



Addressing the Increasing Challenges of Debugging on Accelerated HPC Systems

2012 Symposium on Application Accelerators in High-Performance Computing (SAAHPC'12)

Ed Hinkel
Senior Sales Engineer

Agenda

Overview - Rogue Wave & TotalView
GPU Debugging with TotalView
Nvidia CUDA
Intel Phi

Rogue Wave Today



The largest independent provider of cross-platform software development tools and embedded components for the next generation of HPC applications

Visual Numerics®

Leader in embeddable math and statistics algorithms and visualization software for data-intensive applications.



Leading provider of intelligent software technology which analyzes and optimizes computing performance in single and multi-core environments.



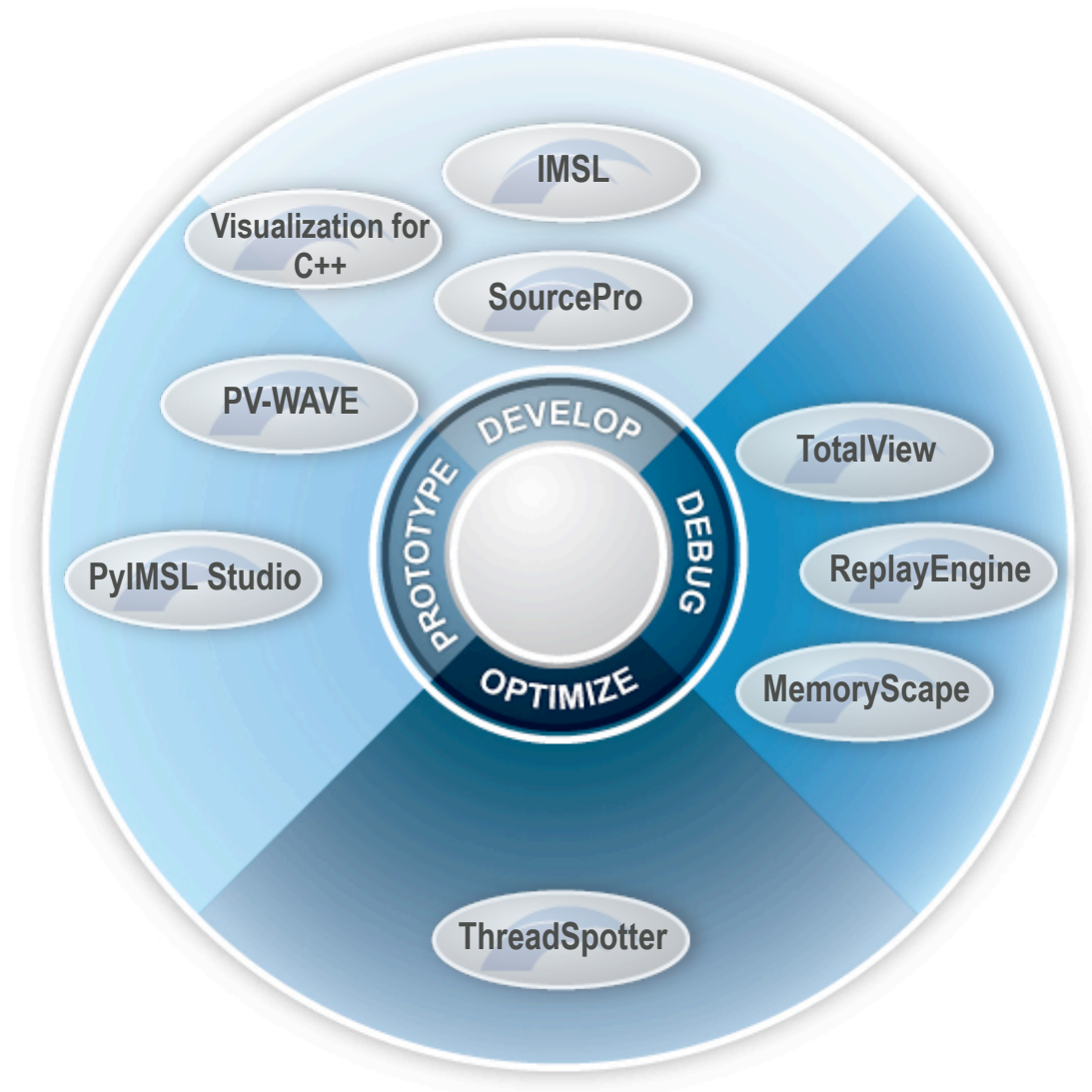
Industry-leading interactive analysis and debugging tools for the world's most sophisticated software applications.

Latest addition to the Rogue Wave family: **Rogue Wave Visualization for C++**
(Formerly IBM's ILOG Visualization for C++ products)

Representative Customers



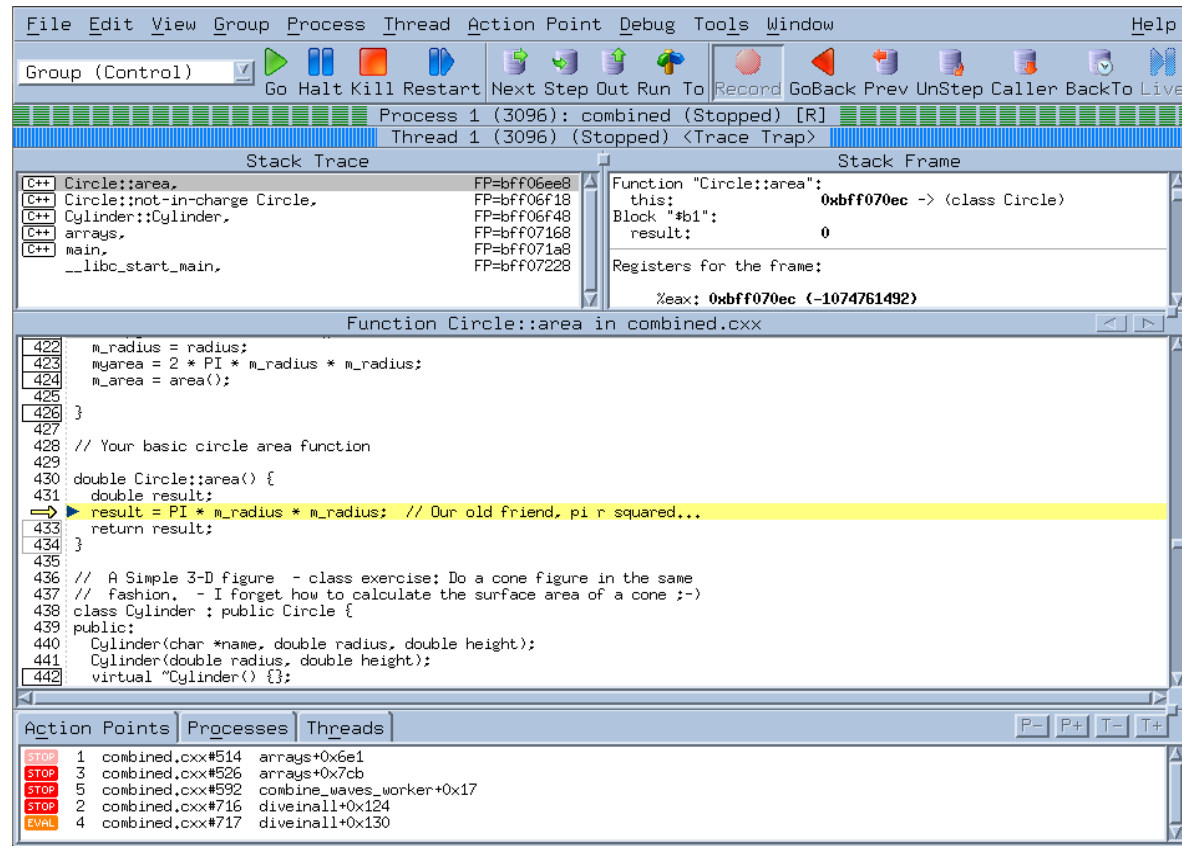
Rogue Wave Product Offerings



What is TotalView?

A comprehensive debugging solution for demanding parallel and multi-core applications

- **Wide compiler & platform support**
 - C, C++, Fortran 77 & 90, UPC
 - Unix, Linux, OS X
- **Handles Concurrency**
 - Multi-threaded Debugging
 - Multi-process Debugging
- **Heterogeneous Systems Support**
- **Integrated Memory Debugging**
- **Reverse Debugging for Linux**
- **Supports Multiple Usage Models**
 - Powerful and Easy GUI – Highly Graphical
 - CLI for Scripting
 - Long Distance Remote Debugging
 - Unattended Batch Debugging

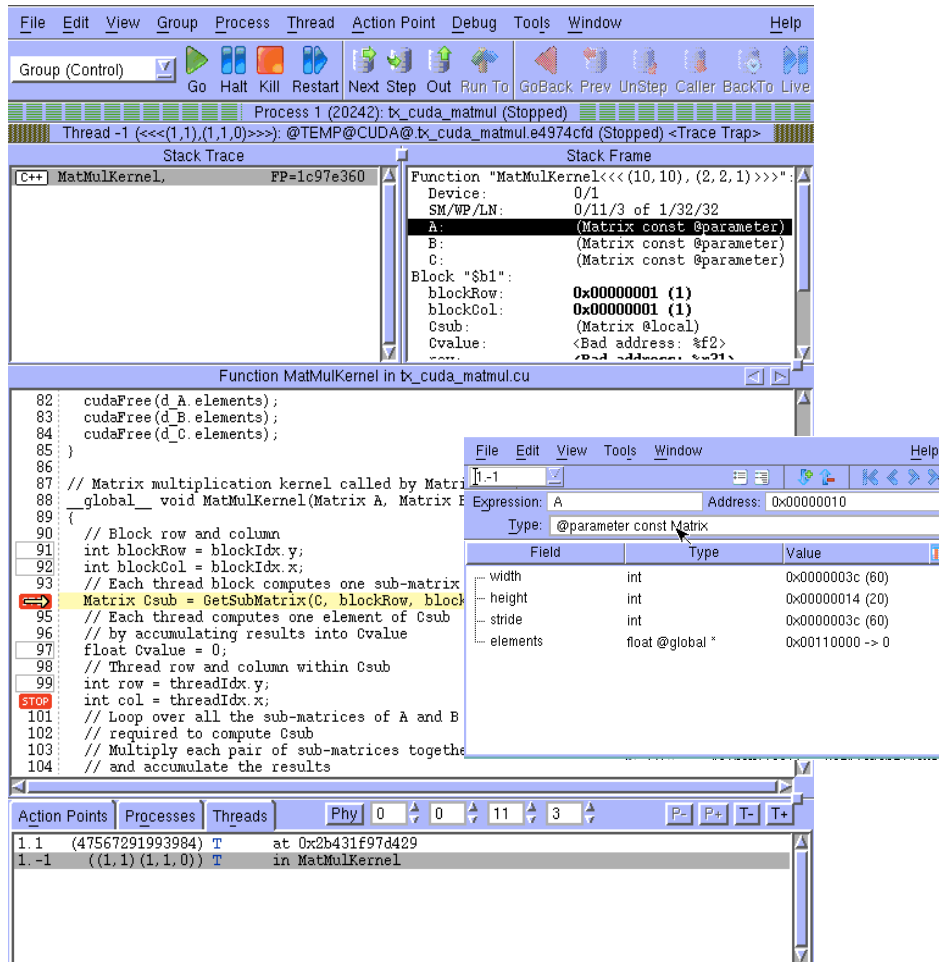


GPU Debugging

with

TotalView

CUDA Port of TotalView



Debugging on the GPU device (not in an emulator)

Full visibility of both Linux and GPU threads

Device threads shown as part of the parent Unix process

Handles all the differences between the CPU and GPU

Fully represent the hierarchical memory

Display data at any level (registers, local, block, global or host memory)

Making it clear where data resides with type qualification

Thread and Block Coordinates

Built in runtime variables display threads in a warp, block and thread dimensions and indexes

Displayed on the interface in the status bar, thread tab and stack frame

Device thread control

Warps advance synchronously

Handles CUDA function inlining

Step into or over inlined functions

Functions show on stack trace

Reports memory access errors

CUDA memcheck

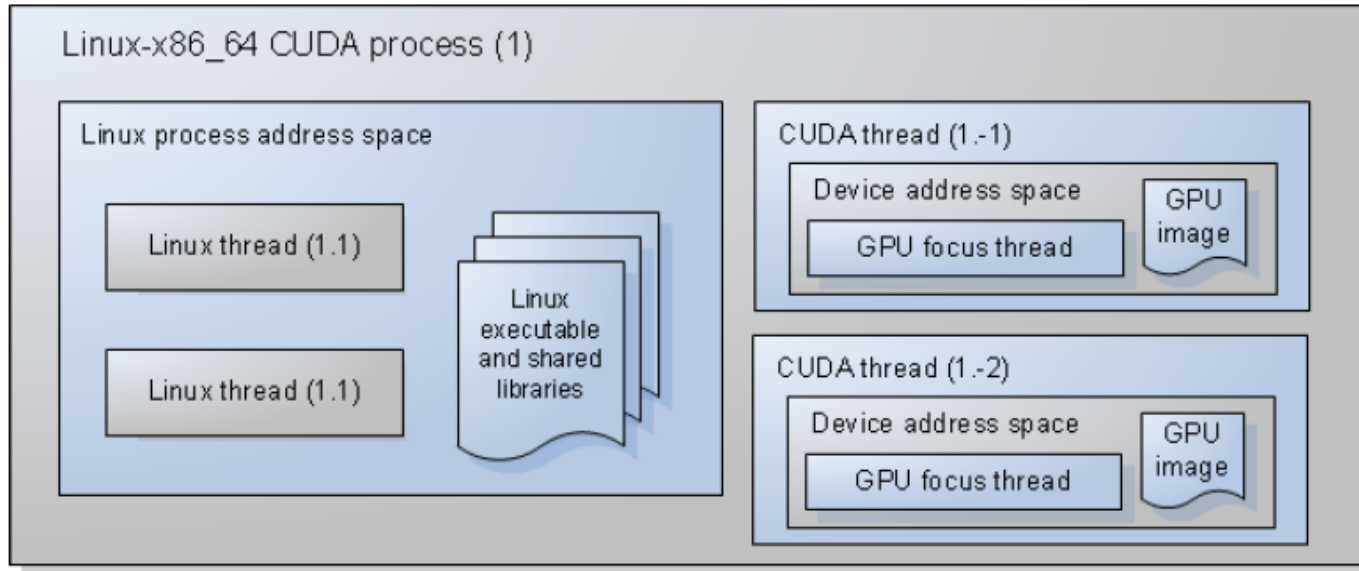
Multi-Device Support

Can be used with MPI

Debugging CUDA applications

- **Applications can take advantage of**
 - **Kernels execute asynchronously**
 - **Overlap of communication and computation**
 - **The same kernel can operate on multiple streams**
 - **Multi-process applications**
 - **Utilization of multiple GPUs at the same time**
 - **Multi-level parallelism**
 - **MPI + OpenMP + CUDA**
- **Interface rapidly developing with each release of the SDK**
 - **Rogue Wave is working closely with NVIDIA to take advantage of capabilities as they are introduced**

TotalView CUDA Debugging Model



- **A Linux-x86_64 CUDA process consists of:**
 - **A Linux process address space, containing:**
 - A Linux executable and a list of Linux shared libraries.
 - **A collection of Linux threads, where a Linux thread:**
 - Is assigned a positive debugger thread ID.
 - **A collection of CUDA threads, where a CUDA thread:**
 - Is assigned a negative debugger thread ID.
 - Has its own separate address space

Starting TotalView

File Edit View Group Process Thread Action Point Debug Tools Window Help

Group (Control) Go Halt Kill Restart Next Step Out Run To Record GoBack Prev UnStep Caller BackTo Live

Physical Device: 0 SM: 2 Warp: 0 Lane: 0

Process 1 (11290): tx_cuda_matmul (Stopped)

Thread -1 (<<<(0,0,0),(0,0,0)>>>): @TEMP@CUDA@tx_cuda_matmul.c621ea5d (Stopped) <Trace Trap>

Stack Trace

Address	Function	FP
C++	GetSubMatrix,	FP=ffcca0
C++	MatMulKernel,	FP=ffcca0

Stack Frame

Function "GetSubMatrix<<<(1, 1), (2, 2, 1)>>>":

Device: 0/3
SM/WP/LN: 2/0/0 of 14/48/32
A: (Matrix @local)
row: 0x00000000 (0)
col: 0x00000000 (0)

Local variables:
Asub: (Matrix @local)

Registers for the frame:
r0: 0x00000000 (0)

Function GetSubMatrix in tx_cuda_matmul.cu

```
34
35 // Get the BLOCK_SIZExBLOCK_SIZE sub-matrix Asub of A that is
36 // located col sub-matrices to the right and row sub-matrices down
37 // from the upper-left corner of A
38 __forceinline__ __device__ Matrix GetSubMatrix(Matrix A, int row, int col)
39 {
40     Matrix Asub;
41     Asub.width = BLOCK_SIZE;
42     Asub.height = BLOCK_SIZE;
43     Asub.stride = A.stride;
44     Asub.elements = &A.elements[A.stride * BLOCK_SIZE * row
45                     + BLOCK_SIZE * col];
46     return Asub;
47 }
48
49 // Forward declaration of the matrix multiplication kernel
50 __global__ void MatMulKernel(const Matrix, const Matrix, Matrix);
51
52 // Matrix multiplication - Host code
53 // Matrix dimensions are assumed to be multiples of BLOCK_SIZE
54 void MatMul(const Matrix A, const Matrix B, Matrix C)
```

Action Points Processes Threads

Address	Process	Thread
STOP	1	tx_cuda_matmul.cu#41 GetSubMatrix+0x08...
STOP	2	tx_cuda_matmul.cu#46 GetSubMatrix+0x1b0...

@TEMP@CUDA@tx_cuda_matmul.c621ea5d (1.-1) has loaded a CUDA GPU image.
Stop so you can set breakpoints?

Don't ask this question again.

Yes No

- Once a new kernel is loaded TotalView provides the option to stop and set breakpoints
- TotalView automatically configures the GUI for CUDA debugging
- Debugging CUDA code is done by using normal TotalView commands and procedures

GPU Device Status Display

Provides the “high-level” view

- Values automatically update as you step through code
- Shows what hardware is in use
- Helps to 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

GPU Device Status Display

Provides detailed information for:

Device and Type

SMs

Warps

Lanes with PC

Name	Description
Device 0/3	
Device Type	gf100
Lanes	32
SM 2/1	
Valid Warps	000000
Warp 00/48	Block
Lane 00/32	Thread
LPC	000000
Lane 01/32	Thread
LPC	000000
Lane 02/32	Thread
LPC	000000
Lane 03/32	Thread
LPC	000000
Lane 04/32	Thread
LPC	000000
Lane 05/32	Thread
LPC	000000

Information updates as you step

GPU Device Status Display

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

It also provides information for divergent GPU threads

Different PC for two groups of Lanes

State of Lanes inside the Warp

Debugging CUDA

Information on GPU execution, location and data is readily available.
... the same as it is for Linux processes and threads.

The screenshot shows a debugger window for a CUDA application. The title bar indicates the file path: `/nfs/nvidia5/u0/home/kduthie/bld/8_10_mainline/debugger/src/tests/bld/nvcc_4.1RC2_...`. The menu bar includes File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, and Help. The toolbar contains various debugging actions like Go, Halt, Kill, Restart, Next Step, Out, Run To, Record, GoBack, Prev, UnStep, Caller, and BackTo Live. Below the toolbar, the 'Logical' tab is active, showing the current thread as 'Thread -1 (<<<(0,0,0),(1,1,0)>>>): @TEMP@CUDA@tx_cuda_matmul.c621ea5d (At Breakpoint 1)'. The 'Stack Trace' pane shows the current function: 'MatMulKernel, FP=fffc0'. The 'Stack Frame' pane displays the function signature: 'Function "MatMulKernel<<<(1,1,1),(2,2,1)>>>": Device: 0/3; SM/WP/LN: 2/0/3 of 14/48/32; A: (Matrix @local); B: (Matrix @local); C: (Matrix @local); Block "\$b1#\$b1#\$b1": Csub: (Matrix @local); blockRow: 0x00000000 (0); blockCol: 0x00000000 (0); Cvalue: <Bad address: 0x00000000>'. The main code window shows the source code for 'Function MatMulKernel in tx_cuda_matmul.cu'. Line 93 is highlighted: 'Matrix Csub = GetSubMatrix(C, blockRow, blockCol);'. The 'Action Points' pane at the bottom shows a list of threads: '1.1 (47350702036240) T in cudbgApiInit', '1.2 (1095072064) T in __select_nocancel', and '1.-1 ((0,0,0)(1,1,0)) B1 in MatMulKernel'. The mouse cursor is pointing at the '1.-1' thread.

Debugging CUDA

The screenshot shows a debugger window with the following components:

- Thread Information:** Thread 1 (31565) is stopped at a breakpoint in `tx_cuda_matmul.c621ea5d`.
- Stack Frame:** The current frame is for the function `MatMulKernel<<<(1, 1, 1), (2, 2, 1)>>>`. It lists parameters: `Device: 0/3`, `SM/WP/LN: 2/0/3 of 14/48/32`, and matrices `A`, `B`, and `C`. It also shows block coordinates: `Block "$b1#$b1#$b1": Csub: (Matrix @local), blockRow: 0x00000000 (0), blockCol: 0x00000000 (0), Cvalue: <Bad address: 0x00000000>`.
- Code Window:** Shows the kernel's implementation, with the line `Matrix(C, blockRow, blockCol);` highlighted in yellow. The comment above it reads: `computes one sub-matrix Csub of C`.

CUDA grid and block dimensions, lanes/warp, warps/SM,

Parameter, register, local and shared variables

Debugging CUDA

GPU focus
thread logical
coordinates in
the header...



File Edit View Group Process Thread Action Point Debug Tools Window

Group (Control) Go Halt Kill Restart Next Step Out Run To Record Go Back

Logical Block: 0, 0, 0 Thread: 1, 1, 0

Process 1 (31565): tx_cuda_matmul (At Breakpoint 1)
Thread -1 (<<<(0,0,0),(1,1,0)>>>): @TEMP@CUDA@tx_cuda_matmul.c621

Stack Trace

Function	FP	Stack
C++ MatMulKernel,	FP=fffca0	

Function "MatMulKernel<<<...>>>":
Device: 0/3
SM/WP/LN: 2/0/0
A: (Mat...)
B: (Mat...)
C: (Mat...)
Block "\$b1#\$b1#\$b1":
Csub: (Mat...)
blockDim: 0x000, 0x000, 0x000
Cvalue: <Bad...>

Function MatMulKernel in tx_cuda_matmul.cu

```
84 cudaFree(d_C.elements);  
85 }  
86  
87 // Matrix multiplication kernel called by MatrixMul()  
88 __global__ void MatMulKernel(Matrix A, Matrix B, Matrix C)  
89 {  
90 // Block row and column  
91 int blockDim = blockDim.y;  
92 int blockDim = blockDim.x;
```



Debugging CUDA

```
87 // Matrix multiplication kernel called by Matr
88 __global__ void MatMulKernel(Matrix A, Matrix
89 {
90     // Block row and column
91     int blockRow = blockIdx.y;
92     int blockCol = blockIdx.x;
93     // Each thread block computes one sub-matrix
94     Matrix Csub = GetSubMatrix(C, blockRow, bloc
95     // Each thread computes one element of Csub
96     // by accumulating results into Cvalue
97     float Cvalue = 0;
98     // Thread row and column within Csub
99     int row = threadIdx.y;
100    int col = threadIdx.x;
101    // Loop over all the sub-matrices of A and B
```

... as well as in
the Process
Window

Action Points	Processes	Threads	
1. 1	(47350702036240)	T	in cudbgApiInit
1. 2	(1095072064)	T	in __select_nocancel
1. -1	((0, 0, 0) (1, 1, 0))	B1	in MatMulKernel

Debugging CUDA

PC arrow shows the Program Counter for the warp

The screenshot shows a debugger interface. At the top, a 'Stack Trace' window displays the current function: 'C++ MatMulKernel, FP=fffca0'. To the right, a panel shows details for the current thread, including 'Function "M', 'Device:', 'SM/WP/LN:', 'A:', 'B:', 'C:', 'Block "\$b1#', 'Csub:', 'blockRow:', 'blockCol:', 'Cvalue:', and 'row:'. Below the stack trace, the 'Function MatMulKernel in tx_cuda_ma' is open, showing C++ code. Line 93 is highlighted in yellow, and a red arrow points to it from the text box on the left. The code includes comments and function calls like 'cudaFree(d_C.elements);', 'GetSubMatrix(C, blockRow, blockCol)', and 'Loop over all the sub-matrices of A and B that'. At the bottom, a table shows 'Action Points', 'Processes', and 'Threads'.

Action Points	Processes	Threads
1. 1	(47350702036240)	T in cudbgApiInit
1. 2	(1095072064)	T in __select_nocancel
1. -1	((0, 0, 0) (1, 1, 0))	B1 in MatMulKernel

Debugging CUDA

The screenshot shows a debugger interface with the following components:

- Debugger Window:** Shows the current process and thread. The thread is identified as "Thread -1 (<<<(0,0,0),(1,1,0)>>>): @TEMP@CUDA@tx_cuda_matm".
- Stack Trace:** Shows the current function call: "C++ MatMulKernel, FP=fffca0".
- Variable Window:** A window titled "A - MatMulKernel - 1.-1" is open, showing the variable "A". The expression is "A" and the address is "0x00000010". The type is "@parameter const Matrix".
- Matrix Structure:** A table below the variable window shows the structure of the matrix "A":

Field	Type	Value
width	int	0x00000002 (2)
height	int	0x00000002 (2)
stride	int	0x00000002 (2)
elements	float @global *	0x00110000 -> 0
- Code Window:** Shows the source code of the kernel. Line 91 is highlighted, and a red arrow points to it from the variable window.

Dive on any variable name to open a variable window

Debugging CUDA

The screenshot shows a debugger window titled "A - MatMulKernel - 1.-1". The menu bar includes File, Edit, View, Tools, Window, and Help. The toolbar contains various navigation and search icons. The "Expression:" field contains "A" and the "Address:" field contains "0x00000010". The "Type:" field contains "@parameter const Matrix". Below this is a table with columns "Field", "Type", and "Value".

Field	Type	Value
width	int	0x00000002 (2)
height	int	0x00000002 (2)
stride	int	0x00000002 (2)
elements	float @global *	0x00110000 -> 0

Four callout boxes provide additional information:

- "@parameter" type qualifier indicates that variable "A" is in parameter storage
- Address 0x10 is an offset within parameter storage
- Pointer value 0x110000 is an offset within global storage
- "elements" is a pointer to a float in global storage

Storage Qualifiers

- Denotes location in hierarchical memory
 - Part of the type – using “@” notation
 - Each memory space has a separate address space so 0x00001234 could refer to several places

Storage Qualifier	Meaning
@parameter	Address is an offset within parameter storage.
@local	Address is an offset within local storage.
@shared	Address is an offset within shared storage.
@constant	Address is an offset within constant storage.
@global	Address is an offset within global storage.
@register	Address is a PTX register name

- Used throughout expression system
 - You can cast to switch between different spaces

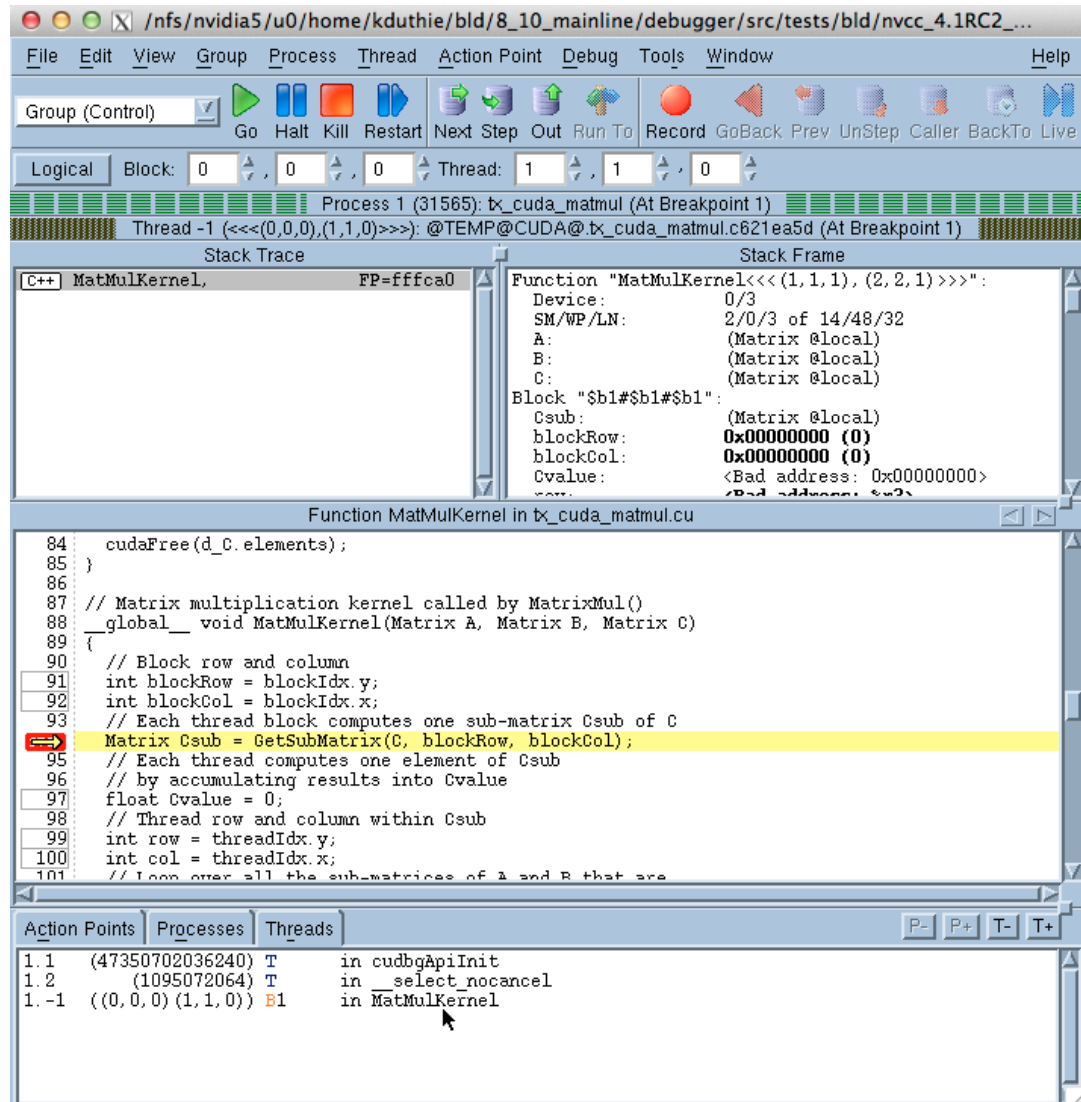
The screenshot shows a debugger window with two panes. The top pane displays details for a variable named 'Bsub'. The expression is 'Bsub', the address is '0x00000120', and the type is '@local Matrix'. Below this, a table lists fields: width (int, 0x00000002 (2)), height (int, 0x00000002 (2)), stride (int, 0x00000014 (20)), and elements (float @global *, 0x00111310 -> 4).

The bottom pane shows a list of expressions and their values. The 'Asub' expression is highlighted in blue. A mouse cursor points to the 'Bs[row][col]' expression.

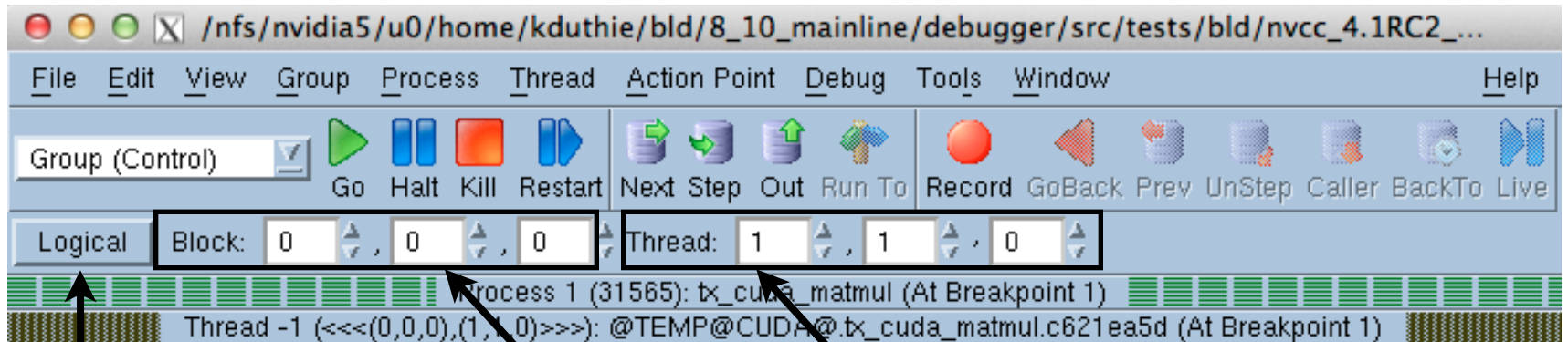
Expression	Type	Value	
Bs[row][col]	float	812	0x0000000000000000
row	@register int	0x00000000 (0)	%r31
col	@register int	0x00000000 (0)	%r33
Bs[1][0]	float	832	0x0000000000000000
Bs[row+1][1]	float	833	0x0000000000000000
row+1	@register int	0x00000001 (1)	(None)
A	@parameter const Matrix	(Matrix const @parameter)	0x0000000000000000
Asub	@local Matrix	(Matrix @local)	0x0000000000000000
*(Asub.elements)	@global float	20	0x0000000000000000

Debugging CUDA - Navigation

Navigate through your CUDA code in the Process Window as you wish... Using either of two coordinate systems:



Debugging CUDA - Navigation



CUDA GPU threads have a negative TotalView thread ID

Block (x,y,z)

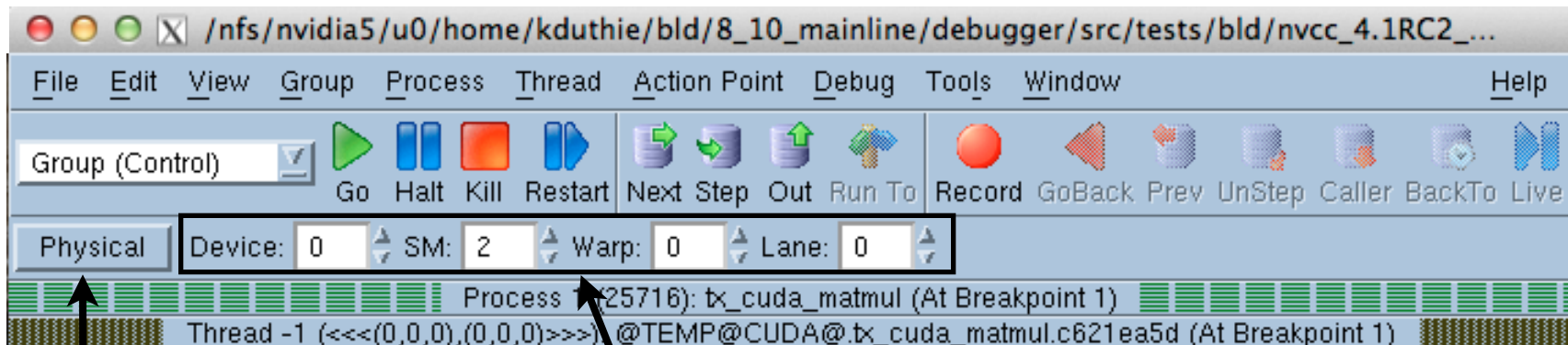
Thread (x,y,z)

User-controlled “spinboxes” allow selection and display of any part of your GPU execution

GPU focus thread selector for changing the logical block and thread indexes of the CUDA thread.

- Logical: 2 or 3D Grid of Blocks, 3D Thread Within Grid

Debugging CUDA - Navigation



Device, SM, Warp,
and Lane

User-controlled “spinboxes” allow selection and display of any part of your GPU execution

GPU focus selector for changing physical indexes of the CUDA thread.

- Physical: Device, SM, Warp, Lane

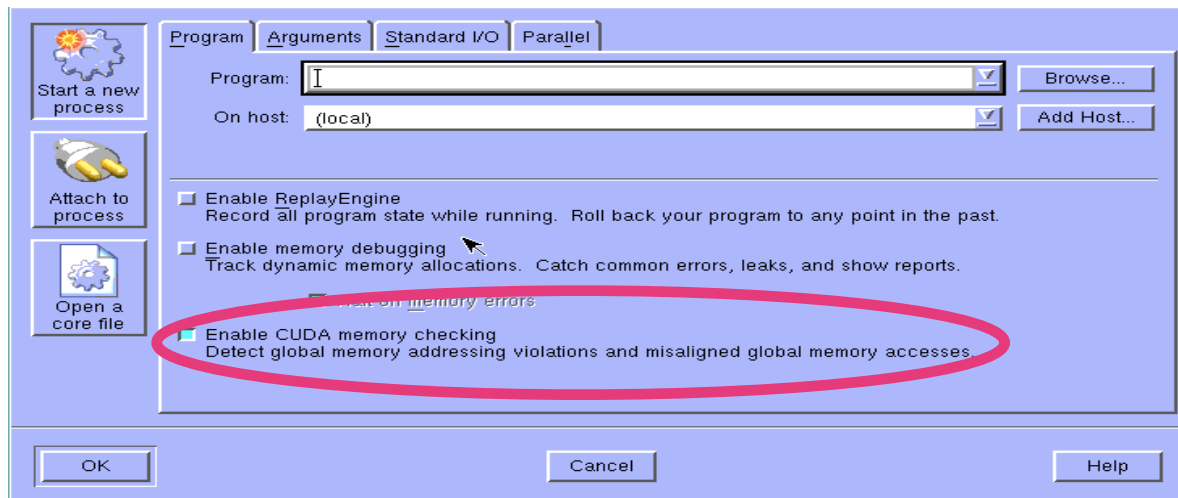
Executing GPU Code - Threads and Warps

Execution Control

- **Single-step operation advances all of the GPU hardware threads in the *same* warp**
- **To advance the execution of more than one warp:**
 - set a breakpoint and continue the process, or
 - select a line number in the source pane and select “Run To”.
- **Warps advance synchronously**
 - Warps share a PC
- **Single stepping**
 - Advances the warp containing the focus thread
 - Stepping over a `__syncthreads()` call advances all the relevant threads
- **Halt**
 - Stops all the host and device threads

CUDA Segmentation Faults

- **TotalView displays segmentation faults as expected**
 - **Enable CUDA memory checking in New Program dialog window**



TV 8.10 support for CUDA 4.1 specific features

- **Works with the CUDA 4.1 SDK and Runtime**
 - New Compiler Front End
 - Slight differences in the debug API
- **Support for no copy pinned memory**
 - This was broken at the driver level in 4.0
- **New support for CUDA device assertions**
- **New support for multiple CUDA contexts from the same process on the same device**
- **Support for CUDA on the Cray XK environment**

OpenACC

- **Pragma Based**
 - Similar to OpenMP
 - Supports C and Fortran
- **OpenACC Vendors**
 - Cray
 - PGI
 - CAPS
- **TotalView early access with Cray CCE 8**
- **Working with PGI and CAPS towards support**

OpenACC

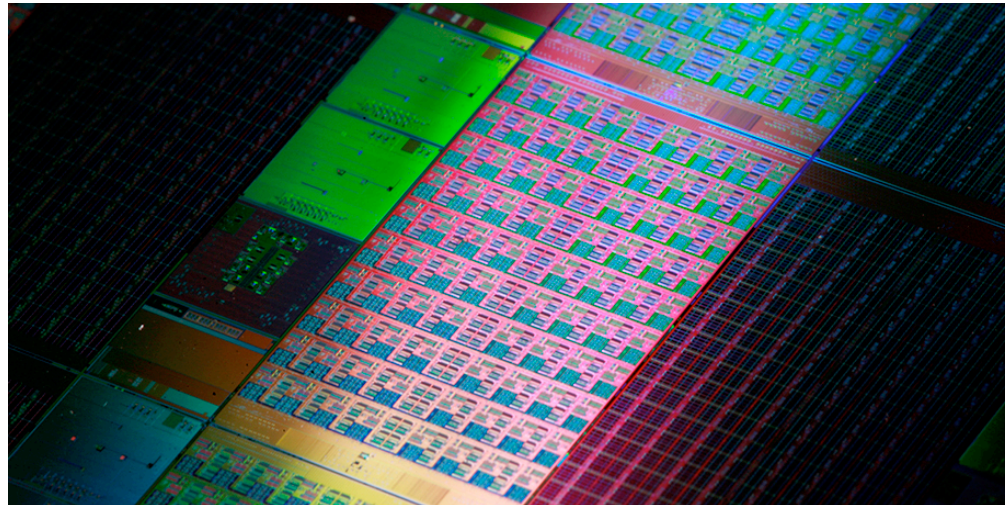
The screenshot displays a development environment with several windows:

- Terminal (n15765@vista:~/.ACC_tests/ftn_test):** Shows the compilation and linking process for an OpenACC program. It details the mapping of ELF string data and symbols from the host to the device, and the linking of the resulting object files into an executable.
- TotalView 8X 10.0-6A:** A process monitoring tool showing a table of active threads. The table has columns for ID, Rank, Host, Status, and Description. It lists two threads: one local thread (rank 1) and one remote thread (rank 2) on host nid00130, both with status 'T' (running).
- Debugger (aprun-<man>.0):** A graphical debugger window showing the execution of the program. The main window displays the source code of the function `test_openacc_sck_L11_1` in `man.f90`. The code includes a parallel loop for setting variables `a`, `b`, and `c`. The debugger has stopped at line 15, which is the end of the first parallel loop. The right-hand pane shows the current stack frame, including the function name, device ID (0/1), and the values of the arguments `a`, `b`, and `c`, all of which are 1. The bottom pane shows the Action Points window, listing the current stop points in the program.

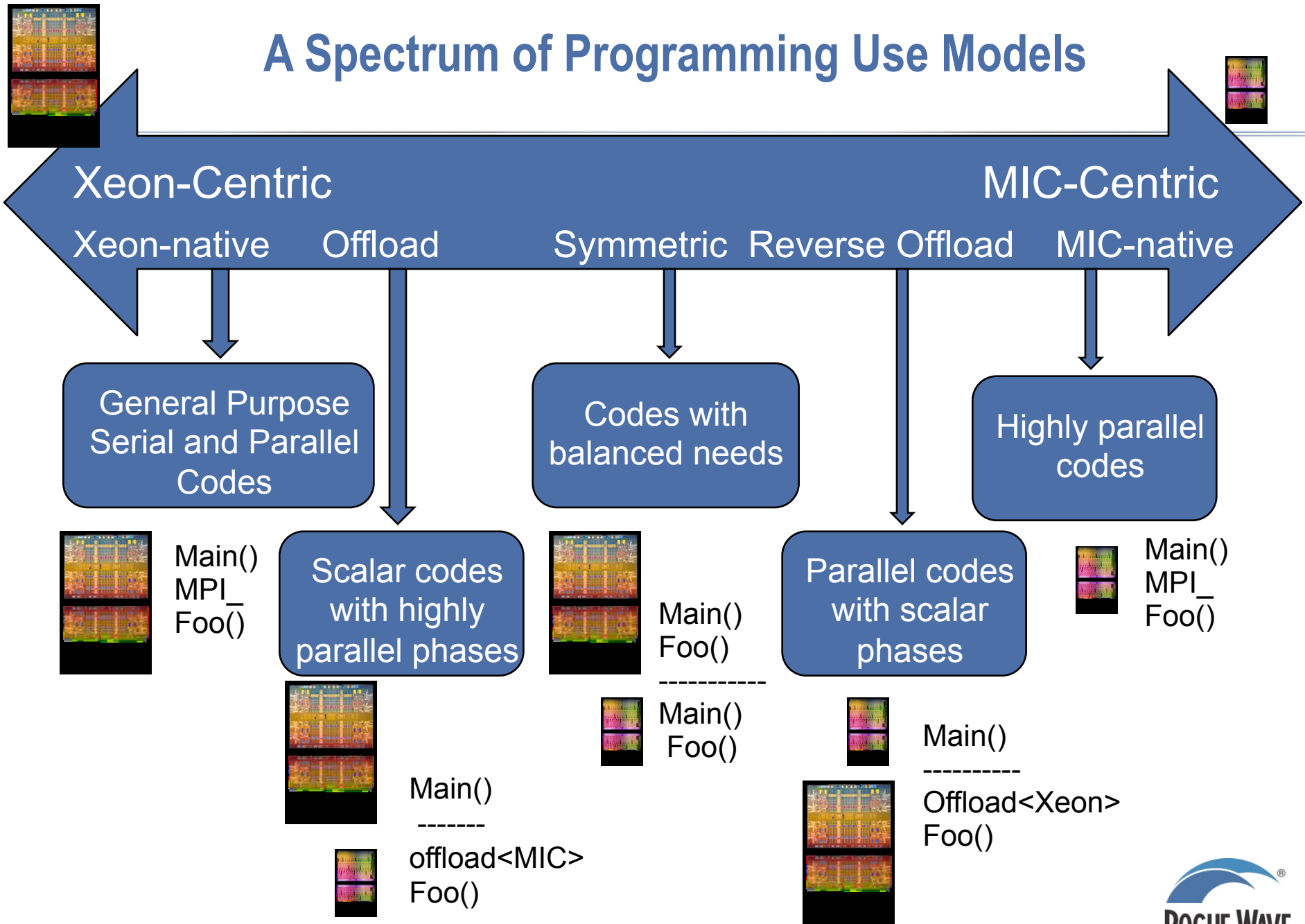
Intel Phi Debugging

with

TotalView



A Spectrum of Programming Use Models



Intel MIC Port of TotalView

The screenshot displays the Intel TotalView debugger interface. At the top, the menu bar includes File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, and Help. Below the menu is a toolbar with icons for Go, Halt, Kill, Restart, Next Step, Out, Run To, Record, Go Back, Prev, UnStep, Caller, Back To, and Live. The main window is divided into several panes:

- Stack Trace:** Shows a list of stack frames for process 2 (5856@192.168.1.100) and thread 2 (139985823807232). The current frame is 'compute07'.
- Stack Frame:** Displays details for the 'compute07' function, including local variables (out, size, i) and registers for the frame.
- Source Code:** Shows the source code for 'Function compute07 in sampleC07.c'. The current line is highlighted: 'for (i=0; i<size; i++)'.
- Process/Thread Table:** A table showing the status of various processes and threads. The table has columns for ID, Rank, Host, Status, and Description.

ID	Rank	Host	Status	Description
1		<local>	R	/opt/intel/composerx64/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		192.168.1.10	M	/tmp/coi_procs/1/5856/offloa
2.1		192.168.1.1	R	in sem_wait
2.2		192.168.1.1	B6	in compute07
2.3		192.168.1.1	R	in __poll
2.4		192.168.1.1	R	in pthread_cond_wait

- Full visibility of both host and Phi threads
- Full support of MPI programs
- Symmetric Debugging of heterogeneous applications with offloaded code
- Remote Debugging of Phi-native applications
- Asynchronous thread control on both Xeon and Phi



Debugging MPI Applications

The screenshot displays the Rogue Wave software interface for debugging MPI applications. The main window shows the source code of a program named `tx_basic_mpi.c`. The code is as follows:

```
84  
85 message = getenv("MESSAGE") ? getenv("MESSAGE") : "";  
86  
87 MPI_Init(&argc,&argv);  
88 MPI_Comm_size(MPI_COMM_WORLD,&numprocs);  
89 MPI_Comm_rank(MPI_COMM_WORLD,&myid);  
90  
91 if (myid == 0)  
92 {  
93     char* outfile = NULL;  
94     outfile = getenv("TX_OUTFILE");  
95  
96     if (outfile)  
97         out = fopen(outfile, "w");  
98  
99     if (!out)  
100         out = stdout;  
101  
102     rank0(numprocs, out);  
103 }  
104 else  
105     rankn(myid);  
106
```

The interface also shows a stack trace for the current thread, a dialog box for selecting processes to attach to, and a process table at the bottom.

Stack Trace:

```
C main, FP=7fff0e864b60  
C __libc_start_main, FP=7fff0e864c20  
_start, FP=7fff0e864c30
```

Function "main":

```
argc: 0x00000001  
argv: 0x7fff0e86  
Block "#1":  
outfile: 0x00000000  
Local variables:  
myid: 0x00000000  
numprocs: 0x00000014  
world: 0x44000000  
out: 0x00000000
```

Select processes to attach to: (20 showing, 0 filtered, 20 total)

Attach	Host	Comm	Rank	Program
<input checked="" type="checkbox"/>	192.168.1.100	0	0	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	1	1	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	2	2	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	3	3	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	4	4	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	5	5	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	6	6	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	7	7	/tmp/tx_basic_mpi
<input checked="" type="checkbox"/>	192.168.1.100	8	8	/tmp/tx_basic_mpi

Filters:

- Communicator: All
- Array of Ranks:
- Talking to Rank: All
- List of Ranks:
- Message Type: Send Receive Unexpected

Process Table:

Process	Rank	Host
p1	0	192.168.1.100
	1	192.168.1.100
	2	192.168.1.100
	3	192.168.1.100
	4	192.168.1.100
	5	192.168.1.100
	6	192.168.1.100
	7	192.168.1.100
	8	192.168.1.100
	9	192.168.1.100
	10	192.168.1.100
	11	192.168.1.100
	12	192.168.1.100
	13	192.168.1.100
	14	192.168.1.100
	15	192.168.1.100
	16	192.168.1.100
	17	192.168.1.100
	18	192.168.1.100
	19	192.168.1.100

- Attach to subset of processes on Phi
- Set breakpoints
- Debug MPI “as usual”

Remote Debugging of Applications on Phi

The screenshot displays a remote debugger interface with the following components:

- Menu Bar:** File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, Help.
- Toolbar:** Go, Halt, Kill, Restart, Next Step, Out, Run To, Record, Go Back, Prev, UnStep, Calle.
- Status Bar:** Process 1 (77308192.168.1.100): omp_offload_native (At Breakpoint 5) / Thread 1 (140269718562624) (Stopped).
- Stack Trace:** Lists call stack entries with frame pointers (FP). The current frame is `mmul` at FP=7ffff1c6226f0.
- Stack Frame:** Shows function parameters: `a: 0x7f93164c9a20 -> 2,033`, `lda: 0x1c6226c0 (476194496)`, `b: 0x00000003 -> <Bad addr>`, `ldb: 0x1c622730 (476194608)`, `c: 0x00000040 -> <Bad addr>`, `ldc: 0x000000c0 (192)`, `n: 0x1c622f10 (476196624)`. Local variables: `iMaxThreads: 0x167ad398 (377148312)`. Registers for the frame: `%rax: 0x00000000 (0)`, `%rdx: 0x0000000a (10)`, `%rcx: 0x00602308 (6300424)`, `%rbx: 0x7ffff1c6225e8 (140733669582)`, `%rsi: 0x00000009 (9)`, `%rdi: 0x00602308 (6300424)`.
- Source Code:** Shows the `mmul` function in `omp_offload_native.cpp`. The current line is `c[j * ldc + i] += a[k * lda + i] * b[j * ldb + k];` at line 34, which is highlighted in yellow.
- Thread List:** Shows a list of threads, with thread 1.3 (ID: 140269687383908) highlighted in blue.

- Start application on Phi card
- Attach to running application
- See thread private data
- Investigate individual threads
- Kill stuck processes on Phi



Debugging Applications with Offloaded Code

Xeon side

File Edit View Group Process Thread Action Point Debug Tools Window

Group (Control) Go Halt Kill Restart Next Step Out Run To Record GoBack

Process 1 (31634): intro_sampleC out (Stopped)

Thread 1 (140091609065248) (Stopped)

Stack Trace

Function "sample08":
No parameters.
Local variables:
pi: 0
count: 0x000027
i: 0x000000
sample08: 0x000000
main: 2.52234e
__libc_start_main: 5.91753e
_start: 0
t: 1.26117e

Function sample08 in sampleC08.c

```
38 // Sample 08 .....  
39 // This sample demonstrates how #pragma offload placed in front of  
40 // an OpenMP construct enables OpenMP  
41 //  
42 //  
43 // Effectively, this is heterogeneous OpenMP  
44 void sample08()  
45 {  
46     float pi = 0.0F;  
47     int count = 10000;  
48     int i;  
49  
50     #pragma offload target (mic)  
51     #pragma omp parallel for reduction(+:pi)  
52     for (i=0; i<count; i++)  
53     {  
54         float t = (float)((i+0.5F)/count);  
55         pi += 4.0F/(1.0F+t*t);  
56     }  
57     pi /= count;  
58  
59     if (fabs(pi-3.14F) <= 0.01F)  
60         printf("PASS Sample08 Pi = %F\n", pi);  
61     #else  
62         printf("*** FAIL Sample08 Pi = %F\n", pi);  
63     #endif  
64 }  
65
```

Action Points Processes Threads

Action Points	Processes	Threads
1.1 (140091609065248) T		in pthread_cond_wait
1.2 (140091596920592) T		in __poll
1.3 (140091586430736) T		in __poll
1.4 (140091575940880) T		in __poll
1.5 (140091565442832) T		in pthread_cond_wait

Phi side

File Edit View Tools Window Help

Process 2 (79768192 168 1 100): offload_main (Mixed)

Thread 2 (139773936764672) (At Breakpoint 6)

Stack Trace

Function "L_sample08_51_par_loop1_2_19":
No parameters.
Local variables:
count: 0x00002710 (10000)
pi: 0
t: 5e-05
i: 0x00000000 (0)

Registers for the frame:
%rax: 0x00000341 (833)

Function L_sample08_51_par_loop1_2_19 in sampleC08.c

```
44 void sample08()  
45 {  
46     float pi = 0.0F;  
47     int count = 10000;  
48     int i;  
49  
50     #pragma offload target (mic)  
51     #pragma omp parallel for reduction(+:pi)  
52     for (i=0; i<count; i++)  
53     {  
54         float t = (float)((i+0.5F)/count);  
55         pi += 4.0F/(1.0F+t*t);  
56     }  
57     pi /= count;  
58  
59     if (fabs(pi-3.14F) <= 0.01F)  
60         printf("PASS Sample08 Pi = %F\n", pi);  
61     #ifdef DEBUG  
62         printf("PASS Sample08 Pi = %F\n", pi);  
63     else  
64         printf("*** FAIL Sample08 Pi = %F\n", pi);  
65     #endif  
66 }
```

Action Points Processes Threads

Action Points	Processes	Threads
2.1 (139774009124800) R		in sem_wait
2.2 (139773936764672) R6		in L_sample08_51_par_loop1_2_19
2.3 (139773947401984) R		in __poll
2.4 (139773955794688) R		in pthread_cond_wait
2.5 (139773922080512) R		in pthread_cond_timedwait
2.6 (139773913687808) R6		in L_sample08_51_par_loop1_2_19
2.7 (139773909489408) R6		in L_sample08_51_par_loop1_2_19
2.8 (139773906991008) R6		in L_sample08_51_par_loop1_2_19

One debugging session for Phi-accelerated code



TotalView provides a full spectrum of debugging solutions



TotalView®

Code debugging

- Highly scalable interactive GUI debugger
- Powerful features for debugging multi-threaded, multi-process, and MPI parallel programs
- Compatible with wide variety of compilers across several platforms and operating systems

- **Memory Debugging**

- Parallel memory analysis and error detection
 - Intuitive for both intensive and infrequent users
- Easily integrated into the validation process

- **Reverse Debugging**

- Parallel record and deterministic replay within TotalView
 - Users can run their program “backwards” to find bugs
- Allows straightforward resolution of otherwise stochastic bugs

- **GPU CUDA Debugging**

- Full Hybrid Architecture Support
- Asynchronous Warp Control
- Multi-Device and MPI Support

- **Intel Phi Debugging**

- Multi-Model Support
- Early Experience Program Available

Summary, Conclusions, Next Steps and Questions

- **Rogue Wave Software provides a suite of tools and libraries that can accelerate your software engineering efforts**
- **TotalView provides comprehensive hybrid debugging environment**
- **Initial support for Intel PHI has begun.**
 - **Including support for multiple workflows**
 - **We are interested in learning the workflows that you are going to apply**
 - **If you are interested in our early experience program, let us know.**
- **Questions?**

Thanks!



Thank You

Download a TotalView evaluation at:

[*www.roguewave.com/products*](http://www.roguewave.com/products)

Academic License

[*tvexpress@roguewave.com*](mailto:tvexpress@roguewave.com)