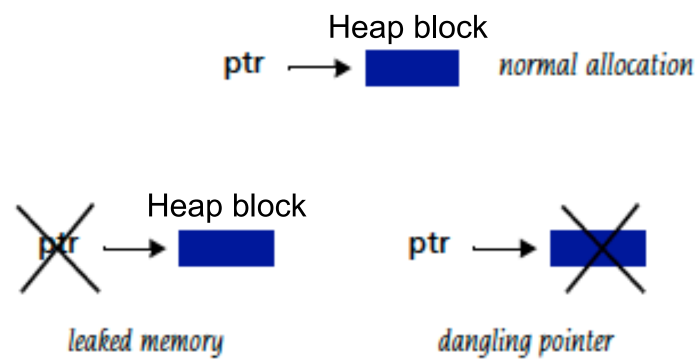


Your Program data -- What is heap?

4. Heap: all variables created or initialized *at* runtime are stored
   - Dynamic memory allocations. Heap size can grow as space is needed.
   - Variables created on the heap must be destroyed manually and never fall out of scope.
   - There's no enforced pattern to the allocation and deallocation of blocks from the heap; you can allocate a block at any time and free it at any time
   - Slower to allocate on the heap in comparison to variables on the stack
   - Can have fragmentation when there are a lot of allocations and deallocations
   - Responsible for memory leaks and many other memory bugs

What is MemoryScape?

- Runtime Memory Analysis to detect memory bugs
  - A memory bug is a mistake in heap memory usage
    - Failure to check for error conditions
    - Leaking: failure to free memory
    - Dangling references: failure to clear pointers
    - Memory corruption
      - Writing to memory not allocated
      - Over running array bounds
- Designed to be used with parallel applications.



Why are memory bugs hard to find?

- Memory problems can be hidden
  - For a given scale or platform or problem, they may not be fatal
  - Failures could occur until modification, reuse of a component, or moving the application to a different cluster with a new OS
  - Libraries could be a source of problem
  - Can also manifest as “random crashes” or “random wrong answers”

MemoryScape Features:

- Stopping execution when heap problems occur
- Viewing the heap graphically
- Leak detection and dangling pointer detection
- Memory Comparisons between processes
- Memory Corruption detection – Bound errors
  - Guard Blocks - allocated additional memory (8 bytes default) before or after a heap block. An event notification occurs when overwriting guard blocks.
  - Red Zone – allocated additional buffer before or after a heap block. An event notification occurs when either writing or reading to the Red Zone. (High memory overhead!)

Memory Debugger Features: (Cont'd)

- Block painting
  - Writing a bit pattern into allocated and deallocated blocks
  - Detecting the use of memory either before it is initialized or after it is deallocated
- Hoarding
  - Holding onto deallocated memory so it cannot be reused immediately
  - Not an often-used feature
  - Mostly used to detect problems related to memory being deallocated by one thread while another thread is using this memory

Start MemoryScape

1. Make sure your program is compiled with –g
2. module load tool/tview-8.9.2-2
3. setenv PATH $PATH:/usr/local/toolworks/memoryscape.3.2.2-2/bin/
4. Launch totalview as usual, i.e.,

totalview <executable>

Then click “Enable memory debugging” in the “Startup Parameters” window

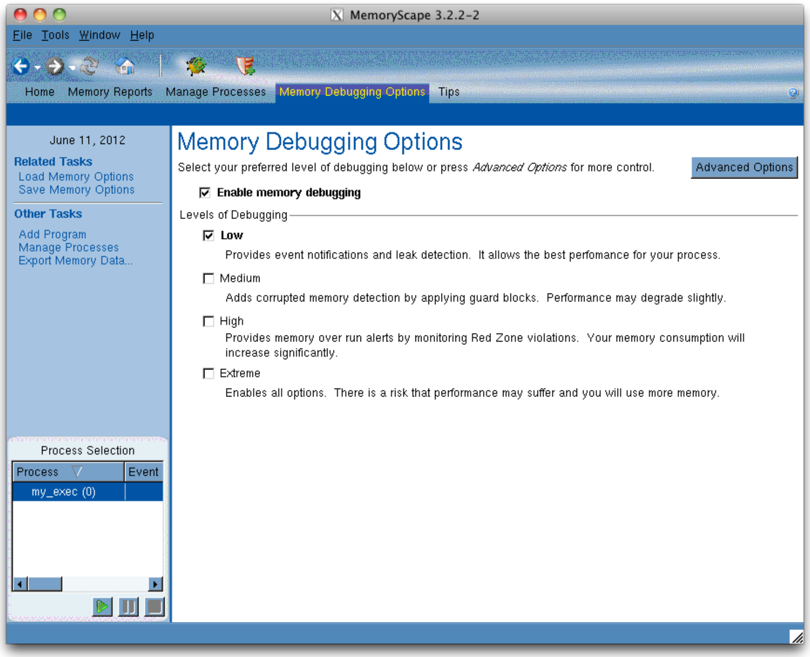
Or just type:

memscape <executable>

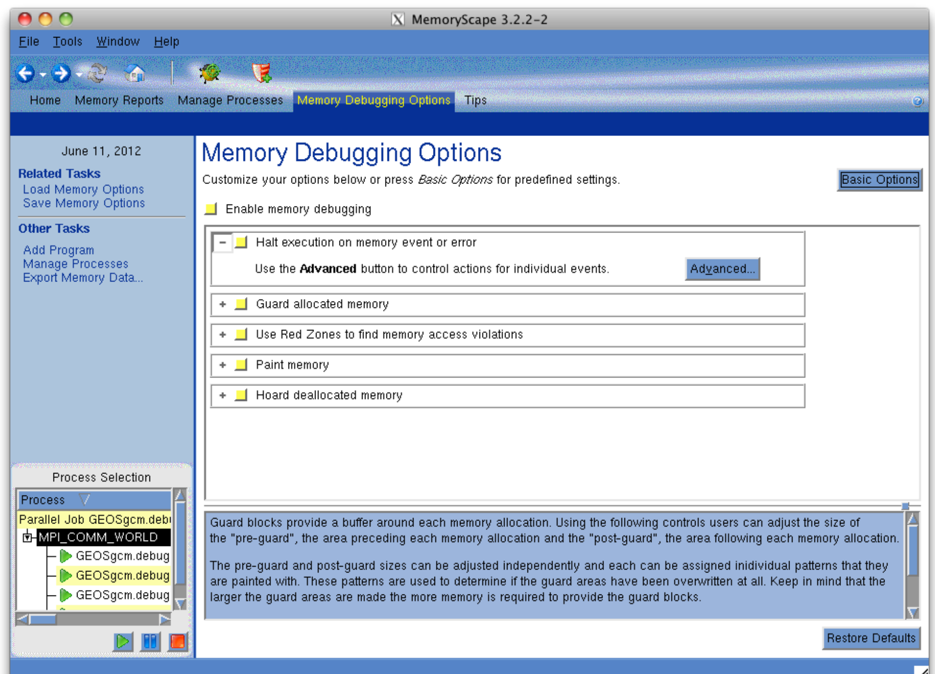
Strategies for Memory Debugging for Parallel Applications

- Run (or step through) the application and see if memory events are detected
- View memory usage across the MPI job
  - Compare memory footprints of the processes
- Gather heap information/Leak reports in all processes of the MPI job
  - Select and examine individually
    - Look at the allocation pattern
    - Look for leaks
  - Compare with the "diff" mechanism
    - Look for major differences

Memory Debugging Options



Low is the default;
Medium adds guard blocks;
High adds Red Zone detection;
Extreme adds above all plus paint memory and hoarding.



Select **Medium** only when you need to check for corrupted memory!
Select **High** only when you need to check for memory overruns!

What are memory events?

Memory Event Notification

Select events to trigger:

All | None

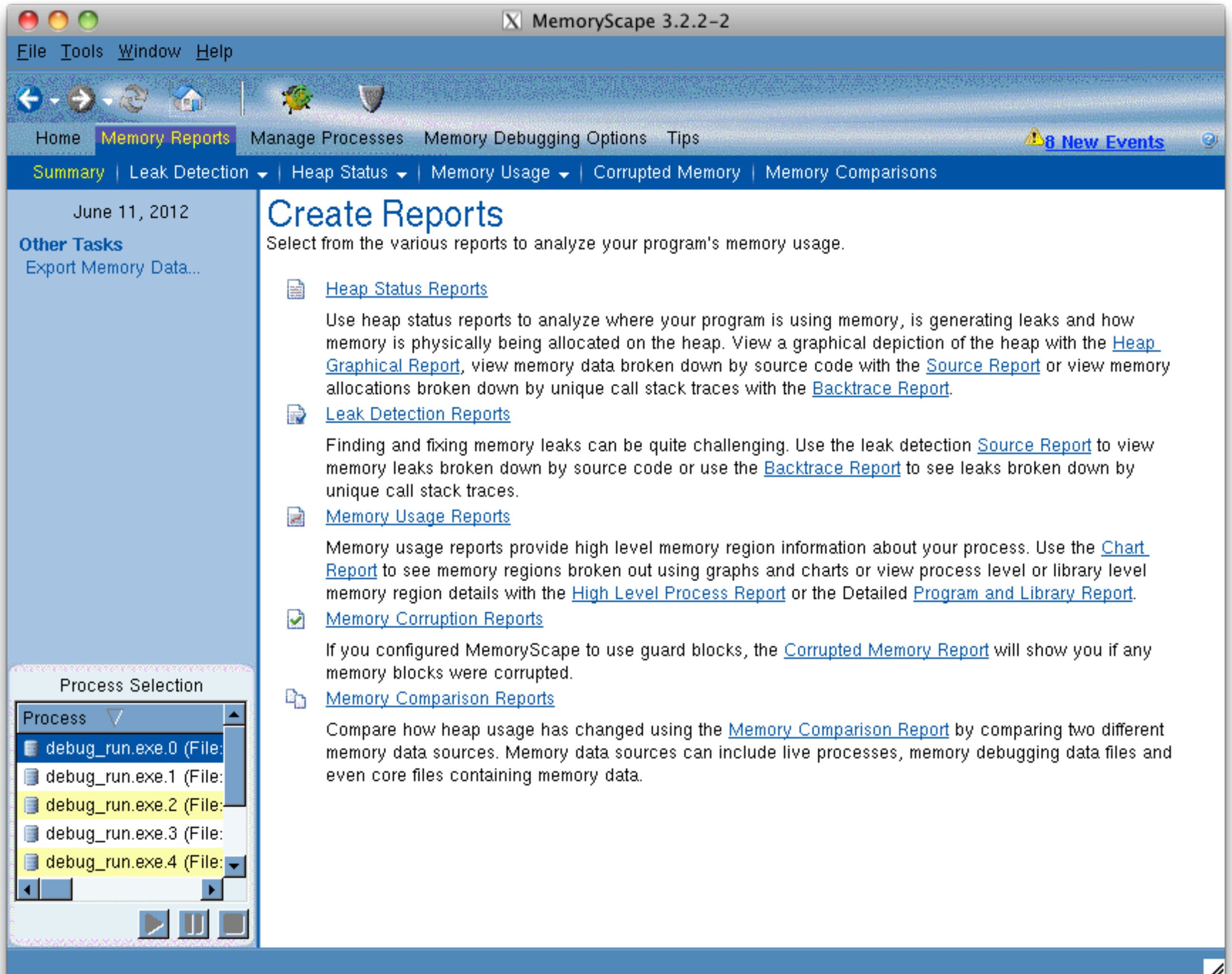
Event	Description
☑ API usage error	Incorrect API or API instance used in operation
☑ Allocation failed	Error: An allocation call failed or the address returned is NULL which generally means out of memory
☑ Double allocation	Error: Allocator returned a block already in use: heap may be corrupted
☑ Double free	Error: Program attempted to free an already freed block
☑ Free interior pointer	Error: Program attempted to free a block incorrectly, via an address in the middle of the block
☑ Free notification	A block for which notification was requested is being freed
☑ Free unknown block	Error: Program attempted to free an address not in the heap
☑ Guard corruption error	Bounds error: The guard area around a block has been overwritten
☑ Invalid aligned allocation request	Error: Program supplied an invalid alignment argument to the heap manager
☑ Misaligned allocation	Error: Allocator returned a misaligned block: heap may be corrupted
☑ Realloc notification	A block for which notification was requested is being reallocated
☑ Realloc unknown block	Error: Program attempted to reallocate an address not in the heap
☑ Red Zone overrun error	Bounds error: Attempting to access memory beyond the end of an allocated block
☑ Red Zone overrun on deallocated block	Bounds error: Attempting to access memory beyond the end of a deallocated block
☑ Red Zone underrun error	Bounds error: Attempting to access memory before the start of an allocated block
☑ Red Zone underrun on dellocated block	Bounds error: Attempting to access memory before the start of a deallocated block
☑ Red Zone use-after-free error	Access error: Attempting to access a block after it has been deallocated
☑ Termination notification	The target is terminating, memory analysis can be performed

Help | OK | Cancel

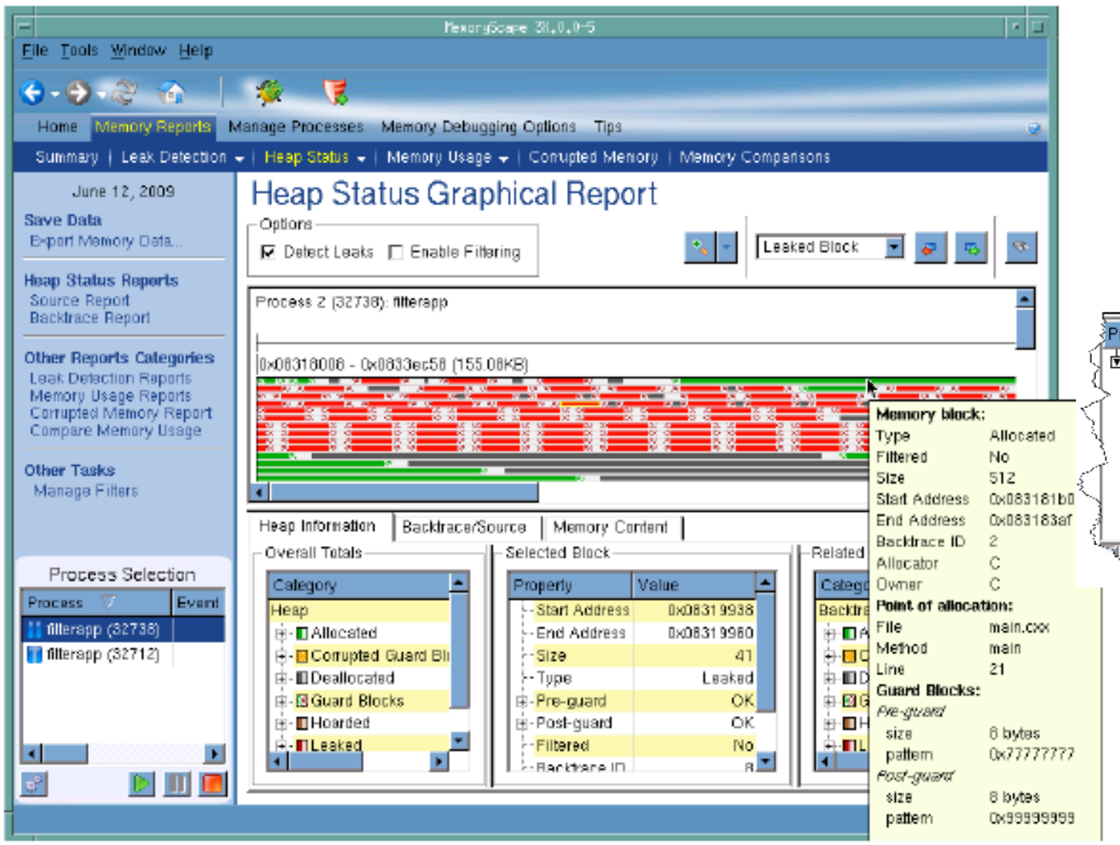
When one of these errors occurs, MemoryScape places event indicators by the process and at the top of the window

Analysis of memory debugging data

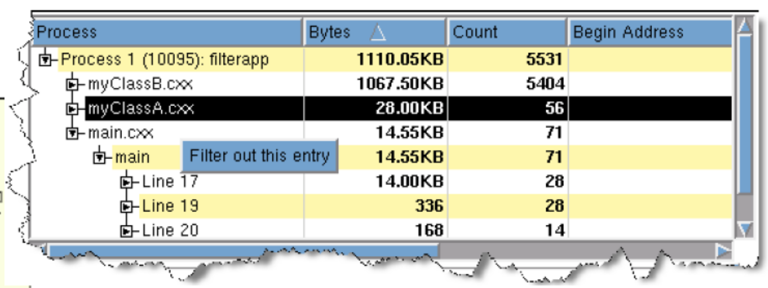
- Heap status
  - Graphical report
  - Source report
  - Backtrace report
- Leak detection
  - Source report
  - Backtrace report
- Memory usage
  - Chart report
  - High-level table
  - Detailed table
- Memory corruption reports
  - If guard blocks are chosen
- Memory comparison among processes



Heap Status Graphic Report

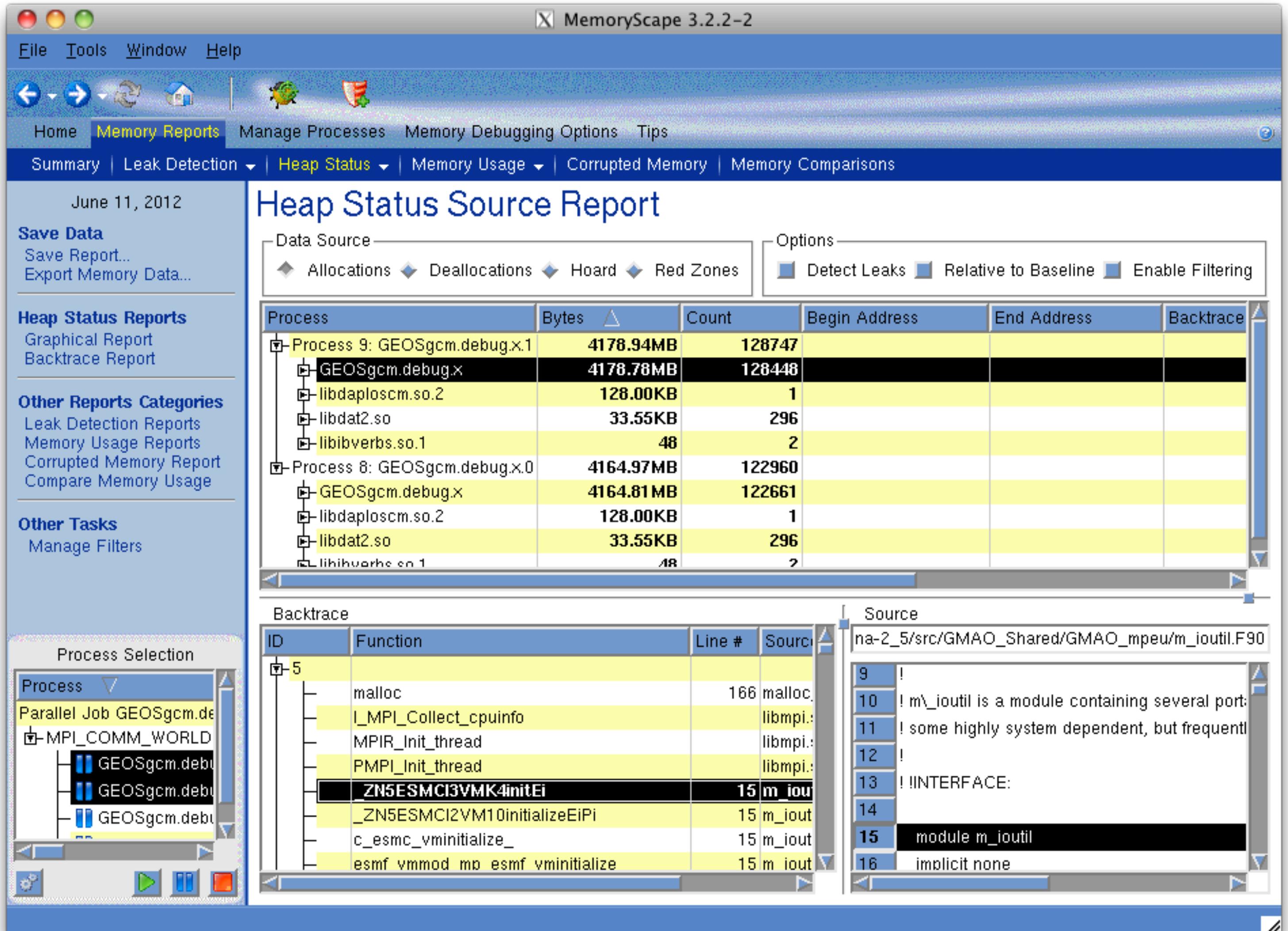


- Filtering reports is usually necessary to suppresses the display of too much information.

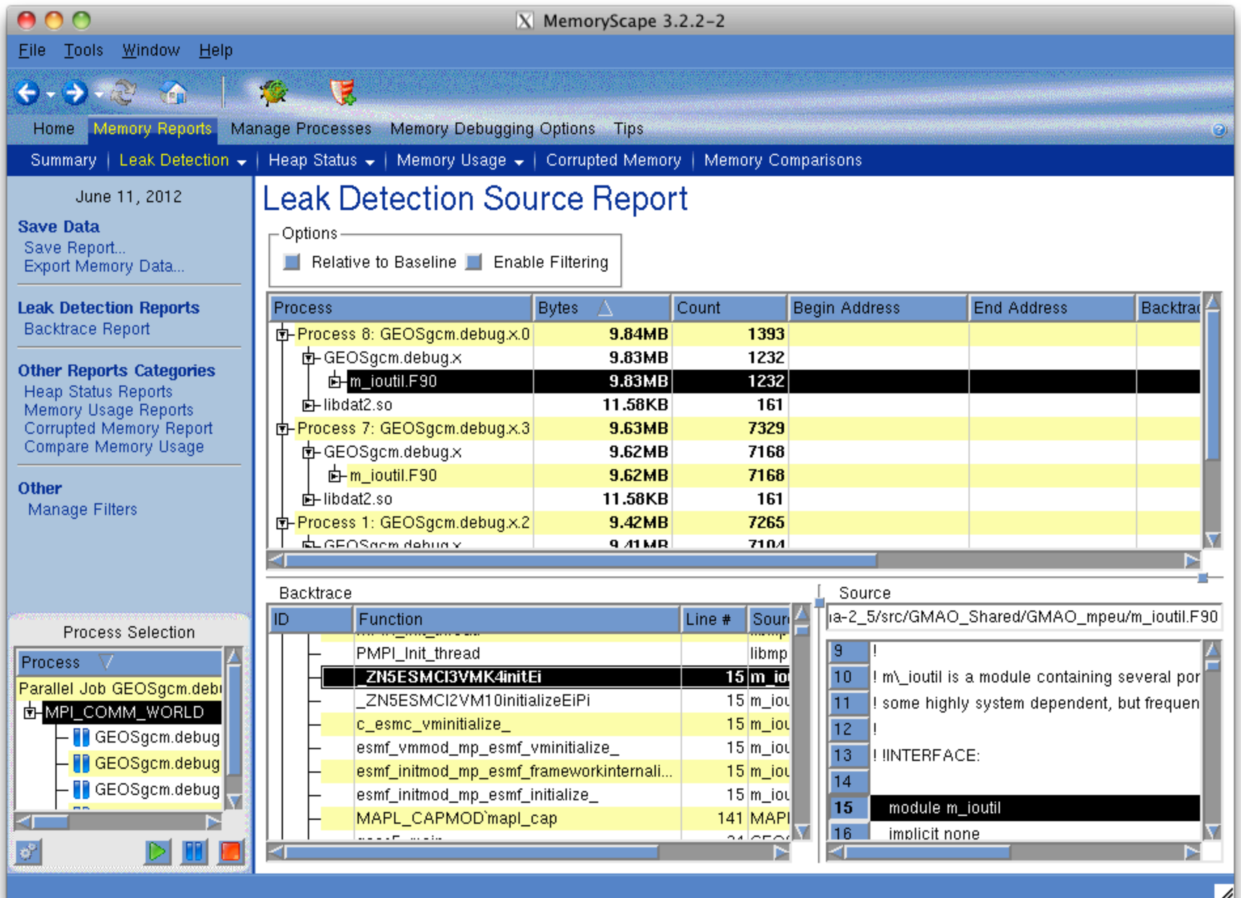


- Right-click routine name or line number in Heap Status Source or Backtrace report or in a Heap Status Graphical report, and choose “Filter out this entry”

Heap Status Source Report

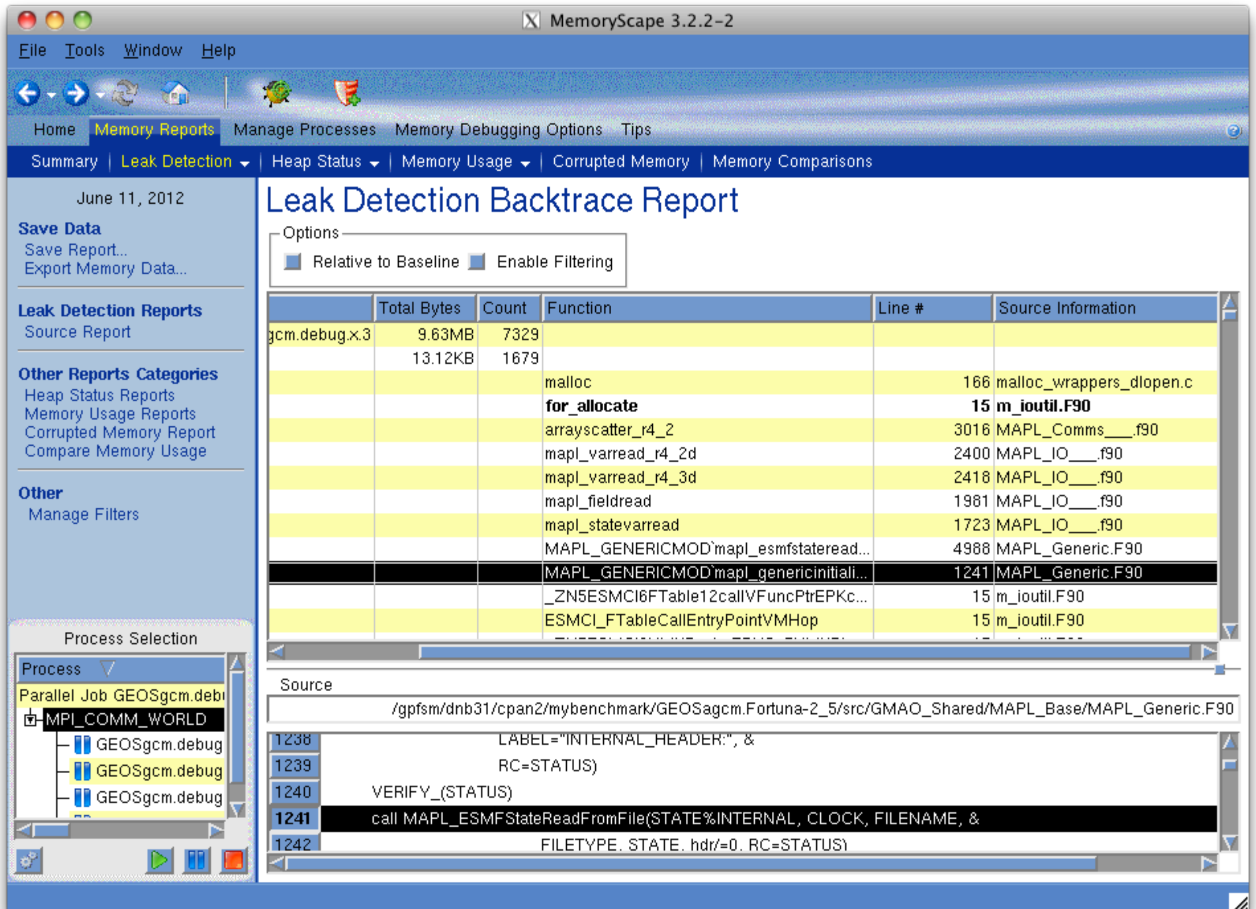


Leak Detection Source Reports



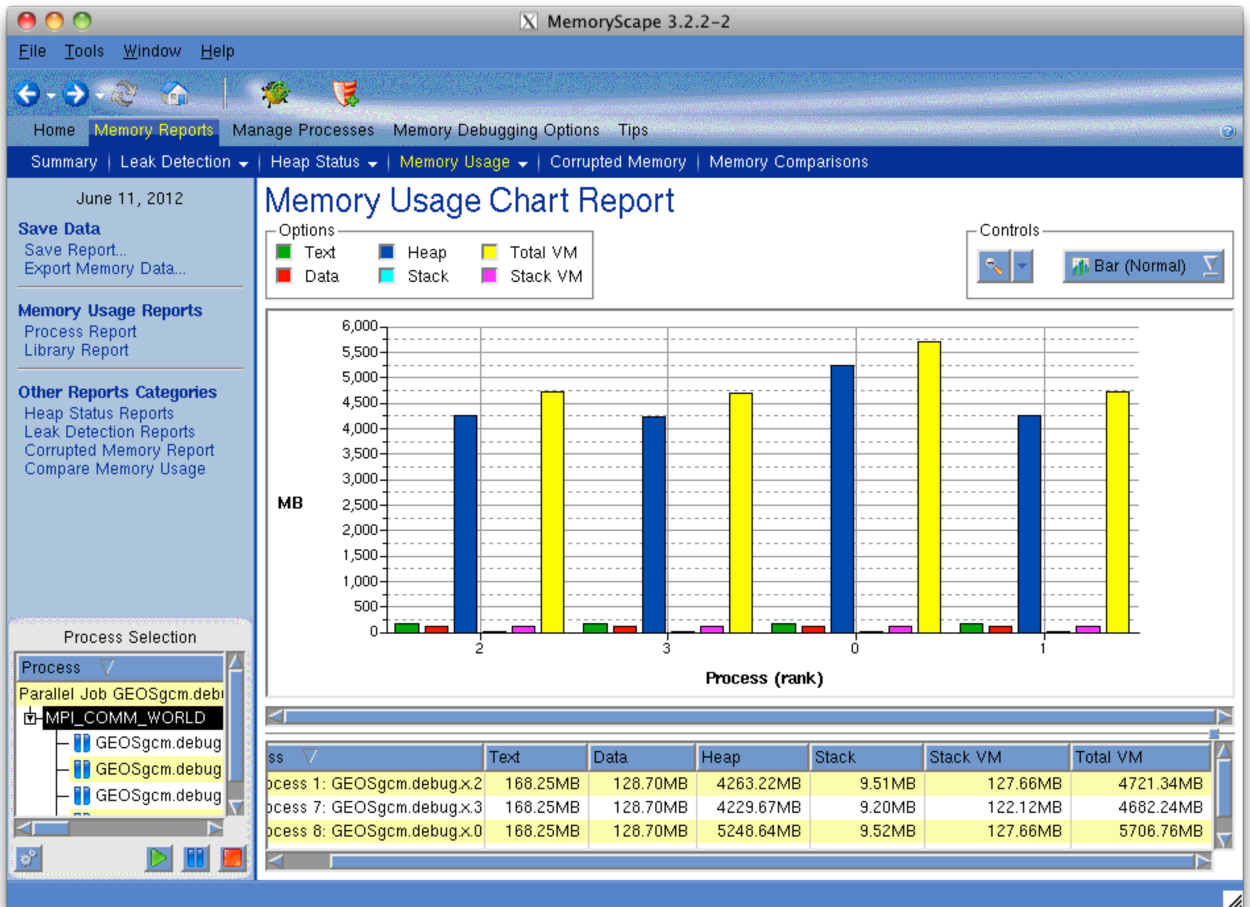
Leak reports are essentially the same as the Heap status reports, only with fewer controls and less data.

Leak Detection Backtrace Reports



Backtrace Report contains similar info as the source report. It is useful in helping you coordinate info in different screens and tabs as it does not change from report to report.

Memory Usage Chart Report



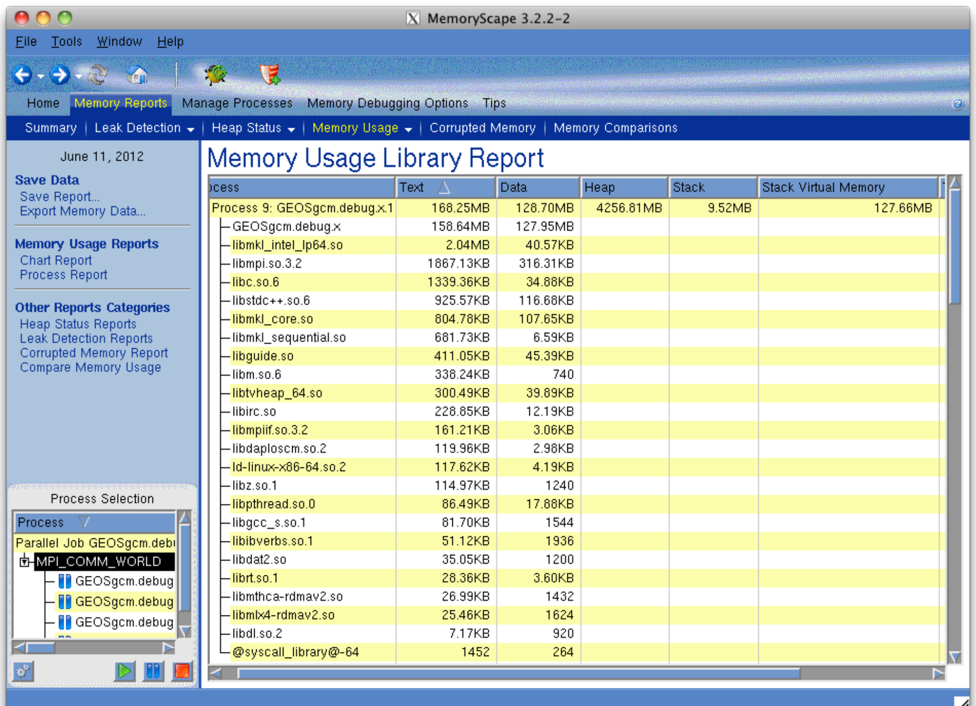
Differ slightly from the Heap Status reports:

- Memory usage data is obtained from the OS perspective, including overhead of the MemoryScape itself;

- Heap status is obtained from monitoring program requests and releases of memory

Memory Usage detailed report

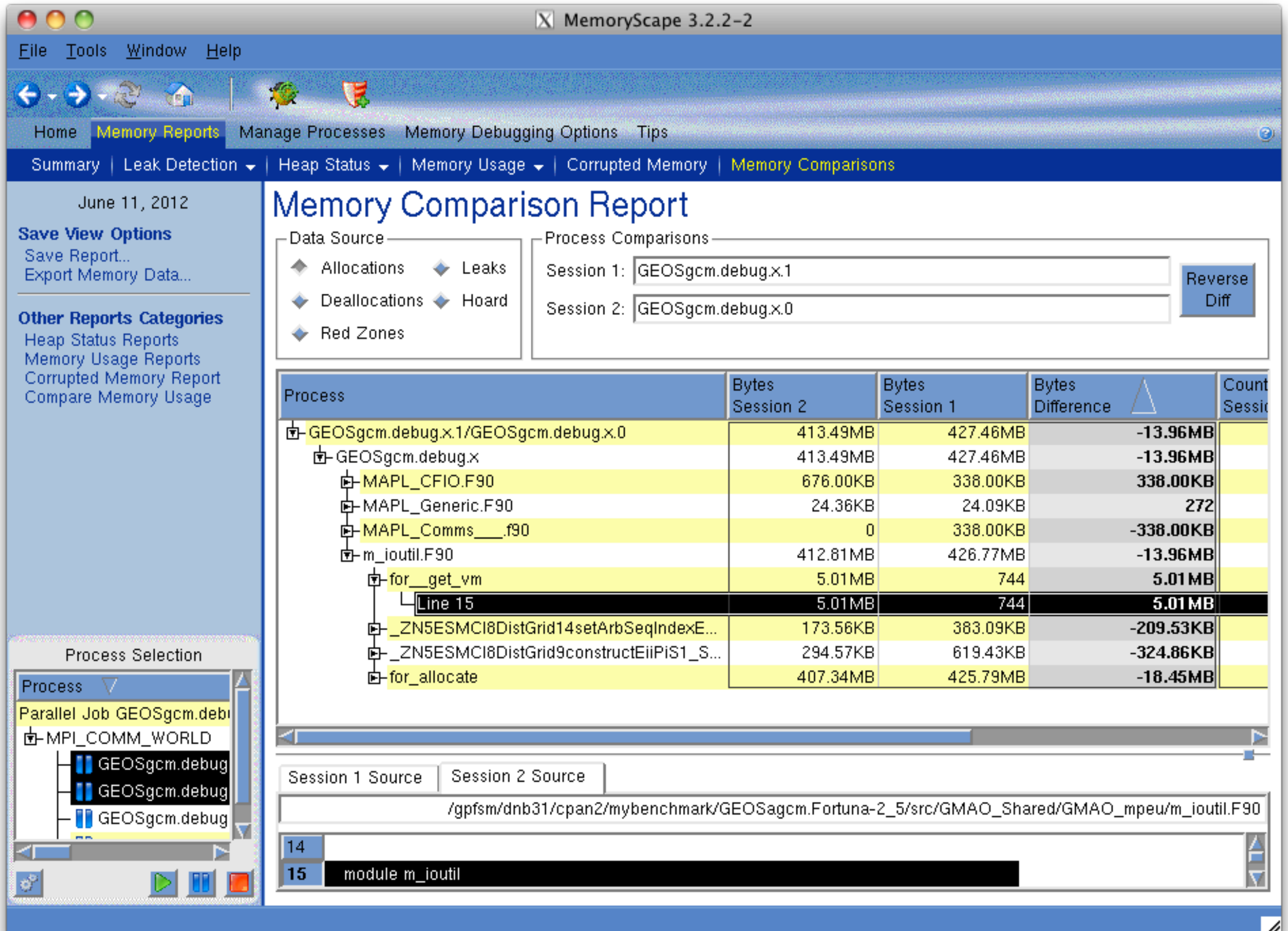
A detailed program/library report table – breakdown for the program and libraries for each process:



ocess	Text	Data	Heap	Stack	Stack Virtual Memory
Process 9: GEOSgcm.debug.x.1	168.25MB	128.70MB	4256.81MB	9.52MB	127.66MB
GEOSgcm.debug.x	158.64MB	127.95MB			
libmkl_intel_lp64.so	2.04MB	40.57KB			
libmpi.so.3.2	1867.13KB	316.31KB			
libc.so.6	1339.36KB	34.88KB			
libstdc++.so.6	925.57KB	116.68KB			
libmkl_core.so	804.78KB	107.65KB			
libmkl_sequential.so	681.73KB	6.59KB			
libguide.so	411.05KB	45.39KB			
libm.so.6	338.24KB	740			
libtvheap_64.so	300.49KB	39.89KB			
libirc.so	228.85KB	12.19KB			
libmpiif.so.3.2	161.21KB	3.06KB			
libdaploscm.so.2	119.96KB	2.98KB			
ld-linux-x86-64.so.2	117.62KB	4.19KB			
libz.so.1	114.97KB	1240			
libpthread.so.0	86.49KB	17.88KB			
libgcc_s.so.1	81.70KB	1544			
libibverbs.so.1	51.12KB	1936			
libdat2.so	35.05KB	1200			
librt.so.1	28.36KB	3.60KB			
libmthca-rdmav2.so	26.99KB	1432			
libmlx4-rdmav2.so	25.46KB	1624			
libdl.so.2	7.17KB	920			
@syscall_library@-64	1452	264			

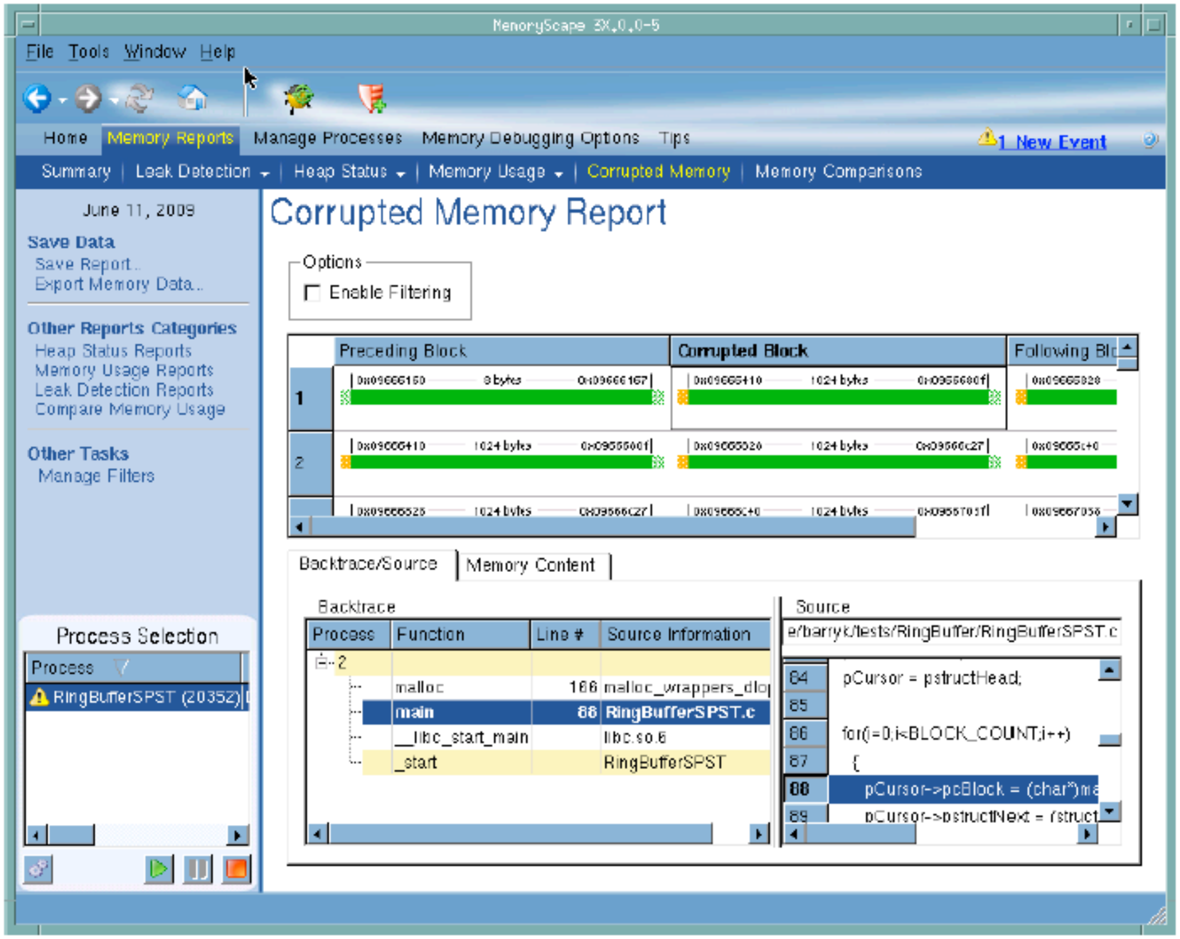
You can also see just a high level process report table – breakdown for each process.

Memory Comparison Report



Memory Corruption Report

Memory Corruption Reports are generated only if you select “Medium” or higher debugging option



- •The top section graphically displaying each corruption.
- •The bottom section contains a backtrace and the allocation source line.

Orange: corrupted guard block

Light green: uncorrupted guard block

Dark green: allocated data block

Agenda For Part 2

- Overview
- Reverse Debugging with ReplayEngine
- Demo 1: Replay Engine
  - Using an OpenMP Space weather application
- Memory Debugging with MemoryScape
- Demo 2: MemoryScape
  - Using GEOS AGCM

Demo 2

- xsub -I -V –l select=1:ncpus=12:mpiprocs=12,walltime=1:00:00 –W group_list=<your_group>

Once the compute node is available:

- cd “$GEOS_EXPDIR”
- source $GEOSBIN/g5_modules
- re-compile GCM with BOPT=g
- mpdboot –n 1 –r ssh –f $PBS_NODEFILE
- module load tool/tview-8.9.2.2
- cp and link necessary files ..(all in gcm_run.j)
- totalview ./GEOSgcm.debug.x

More info and references…

- MemoryScape_User_Guide.pdf
- ReplayEngine_Getting_Started_Guide.pdf

Under /usr/local/toolworks/totalview.8.9.2-2/doc/pdf

- "Memory Debugging Parallel Programs" tutorial offered by the Totalview in SC009

http://www.crc.nd.edu/~rich/SC09/docs/tut120/tut120.pdf