



New Developments with the TotalView Debugger: Scalable messaging and support for the Intel Xeon Phi

Chris Gottbrath, Product Manager

Oct 3rd, 2012

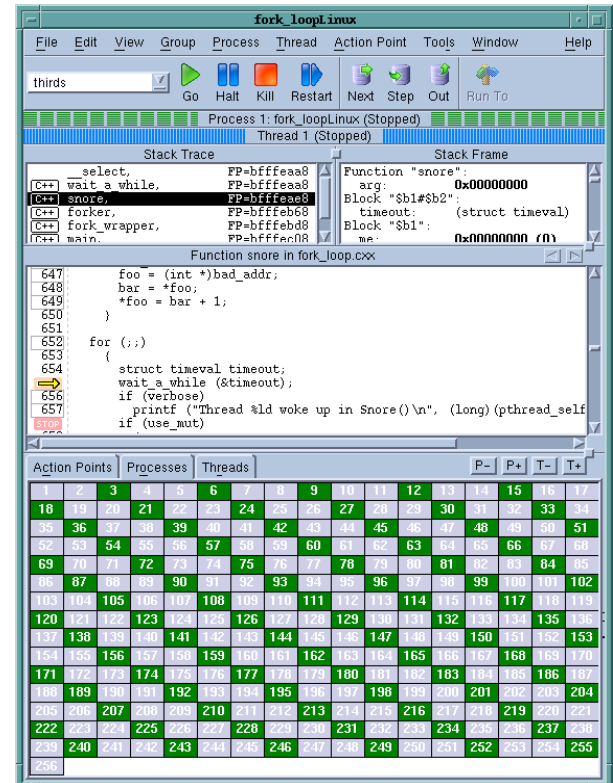
What is TotalView?

- **Application Analysis and Debugging Tool: Code Confidently**

- Debug and Analyze C/C++ and Fortran on Linux, Unix or Mac OS X
- Laptops to supercomputers (BG, Cray)
- Makes developing, maintaining and supporting critical apps easier and less risky

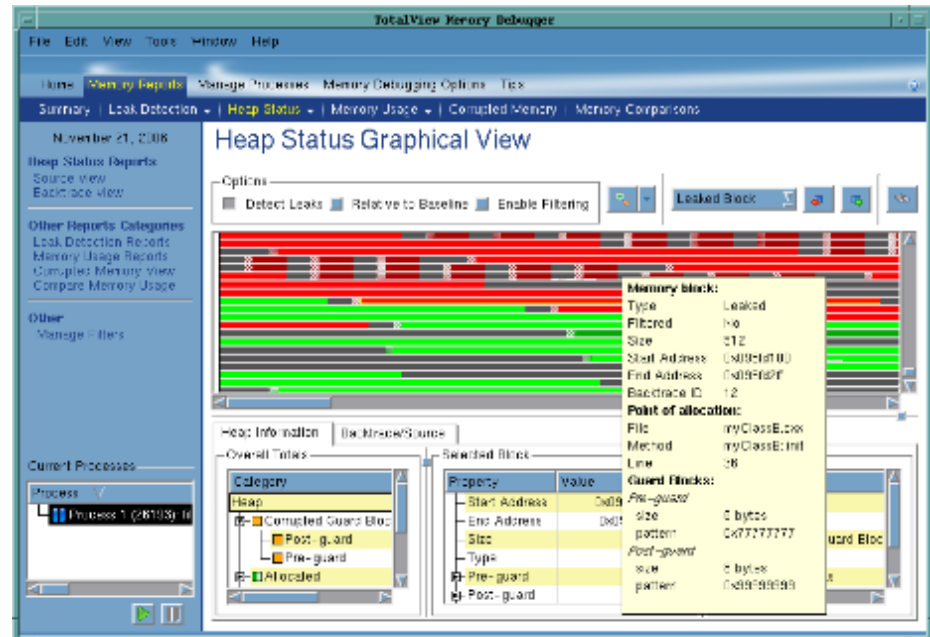
- **Major Features**

- Easy to learn graphical user interface with data visualization
- Parallel Debugging
 - MPI, Pthreads, OpenMP, GA, UPC
 - CUDA and OpenACC
- Includes a Remote Display Client freeing you to work from anywhere
- Memory Debugging with MemoryScope
- Deterministic Replay Capability included on Linux/x86-64
- Non-interactive Batch Debugging with TVScript and the CLI
- TTF & C++View to transform user defined objects



What Is MemoryScape?

- **Runtime Memory Analysis : Eliminate Memory Errors**
 - Detects memory leaks *before* they are a problem
 - Explore heap memory usage with powerful analytical tools
 - Use for validation as part of a quality software development process
- **Major Features**
 - Included in TotalView, or Standalone
 - Detects
 - Malloc API misuse
 - Memory leaks
 - Buffer overflows
 - Supports
 - C, C++, Fortran
 - Linux, Unix, and Mac OS X
 - MPI, pthreads, OMP, and remote apps
 - Low runtime overhead
 - Easy to use
 - Works with vendor libraries
 - No recompilation or instrumentation



Deterministic Replay Debugging

- Reverse Debugging: Radically simplify your debugging

- Captures and Deterministically Replays Execution
 - Not just logging or “checkpoint and restart”
- Eliminate the Restart Cycle and Hard-to-Reproduce Bugs
- Step Back and Forward by Function, Line, or Instruction

- Specifications

- A feature included in TotalView on Linux x86 and x86-64
 - No recompilation or instrumentation
 - Explore data and state in the past just like in a live process, including C++View transformations
- Replay on Demand: enable it when you want it
- Supports MPI on Ethernet, Infiniband, Cray XE Gemini
- Supports Pthreads, and OpenMP

```
40
41
42  int funcB(int
43  int c;
44  int i;
45  int v[MAXDEPT
46  int *p;
    → c=b+2;
48  p=&c;
49  ▶ if(c<MAXDEPTH
50      c=funcA(c);
51  for (i=array1
52      v[i]=*p;
```

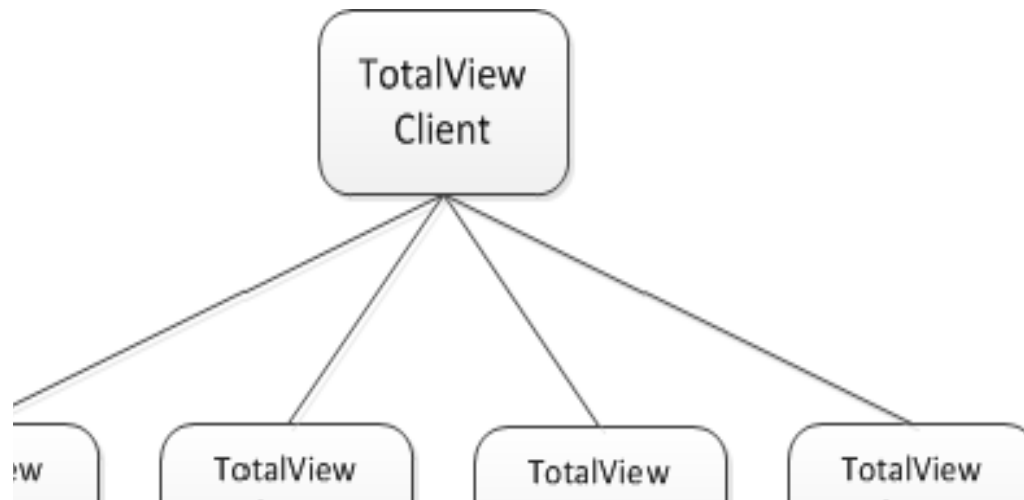


Scalability Collaboration

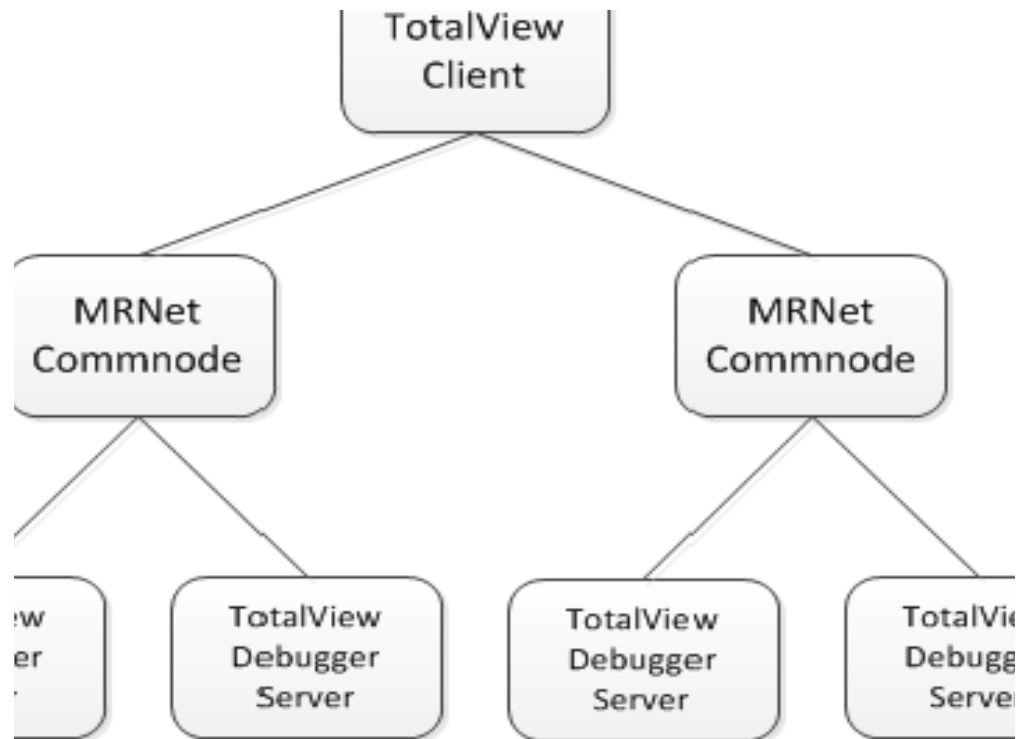
– Collaboration with LLNL

- Goal is Petascale Parallel Debugger Scalability
- MRNet - product R & D
 - Tree overlay network
- Multi-platform:
 - BlueGene/Q
 - Cray XT/XE/XK
 - Linux Cluster
- Status
 - MRNet infrastructure in place
 - Currently optimizing startup and individual operations to use MRNet
 - Significant measured improvements (5x, 20x) in performance
 - Will be available to select customers in 8.11

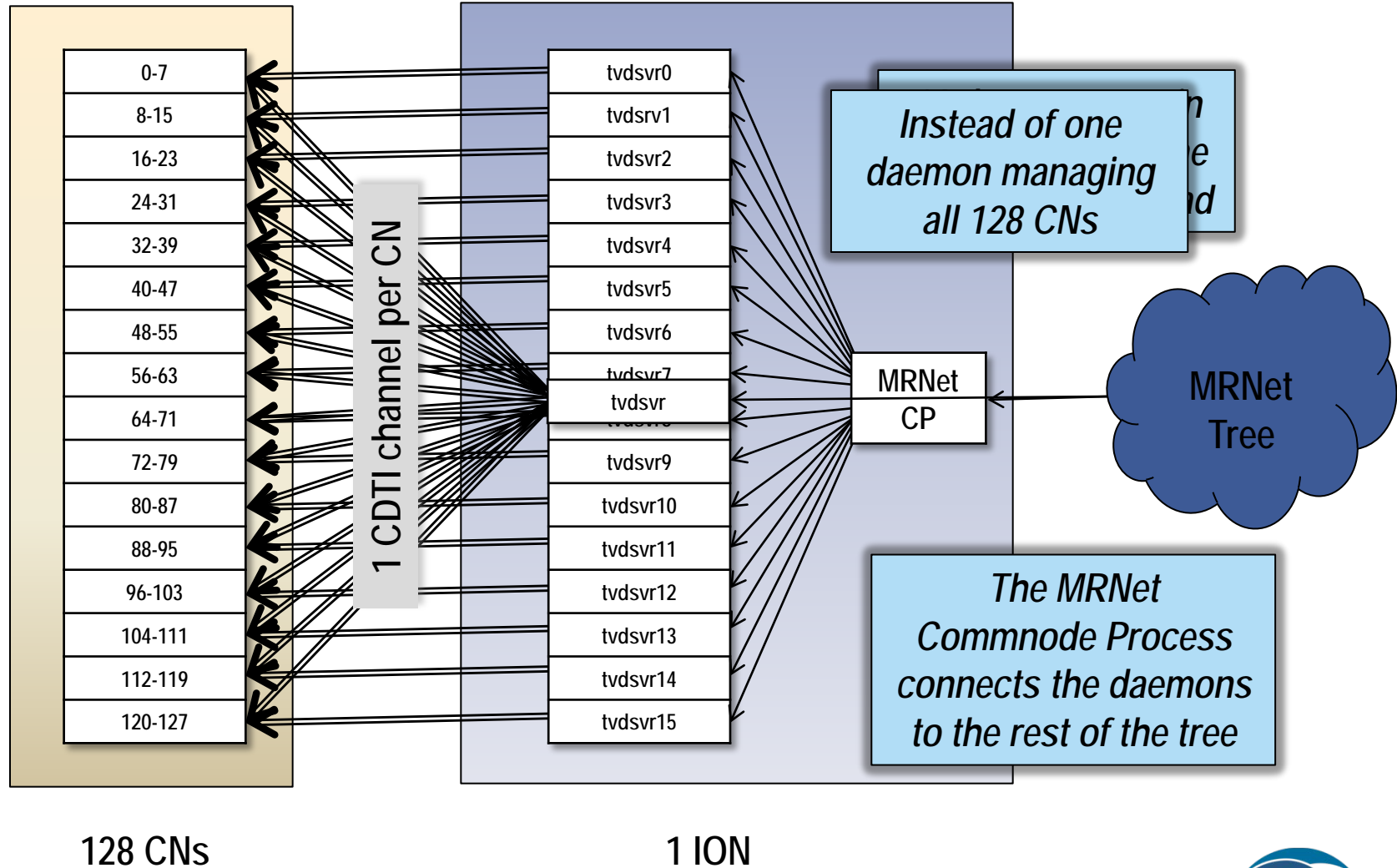
Flat Vector of Servers Infrastructure Model



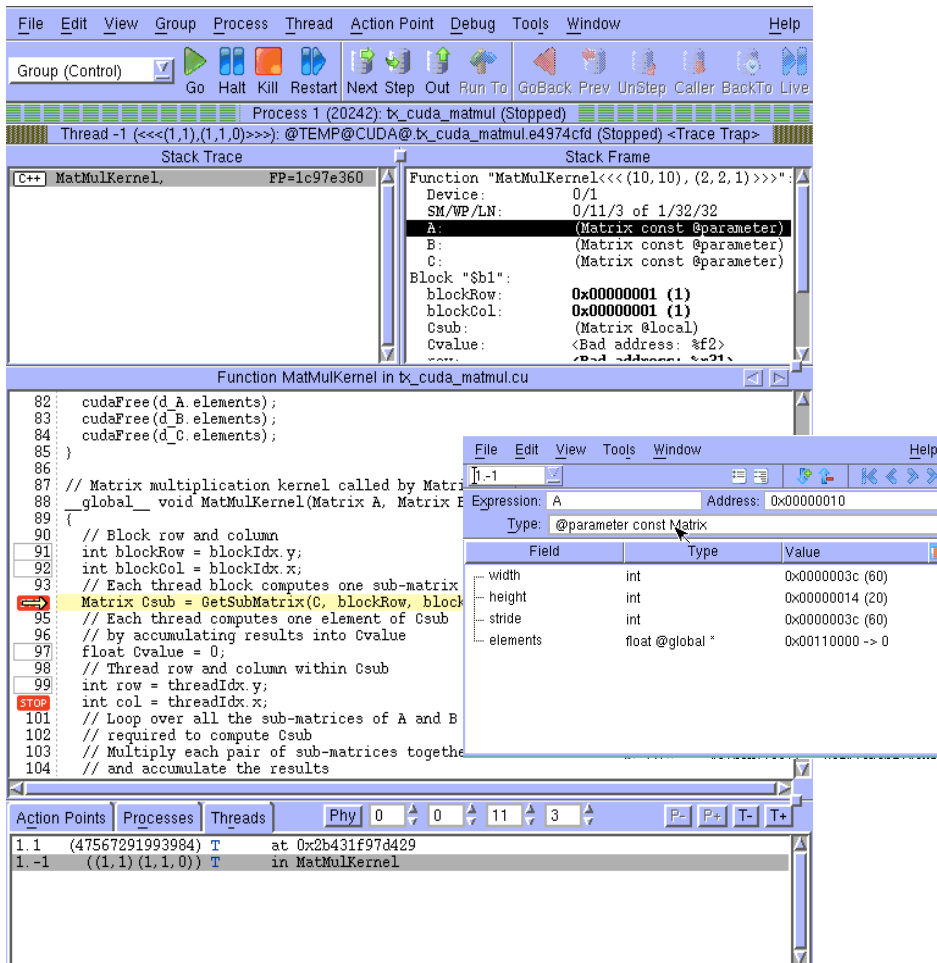
MRNET Infrastructure Model



Solution: TotalView/MRNet Trees on the IO Nodes



TotalView for CUDA



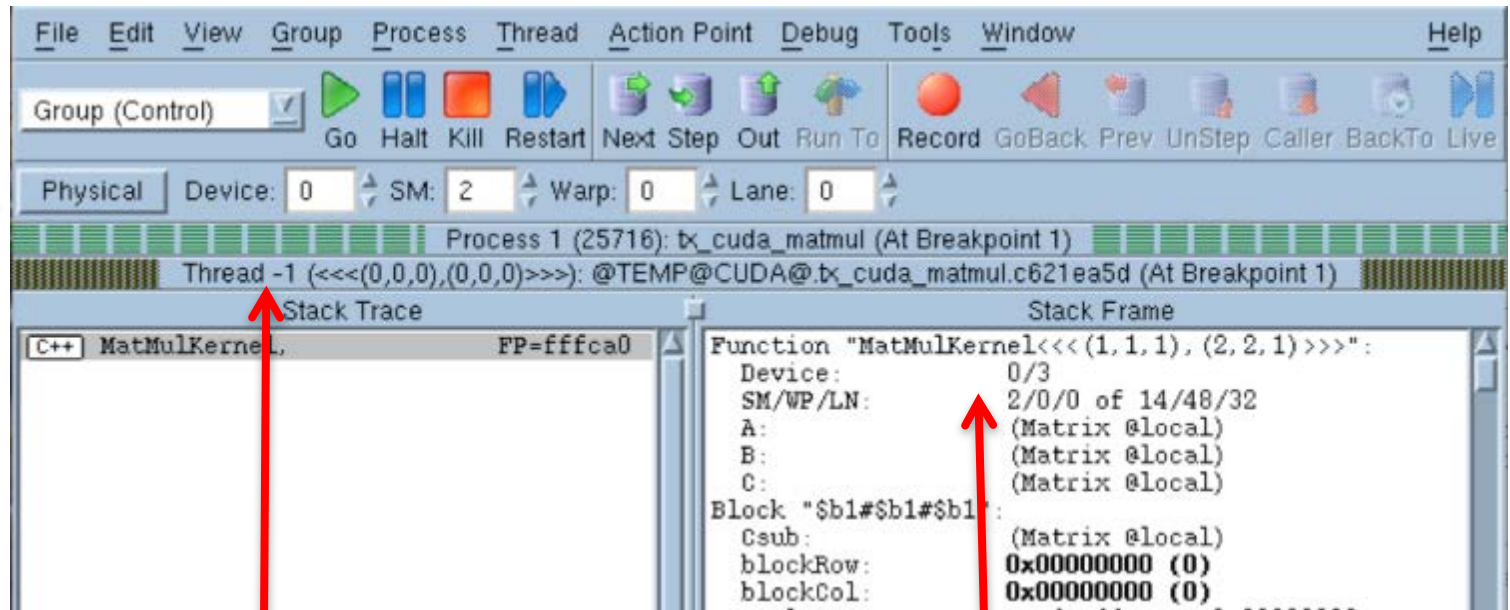
- Characteristics
 - Full visibility of both Linux threads and GPU device threads
 - Fully represent the hierarchical memory
 - Supports Unified Virtual Addressing and GPUDirect
 - Thread and Block Coordinates
 - Device thread control
 - Handles CUDA function inlining and CUDA stacks
 - Support for C++ and inline PTX
 - Reports memory access errors
 - Handles CUDA exceptions
 - Multi-Device Support
 - Can be used with MPI
- Supports CUDA 3.2, 4.0 and 4.1

Debugging CUDA with Totalview



Navigate freely through your code

Reference
by
Logical
or
Physical
attributes



Negative IDs indicate
CUDA Threads

Full Thread and Device information
in Stack Frame Pane

Support for CUDA 3.2, 4.0, and 4.1

GPU Device Status Display

- Display of PCs across SMs, Warps and Lanes
- Updates as you step
- Shows what hardware is in use
- Helps you map between logical and hardware coordinates

Name	Description
[-] Device 0/3	
[-] Device Type	gf100
[-] Lanes	32
[-] SM 2/1	
[-] Valid Warps	0000000000000001
[-] Warp 00/48	Block (0,0,0)
[-] Lane 00/32	Thread (0,0,0)
LPC	0000000019aa94d8
[-] Lane 01/32	Thread (1,0,0)
LPC	0000000019aa94d8
[-] Lane 02/32	Thread (2,0,0)
LPC	0000000019aa94f0
[-] Lane 03/32	Thread (3,0,0)
LPC	0000000019aa94f0
[-] Lane 04/32	Thread (4,0,0)
LPC	0000000019aa94f0
[-] Lane 05/32	Thread (5,0,0)
LPC	0000000019aa94f0
[-] Lane 06/32	Thread (6,0,0)
LPC	0000000019aa94f0
[-] Lane 07/32	Thread (7,0,0)
LPC	0000000019aa94f0
[-] Lane 08/32	Thread (8,0,0)
LPC	0000000019aa94f0
[-] Lane 09/32	Thread (9,0,0)
LPC	0000000019aa94f0
[-] Valid/Active/Divergent	000003ff, 000003fc, 00000003
[-] SM Type	sm_20
[-] SMs	14
[-] Warps	48
[-] Device 1/3	
[-] Device Type	gt200
[-] Lanes	32
[-] SM Type	sm_13

Example of Divergent GPU threads

Different PC for two groups of Lanes

State of Lanes inside the warp

CUDA Variables

- Storage qualifiers appear in the data type

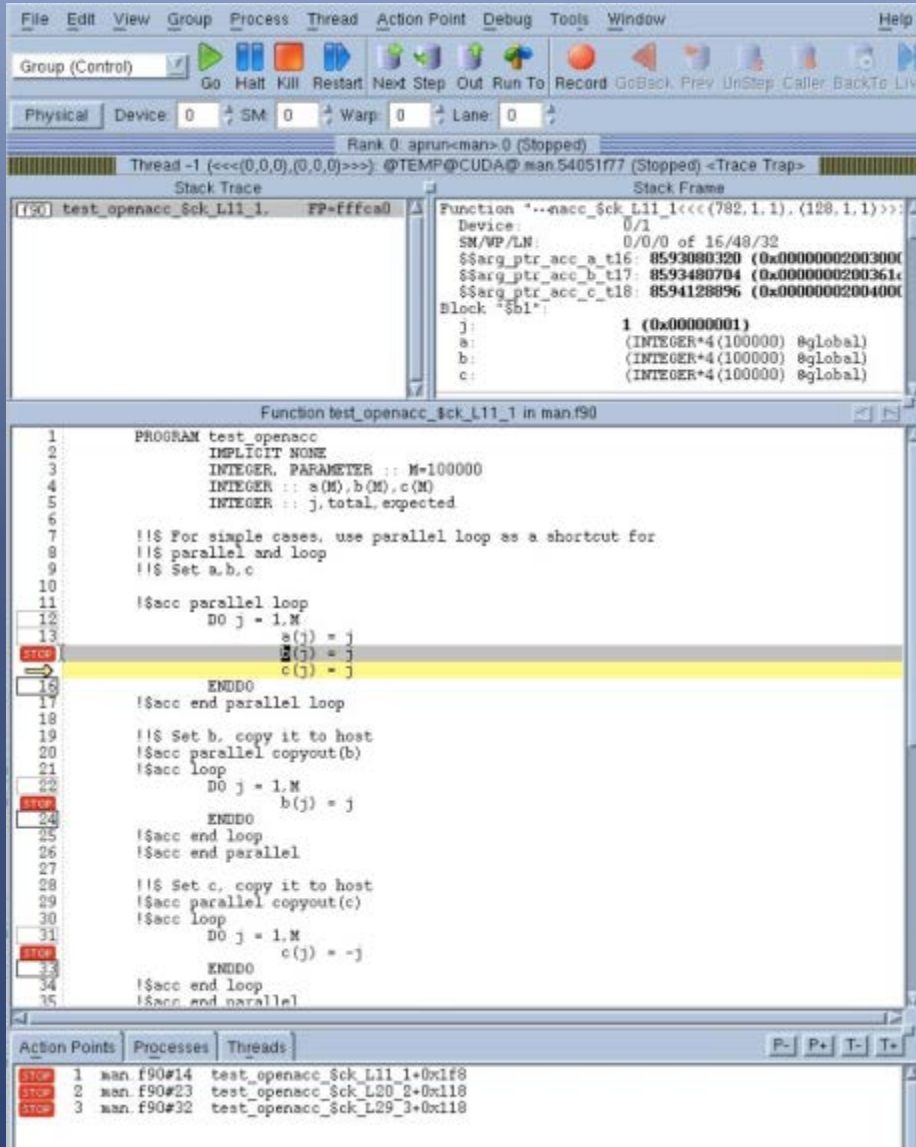
The image shows a debugger window with a variable list and a detailed view of a matrix variable. The variable list includes:

Expression	Type	Value	Address
Bs[row][col]	float	812	0x00000068
row	@register int	0x00000000 (0)	%r31
col	@register int	0x00000000 (0)	%r33
Bs[1][0]	float	832	0x00000070
Bs[row+1][1]	float	833	0x00000074
row+1	@register int	0x00000001 (1)	(None)
A	@parameter const Matrix	(Matrix const @parameter)	0x00000010
Asub	@local Matrix	(Matrix @local)	0x000000c0
*(Asub.elements)	@global		1e0

The detailed view for 'Asub' shows the following fields:

Field	Type	Value
width	int	0x00000002 (2)
height	int	0x00000002 (2)
stride	int	0x00000014 (20)
elements	float @global *	0x00111310 -> 4

TotalView for OpenACC



OpenACC®

DIRECTIVES FOR ACCELERATORS

- Step host & device
- View variables
- Set breakpoints

Compatibility with Cray
CCE 8 OpenACC

TotalView for MIC

- Support Key Intel MIC usage models
 - Native Mode
 - With MPI
 - Offload Directives
 - Similar to GPU
 - Multi-device
- User Interface
 - MPI Debugging Features
 - Process Control
 - View Across
 - Shared Breakpoints
 - Heterogeneous Debugging
 - Debug Both Xeon and Xeon-Phi Processes
- Usage
 - No special settings or set up

ID	Rank	Host	Status	Description
1		<local>	R	/opt/intel/composerxe/Sample
1.1		<local>	R	in main
1.2		<local>	R	in __poll
1.3		<local>	R	in __poll
1.4		<local>	R	in pthread_cond_wait
2		192.168.1.1(M)	R	/tmp/coi_procs/1/5856/offload
2.1		192.168.1.1(R)	R	in sem_wait
2.2		192.168.1.1(B6)	R	in compute07
2.3		192.168.1.1(R)	R	in __poll
2.4		192.168.1.1(R)	R	in pthread_cond_wait

Stack Trace

Function	Address	FP
compute07	0x7f50fd4d24f0	FP=7f50fd4d24f0
L_sample07_76__par_region1_2_39	0x7f50fd4d24c0	FP=7f50fd4d24c0
_offload_entry_sample07_c_76sample07	0x7f50fd4d24b0	FP=7f50fd4d24b0
_ZN17OffloadDescriptor7offloadEjPPvS0_tS0_t	0x7f50fd4d24a0	FP=7f50fd4d24a0
_COISinkPipe::RunFunction	0x7f50fd4d2400	FP=7f50fd4d2400
_COISinkPipe::ProcessMessages	0x7f50fd4d23e0	FP=7f50fd4d23e0
_COISinkPipe::ThreadProc	0x7f50fd4d23c0	FP=7f50fd4d23c0
start_thread	0x7f50fd4d2f30	FP=7f50fd4d2f30
_clone	0x7f50fd4d2f38	FP=7f50fd4d2f38

Registers for the frame:

%rax	0x7f50fd4d2754 (139985823803220)
%rdx	0x00000010 (16)
%rcx	0x7f50fd4d2754 (139985823803220)

```
Function compute07 in sample07.c
90 for (i=0; i<s; i++)
91 {
92     arrayd[i] = p[i];
93 }
94
95 #ifdef __MIC__
96     retval = i;
97 #else
98     retval = 0;
99 #endif
100
101 // Return 1 if array initialization was done on target
102 return retval;
103 }
104
105 __attribute__((target(mic))) void compute07(int* out, int size)
106 {
107     int i;
108
109     for (i=0; i<size; i++)
110     {
111         out[i] = arrayd[i]*2;
112     }
113 }
114 //.....07
```

Action Points | Processes | Threads

2.1	(139985896167360) R	in sem_wait
2.2	(139985823807232) B6	in compute07
2.3	(139985834444544) R	in __poll
2.4	(139985842837248) R	in pthread_cond_wait

TVScript Overview

- Gives you non-interactive access to TotalView's capabilities
- Useful for
 - Debugging in batch environments
 - Watching for intermittent faults
 - Parametric studies
 - Automated testing and validation
- TVScript is a script (not a scripting language)
 - It runs your program to completion and performs debugger actions on it as you request
 - Results are written to an output file
 - No GUI
 - No interactive command line prompt

TVScript Syntax

- tvscript syntax:
 - `tvscript [options] [filename] [-a program_args]`
- Options express (“event”, “action”) pairs
 - Typical events
 - Action_point
 - Any_memory_event
 - Guard_corruption
 - error
 - Typical actions
 - Display_backtrace [-level *level-num*] [*num_levels*] [*options*]
 - List_leaks
 - Save_memory
 - Print [-slice {*slice_exp*} {*variable* / *exp*}

```
•!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
•! Print
•!
•! Process:
•!   ./server (Debugger Process ID:  1, System ID:  12110)
•! Thread:
•!   Debugger ID:  1.1, System ID:  3083946656
•! Time Stamp:
•!   06-26-2008 14:04:09
•! Triggered from event:
•!   actionpoint
•! Results:
•!   foreign_addr = {
•!       sin_family = 0x0002 (2)
•!       sin_port = 0x1fb6 (8118)
•!       sin_addr = {
•!           s_addr = 0x6658a8c0 (1717086400)
•!       }
•!       sin_zero = ""
•!   }
•!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

- Example

```
-create_actionpoint "#85=>print foreign_addr"
```



Case Study 2: Weather and Climate Modeling

- A technique for statistical comparison of arrays using TVScript

Jacob Wiseman Poulsen

Danish Meteorological Institute

Context

- Requirement to ensure the accuracy of very complex meteorological codes with rich histories (20+ years)
- Periodic comparative study of results
 - Many compilers, multiple platforms
 - Automake with different compiler options
 - With and without various different parallelism (MPI, OpenMP, GPU, etc)
 - Exercise different decompositions
- Looking at numerical robustness and correctness of solution
 - Examine results statistically
 - Compare pointwise time series results to observations at a weather station

Debugging and Validation Framework

- Goal: easily gathering statistical and variable information
- Based on f90 comments in the following format
 - ! TVSCRIPT : dprint [level] printargument
 - ! TVSCRIPT : dprintstat [level] printargument
 - ! TVSCRIPT : statistics [level]
 - ! TVSCRIPT : cmd [level] cmdarg
- Together with a driver script that parses these arguments and turns them into TVScript event action pairs
 - The event location is a breakpoint with a position based on the comment
 - The action depends on the particular comment

Example 1

- Code Annotation

```
integer(4), parameter :: nsize=200000
do i=1,nsize,2
  asum(i)=-1.0_8
enddo
do i=1,nsize-1,2
  asum(i+1)=1000000000000_8
enddo
! TVSCRIPT: dprintstats asum
call proper_sum(nsize,asum)
x=0
do i=1,nsize
  x=x+asum(i)
enddo
```

- Output

```
jwp@munin-1:-> ./runtv.sh
Parsing test_sum.f90 for tvscript comments
TVSCRIPT: CMD LEVEL= with arg=dprint -stats asum in test_sum.f90#54
```

```
Count:      200000
Zero Count: 0
Sum:        9.9999999999991e+16
Minimum:    -1
Maximum:    1000000000000
Median:     499999999999.5
Mean:       499999999999.955
Standard Deviation: 500001250005.604
First Quartile: -1
Third Quartile: 1000000000000
Lower Adjacent: -1
Upper Adjacent: 1000000000000

NaN Count: 0
Infinity Count: 0
Denormalized Count: 0

Checksum:   3476
```

```
wp@munin-1:->
```

Example 2 and 3

- **dprint**

```
! TVSCRIPT: dprint cmod_arrays`temp(1)%p(924:927);
```

Prints a small subset of the temperature array


- **dprintstat**

```
! TVSCRIPT: dprintstat cmod_arrays`temp(1)%p(2:);
```

prints statistical information about the temperature in the part of the array indicated

TotalView Remote Display Client

File Help



Session Profiles:

- DTU
- IDRIS
- ORNL
- fez
- power6_53
- power6_61
- toro

1. Enter the Remote Host to run your debug session:
Remote Host: [jaguarpf.ccs.ornl.gov] User Name: [max100] Commands: []

2. As needed, enter hosts in access order to reach the Remote Host:

	Host	Access By	Access Value	Commands
1		User Name		
2		User Name		

3. Enter settings for the debug session on the Remote Host :
TotalView | MemoryScape |

Path to TotalView on Remote Host: [totalview]
Arguments for TotalView: [-geometry 1400x1200]
Your Executable (path & name): []
Arguments for Your Executable: []
Submit Job to Batch Queueing System: [PBS Pro]

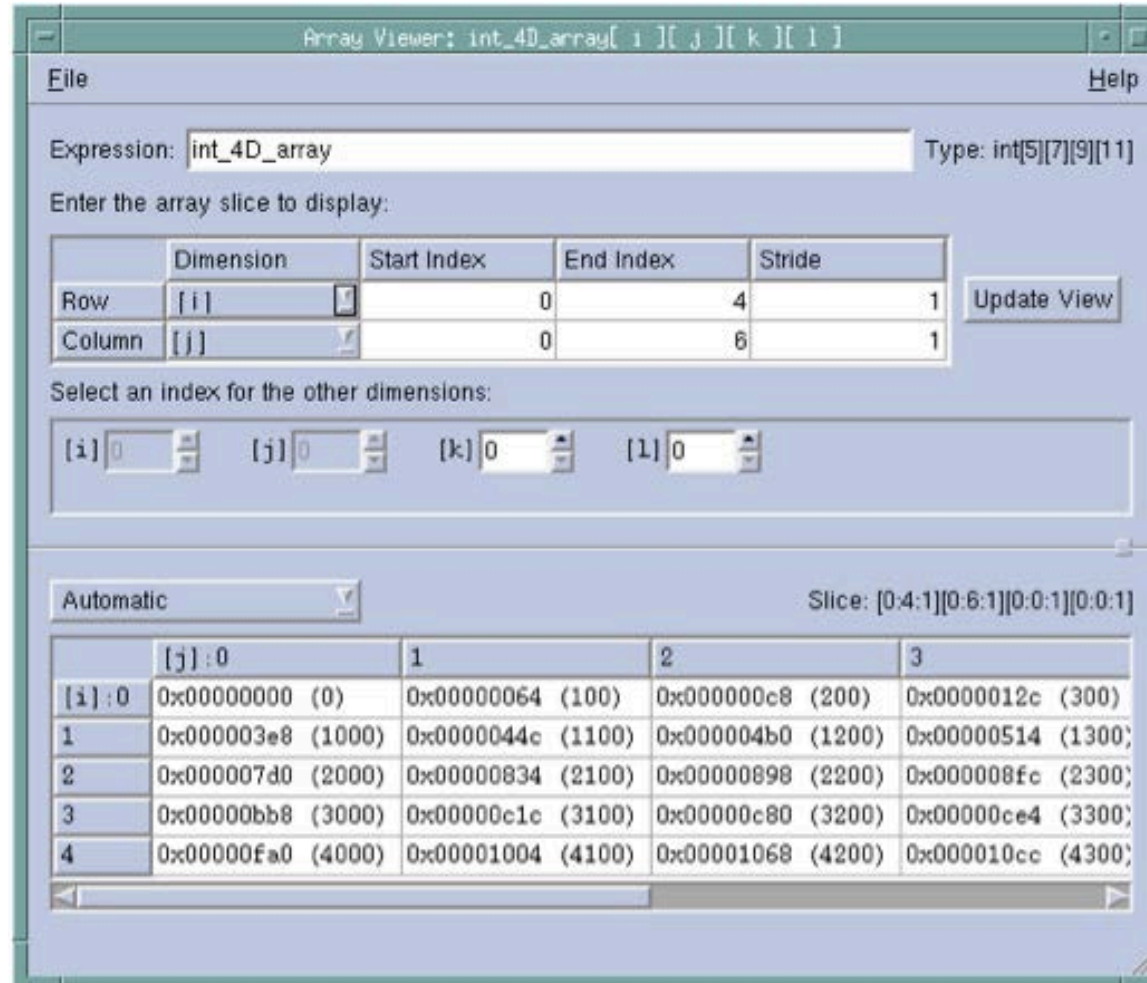
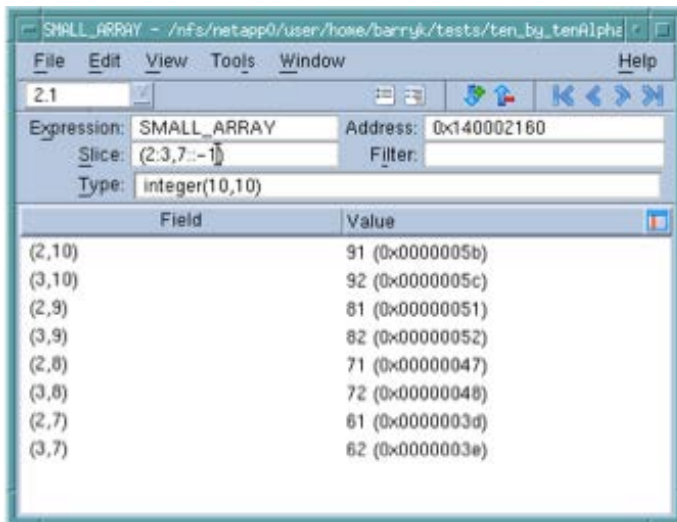
4. Enter batch submission settings for the Remote Host :
PBS Submit Command: [qsub]
TotalView PBS Script to Run: [tv_PBS.csh]
Additional PBS Options: []

Profile ORNL/TotalView debug session is running...

- The Remote Display Client offers users the ability to easily set up and operate a TotalView debug session that is running on another system.
- Provides for a connection that is
 - Easy
 - Fast
 - Secure
- The Remote Display Client is available for:
 - Linux x86
 - Linux x86-64
 - Windows XP
 - Windows Vista
 - Windows 7
 - Mac OS X Leopard and Snow Leopard
- The Client also provides for submission of jobs to batch queuing systems PBS Pro and Load Leveler

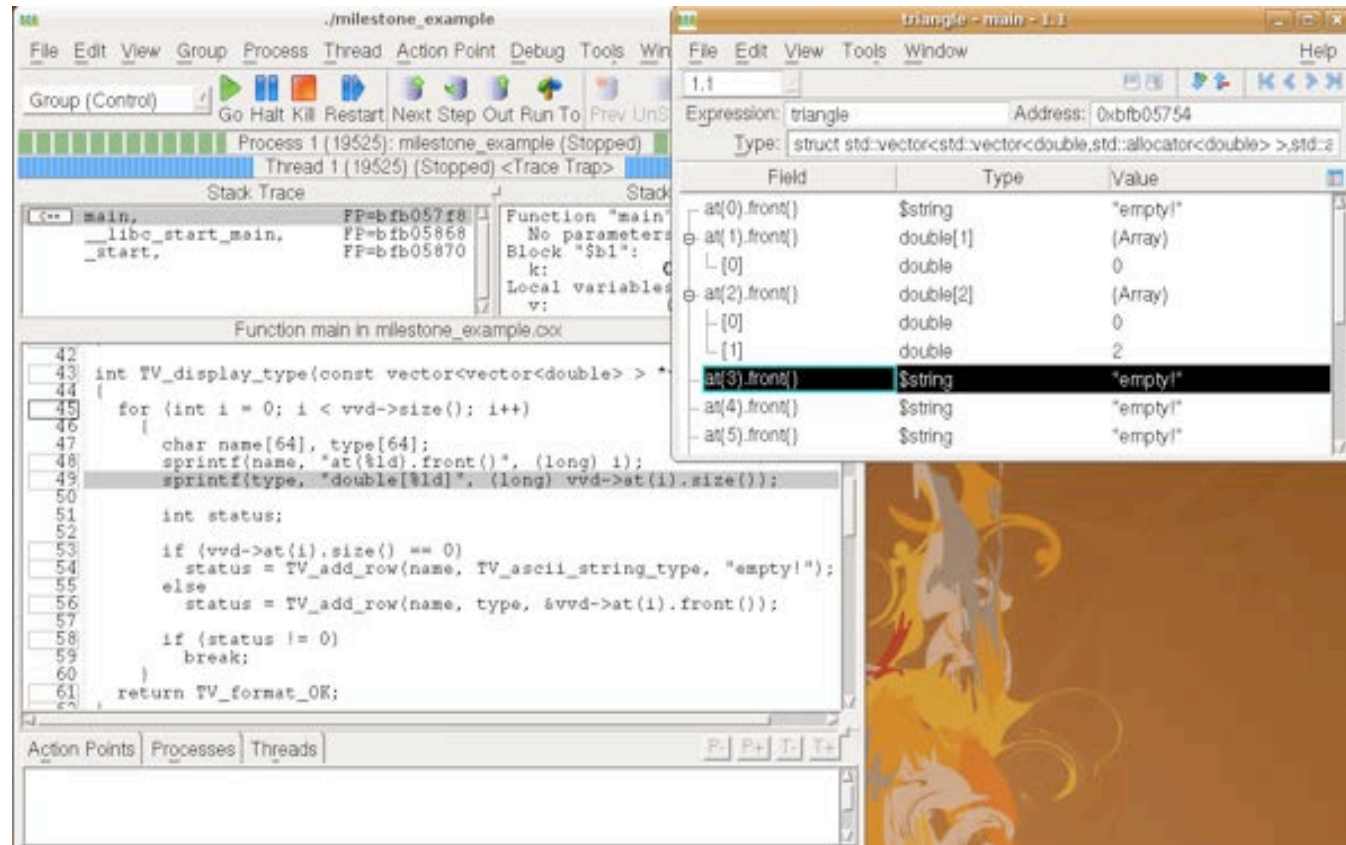
Multi-Dimensional Array Viewer

- See your arrays on a “Grid” display
- 2-D, 3-D... N-D
- Arbitrary slices
- Specify data representation
- Windowed data access
 - Fast



C++View

- C++View is a simple way for you to define type transformations
 - Simplify complex data
 - Aggregate and summarize
 - Check validity
- Transforms
 - Type-based
 - Compose-able
 - Automatically visible
- Code
 - C++
 - Easy to write
 - Resides in target
 - Only called by TotalView



What is ThreadSpotter?

Runtime Cache Performance Optimization Tool: Tune into the Multi-Core Era

- Realize More of the Performance Offered by Multi/Many-Core Chips
- Quickly Detects and Prioritizes Issues -- and then Provides Usable Advice!
 - Brings Cache Performance Into Reach for Every Developer
 - Makes Experienced Cache Optimizers Hyper-Efficient

Features

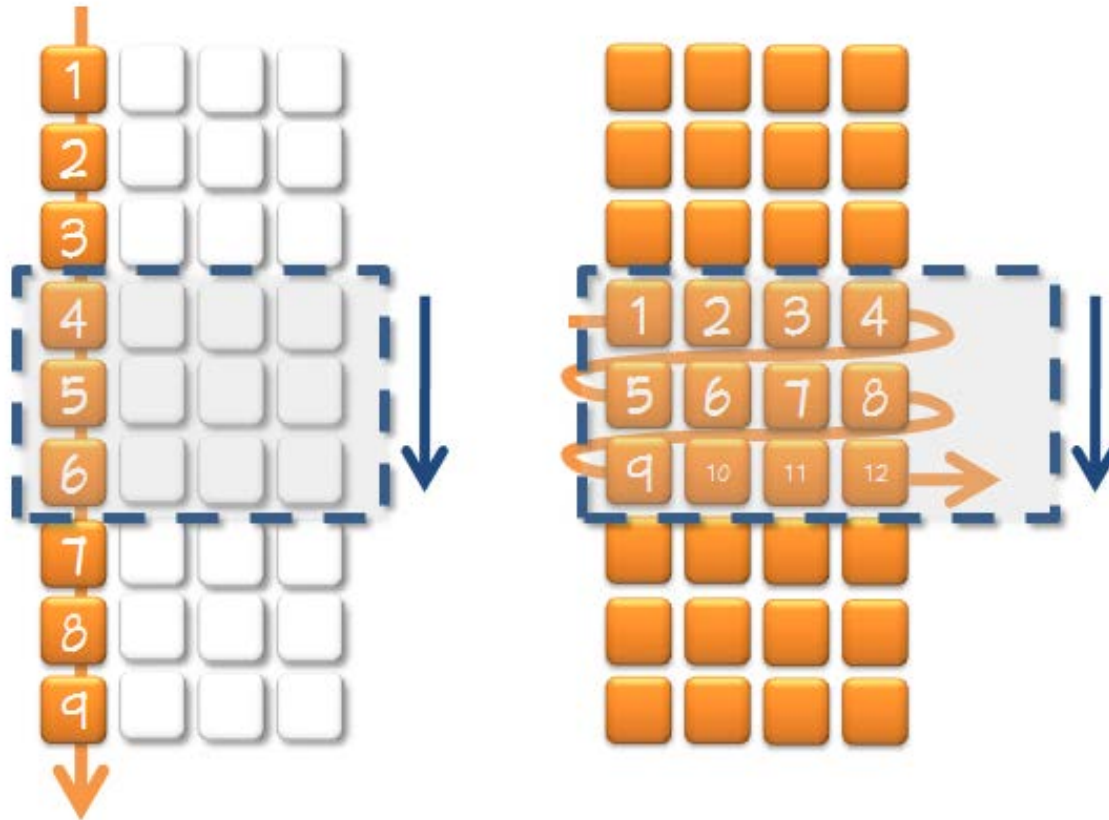
- Supports Linux x86/x86-64, Windows
- Any compiled code
- Runtime Analysis
 - Low overhead
- Cache Modeling
 - Prioritizes Issues
 - Identifies Problem Lines of Code
- Provides Advice
 - Explanations
 - Examples
 - Detailed statistics (if desired)

The screenshot displays the ThreadSpotter application interface. At the top, a browser window shows the URL: file:///C:/Documents%20and%20Settings/Erik/Mina%20do%20kument/Acumen/Presentation/SC07/Demo/acumen-report-art.html. The main interface is divided into several sections:

- Summary Table:** A table with columns for 'Loop / Issue', 'Summary', '% of fetches', 'Utilization', 'HW-Prefetch', and 'Randomness'. It lists issues like 'Poor utilization', 'Loop fusion', and 'Inefficient loop nesting'.
- Issue #8: Cache line utilization:** A detailed view of a specific issue. It includes a table of statistics for instructions in this issue, a line graph showing utilization over time, and a list of instructions involved in the issue.
- Code Editor:** A window showing assembly code with instructions highlighted in yellow, corresponding to the issues listed in the table.

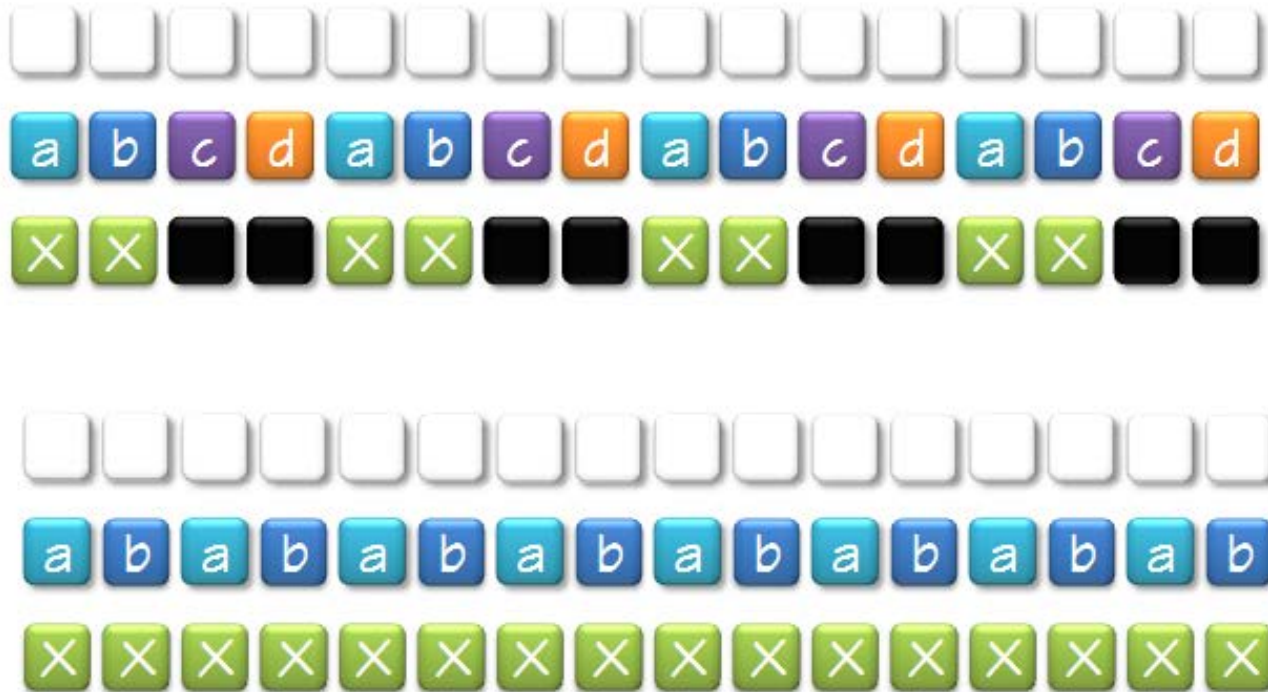
Inefficient Loop Nesting

Explanation



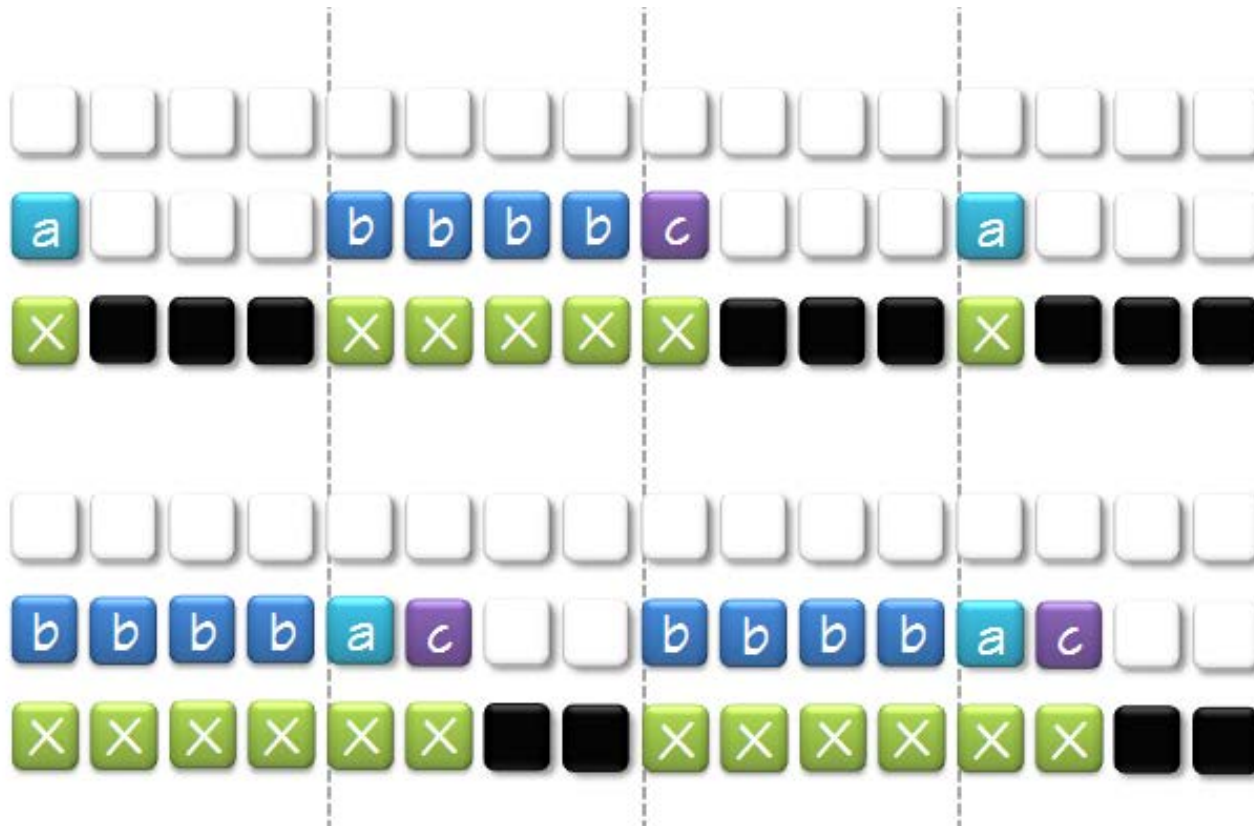
Partially Used Structure

Explanation



Alignment Problem

Explanation



Thank you!

Rogue Wave Software

Developing parallel, data-intensive applications is hard.
We make it easier.

Chris.Gottbrath@roguewave.com

Neil.Foster@roguewave.com