TotalView on CORAL Update

LLNL 10/24/18

Peter Thompson

TotalView Agenda

- TotalView Debugging on Sierra CORAL with GPUs and OpenMP
- CUDA Unified Breakpoints Project
- TotalView New UI
- OpenMP Debugging (OMPD) Standardization Effort
- TotalView Road Map
- TotalView Technical Support

Peter Thompson: TotalView Specialist

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView Agenda

- TotalView Debugging on Sierra CORAL with GPUs and OpenMP
- CUDA Unified Breakpoints Project
- TotalVIew New UI
- OpenMP Debugging (OMPD) Standardization Effort
- TotalView Roadmap
- TotalView Technical Support

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView on CORAL

- TotalView has been available on CORAL EA: rzmanta, ray and shark (for almost 2 years)
  - Power8+/Pascal GPUs
  - MPI, CUDA, OpenMP 4, target region, etc.
- TotalView being brought up on CORAL Sierra: butte, sierra, lassen, rzansel
  - Power9/Volta GPUs
  - MPI, CUDA, OpenMP 4, target region, etc.
- But there have been some problems… Which we'll talk about in a bit

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView Startup on CORAL EA

- Rzmanta, ray, and shark
- https://lc.llnl.gov/confluence/display/CORALEA/Debuggers#Debuggers-TotalView
- An interactive allocation for your debugging session
  - `$ bsub -x -n 4 -Is -XF -W 60 -G guests /usr/bin/bash`
- TotalView startup on CORAL EA
  - On EA:
    ```
    $ mpirun -debug -np 4 a.out
    $ mpirun –tv -np 4 a.out
    ```
  - See EA page:
    ```
    $ totalview –args orterun -np 4 a.out
    ```


© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView Startup on CORAL SIERRA

- Sierra, lassen, rzansel, butte
  - Use: **/usr/global/tools/totalview/v/totalview.2018X.3.5/bin/totalview**

- https://lc.llnl.gov/confluence/display/SIERRA/Debuggers

- An interactive allocation for your debugging session
  - `$ bsub -nnodes 4 -Is -XF -W 60 -G guests /usr/bin/bash`
  - `Lalloc 4 60 –q pdebug`

- TotalView startup on CORAL EA
  - On SIERRA: `$ totalview -args jsrun -p4 -r1 a.out`
    `$ totalview -args lrun -p4 -T1 a.out`
  - `-T#  - number of tasks per node – can be used to spread out the job`

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView/CORAL Compiler Support

- CPU code: all compilers fully supported
- GPU code: NVCC fully supported
- OpenMP target regions GPU code: IBM XL and CLANG are “pretty good”
  - If debugging OpenMP target regions seems “broken” please report it
  - The compiler and/or TotalView can be fixed, maybe

- Tool-Friendly compiling/linking options (see the Wiki)
  - NVCC:
    - **-O0 -g -G -arch sm_60** (to generate GPU DWARF and avoid JIT)
    - **-dlink** (to link GPU object files into a large image)
  - XL:
    - 1st try: **-O0 -g -qsmp=omp:noopt -qfullpath –qoffload**
    - 2nd try: **-qnoinline -Xptxas -O0 -Xllvm2ptx -nvvm-compile-options=-opt=0**

No inline helps w/ heavily templated code, e.g., RAJA

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Debug Fission – Split Dwarf Support

Debug Information takes up a lot of space

- Line and symbol information generally represented in DWARF format
  - Allows us to show the source code and locate variables
  - The larger and more complex the code, the more data is needed to represent it. This can grow to GB's in size
- DebugFission SplitDwarf, gdb_index, dwz methods of dealing with this are now all supported.

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView / MRNet Super Bushy

- Due to limitations in the NVIDIA CUDA debug API…
  - **TV::mrnet_super_bushy** must be set to true in TotalView
  - It's the default at LLNL

- Super bushy arranges for 1 debugger server process *per* MPI process
  - Normally, TotalView uses 1 debugger server process *per* node

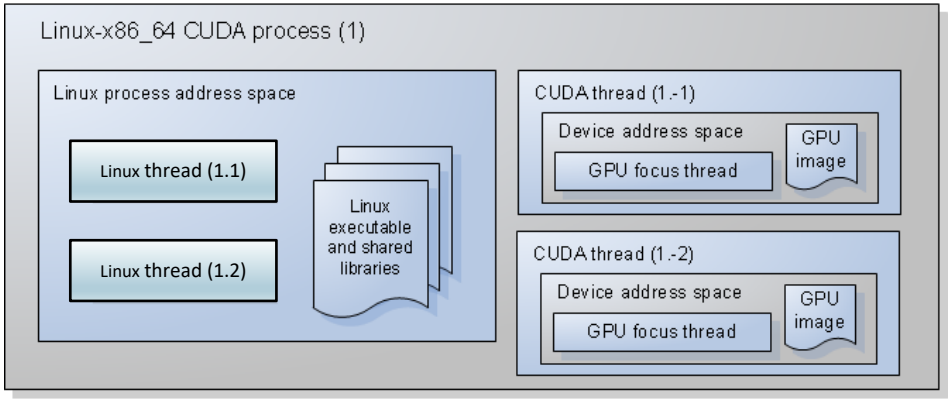
© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView fork/exec Handling

- **jsrun** and **lrun** exec other programs *before* launching the MPI processes
- lrun calls jsrun and jsrun will call mpibind to distribute the GPU's among the processes
- TotalView was modified to make “stop on exec” configurable
  - **TV::exec_handling {{***regexp action***} …}**
  - Action: **go**, **halt** or **ask**
- TotalView was also modified to make “follow fork” configurable
  - **TV::fork_handling {{***regexp action***} …}**
  - Action: **attach** or **detach**
- TotalView has been configured to “**go**” on **jsrun** and **lrun** exec calls
  - Make the exec calls transparent on LLNL systems, but…
  - MPI processes are named: “lrun<bash><jsrun><jsrun><mpibind>.0”
- The actual name of the MPI executable is *not* part of the process name
- This affects the saving of breakpoints since TotalView saves the breakpoint file as 'program.tvd.v4breakpoints' in the same directory as the executable.

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView CUDA Debugging Model
The original take



A CUDA process is-a “bag” consisting of:

1. **Linux process address space**
   - Containing a Linux executable and a list of Linux shared libraries
2. **Collection of Linux threads**, where a Linux thread:
   - Is assigned a positive debugger thread ID: 1.1, 1.2, etc…
   - *Shares* the Linux process address space with other Linux threads
3. **Collection of CUDA threads** (contexts), where a CUDA thread/context:
   - Is assigned a negative debugger thread ID: 1.-1, 1.-2, etc…
   - Has an address space *separate* from the Linux process *and* other CUDA threads
   - Has a "GPU focus thread", which is focused on a specific hardware lane

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Simple Debug Session Walkthrough

butte5{jdelsign}204: **bsub -nnodes 4 -W 60 -Is -XF -G guests /bin/tcsh**

Job <3749> is submitted to default queue <pbatch>.

<<ssh X11 forwarding job>>

<<Waiting for dispatch ...>>

<<Starting on butte5>>

butte5{jdelsign}21: **/usr/global/tools/totalview/v/totalview.2018X.3.5/bin/totalview **

-args lrun -N4 -T1 -p4 ./mpihasgpu_debug

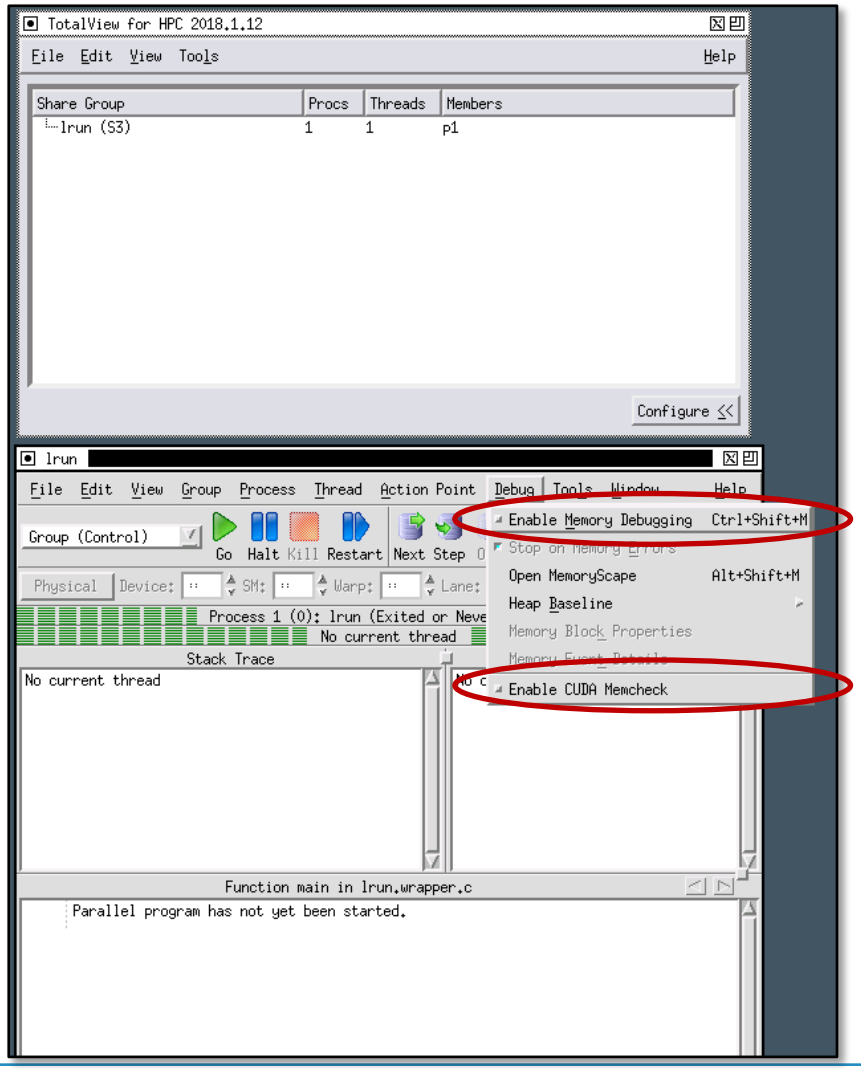
- This program prints something like:

```
Rank    3 Host butte17        Able to use GPU 0 1  CPUs 0
Rank    2 Host butte16        Able to use GPU 0 1  CPUs 0
Rank    0 Host butte13        Able to use GPU 0 1  CPUs 0
Rank    1 Host butte14        Able to use GPU 0 1  CPUs 0
```

mpibind sets CUDA_VISIBLE_DEVICES

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TV Starts on “lrun”



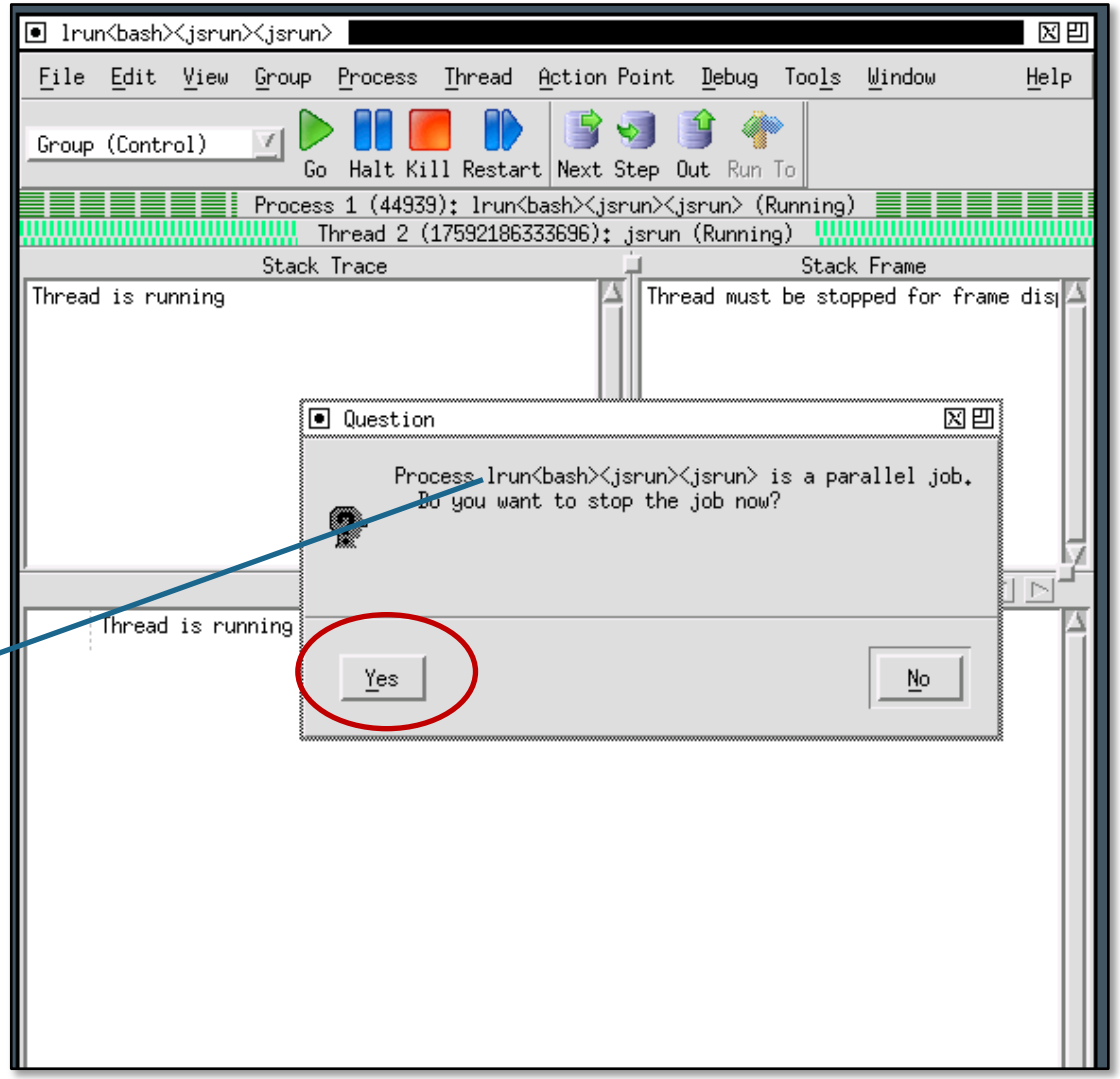
- Before starting lrun, you can
  - Enable CUDA_Memcheck (GPU) – recommended for odd GPU failures
  - Enable TotalView Memory Debugging (CPU)

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Start “lrun”; stop the MPI processes

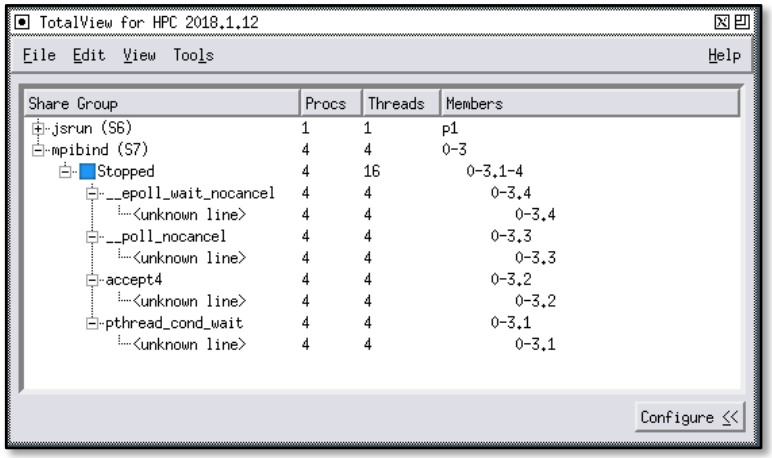
Why the funny name?

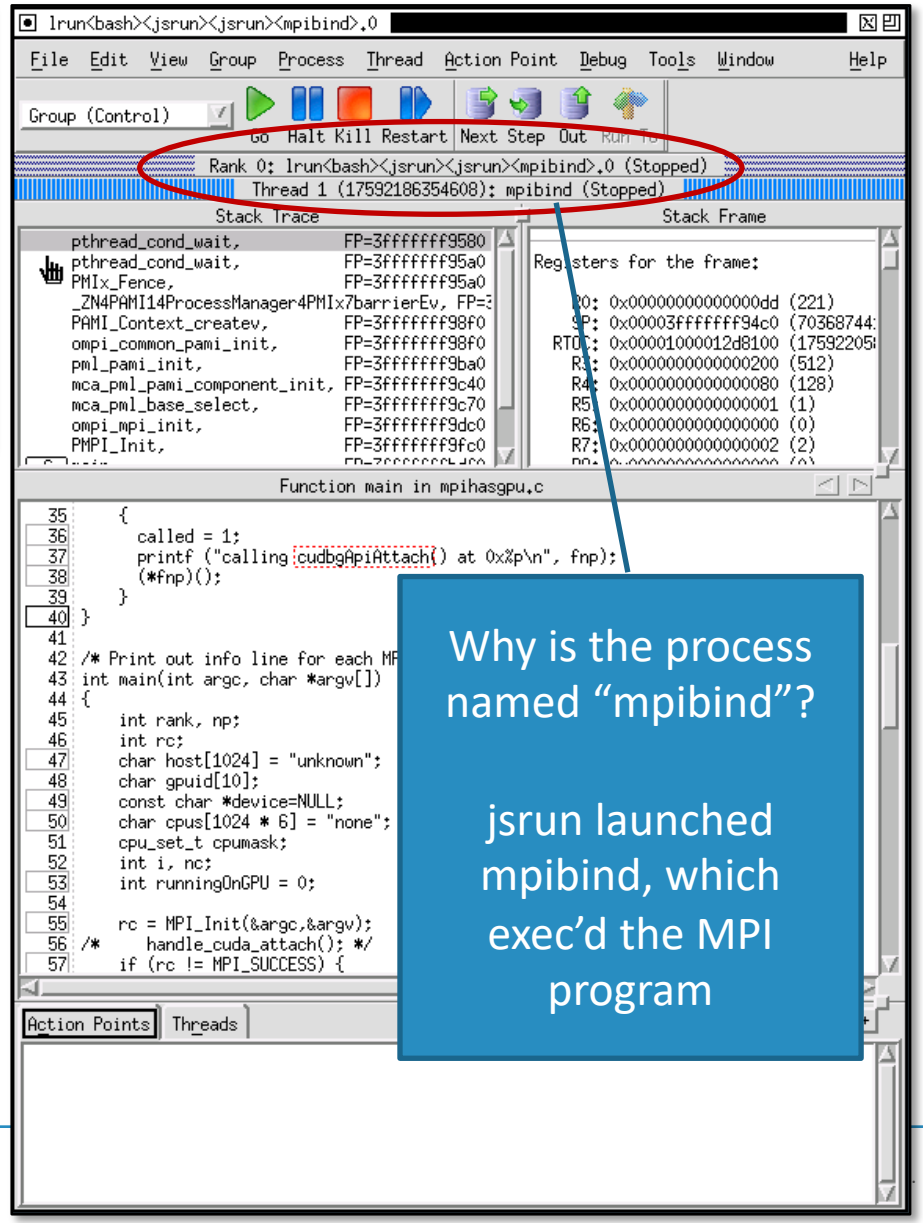
lrun exec’d bash, which exec’d jsrun, which exec’d a different jsrun



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

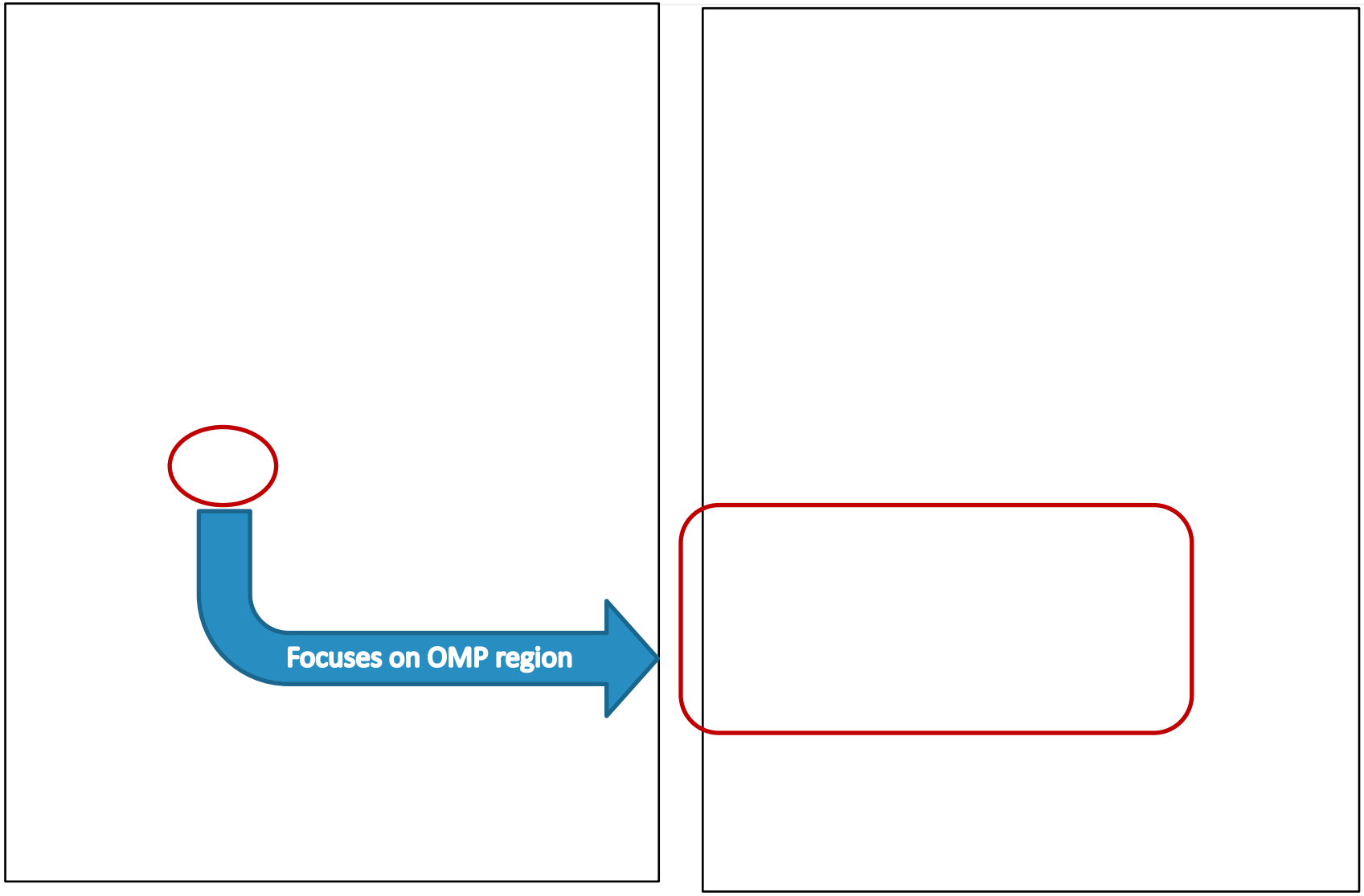
Stopped, focused on rank 0





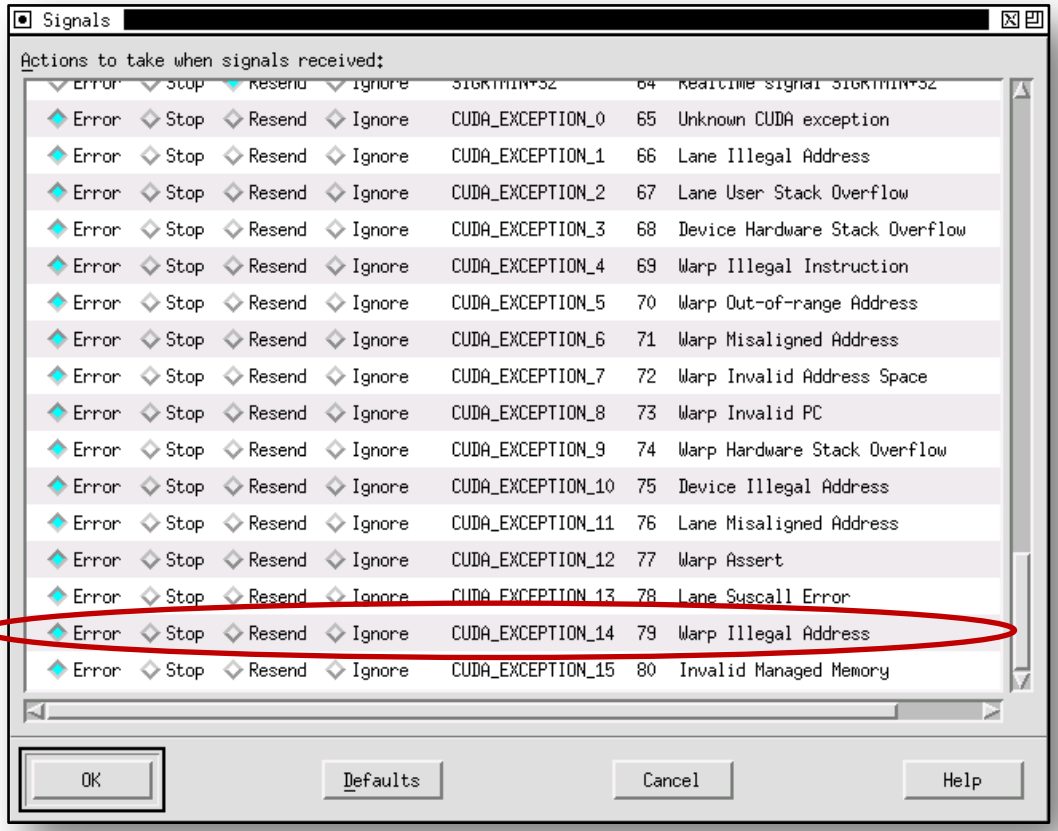
- So far, no different than normal MPI debugging
- Breakpoint host code, run, step, etc…

First Kernel launch …



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

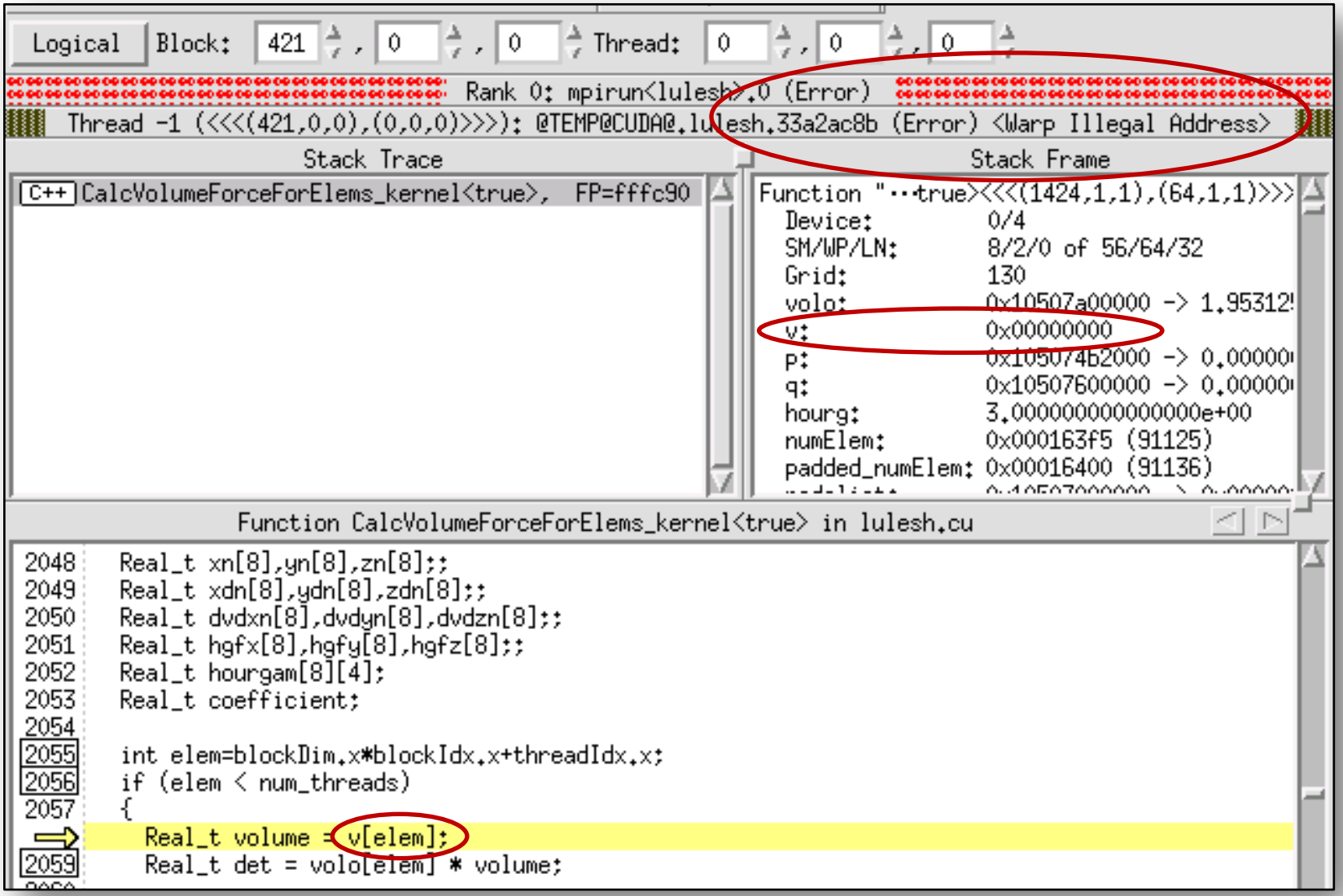
CUDA Exception Handling



- File > Signals…
- Scroll to the bottom to see the CUDA exception
- http://docs.nvidia.com/cuda/cudagdb/#gpu-error-reporting

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

CUDA exceptions *are* error signals



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Currently Known Issues

- Saving breakpoints from a debug session!
  - Unless you have access to the mpibind location you can't save the breakpoint! (or you are not using mpibind!)
  - Even using 'Action Points -> Save as' doesn't help
- Next and step can be slow on the GPU depending on the complexity of the action at the line - Simple assignment is okay, but calling functions, or where expressions manipulating variable are involved not so much.
- Large slowdown seen in DebugAPI calls when debugging on multiple nodes.
  - When running on a single node response time in milliseconds
  - When running between nodes however, 10, 20, 30 second response times
  - Nvidia and Rogue Wave are investigating

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView Agenda

- TotalView Debugging on Sierra CORAL with GPUs and OpenMP
- CUDA Unified Breakpoints Project
- TotalView New UI
- OpenMP Debugging (OMPD) Standardization Effort
- TotalView Roadmap
- TotalView Technical Support

Peter Thompson

TotalView Specialist

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

CUDA Unified Breakpoints (CUB) Project

- Project partially funded by a LLNL SOW milestone

- Basically, modifies TotalView to “unify” CPU and GPU breakpoints
  - Breakpoints are unified across CPU processes and GPU contexts
  - Source file and line number symbols unified across all image files
    - CPU executable
    - CPU shared libraries
    - GPU ELF images

- Work flow change: Set GPU breakpoints before the kernel is launched
- Current default on CORAL systems

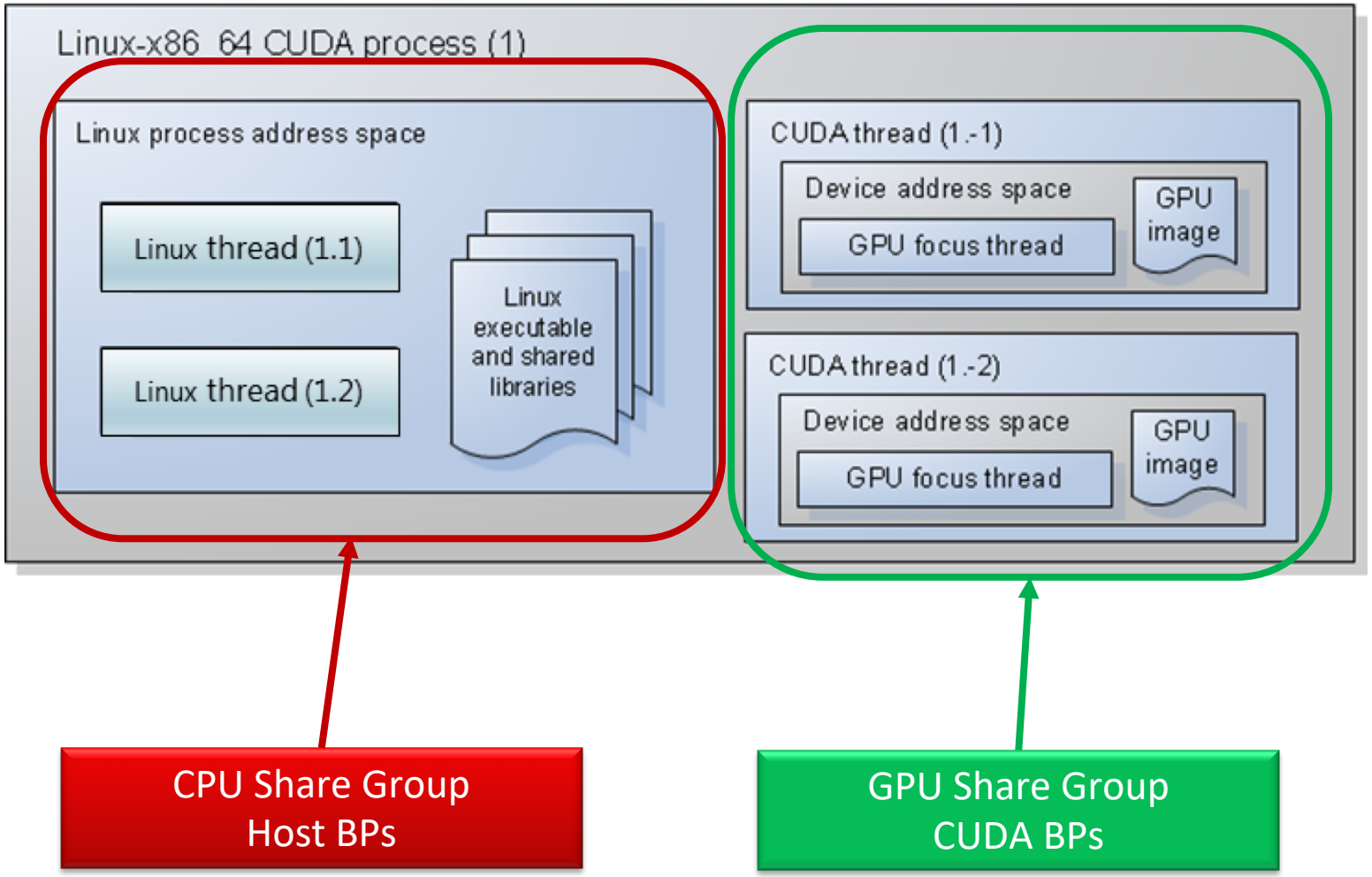
© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView's old CUDA BP model

- The CPU and GPU address spaces are kept in separate **share groups**
- TotalView creates separate share groups
  - One for each unique CPU executable
  - One for each unique GPU application
- Breakpoints are planted in the share group
  - Breakpoints do not cross share group boundaries
  - Breakpoint cannot be created until the share group exists
- Share groups for the GPU are not created until after kernel launch, thus…
- It is not possible to plant breakpoints in the GPU before we see the first CUDA context created using a particular GPU application

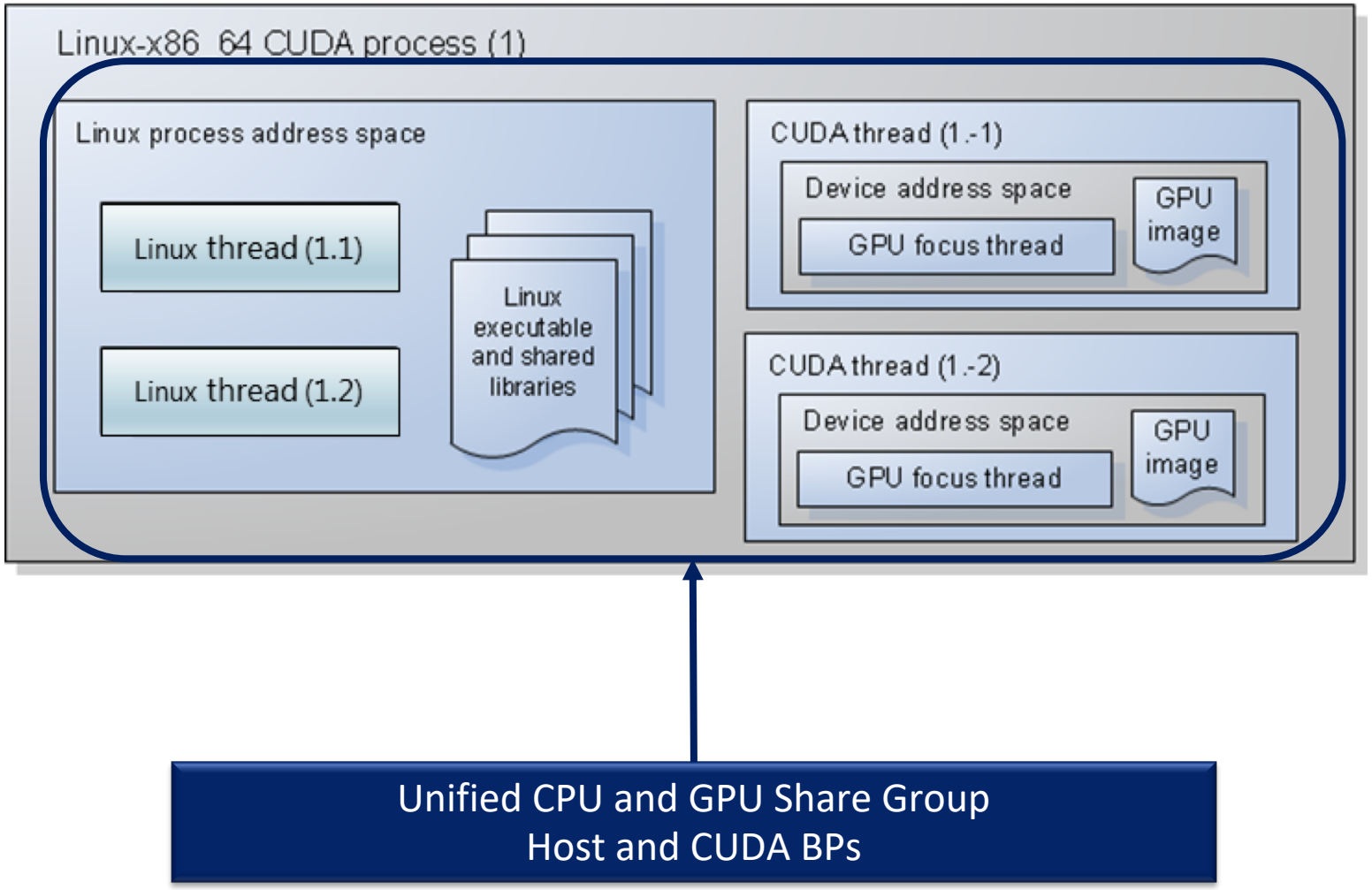
© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView's old shared group model



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

A new unified shared group model



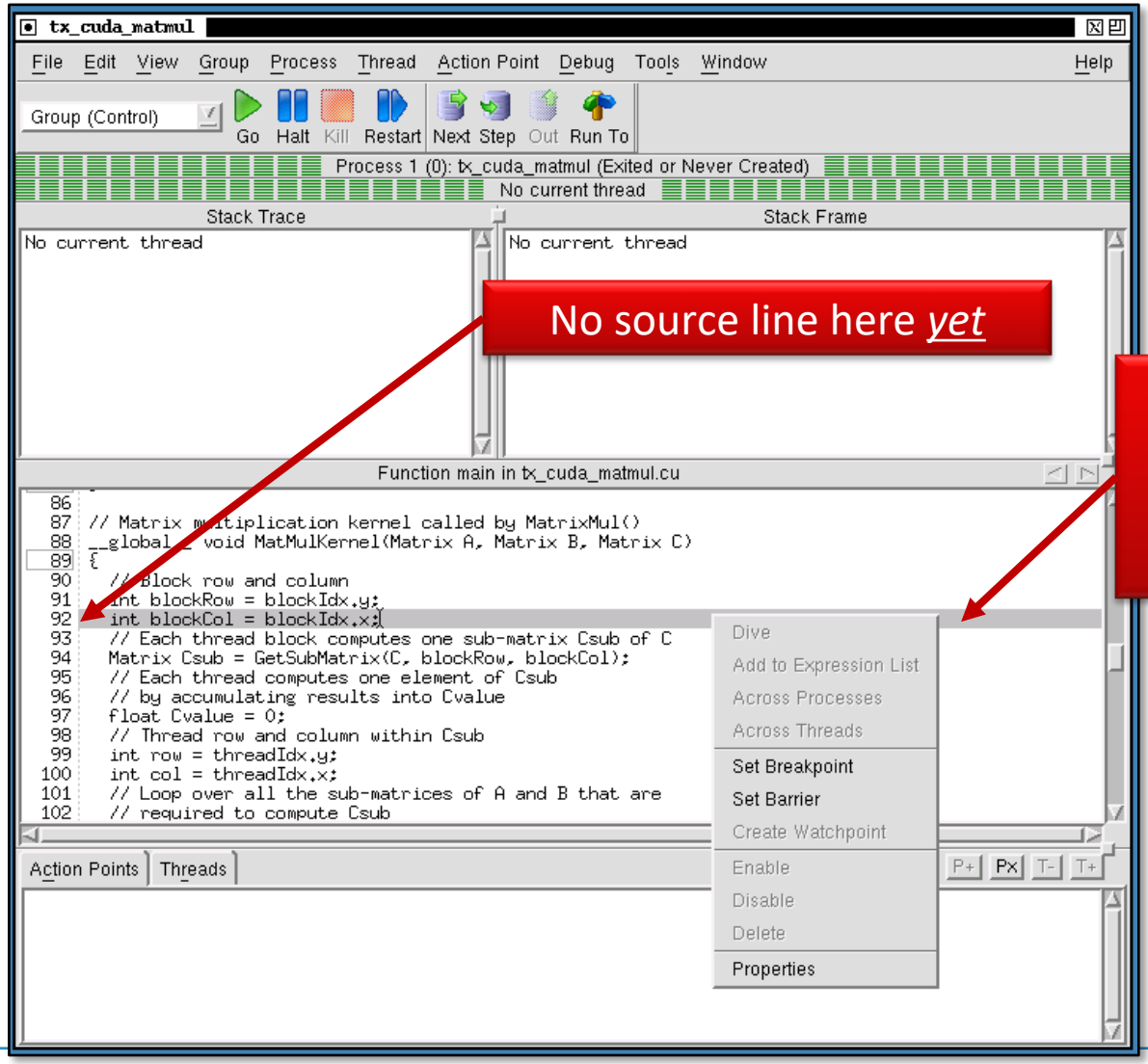
© 2018 Rogue Wave Software, Inc. All Rights Reserved.

CUB: Unified shared groups means…

- Breakpoints are shared between CPU and GPU code
- You can (must) set breakpoints in the GPU before a kernel is launched
  - Similar to what CUDA-GDB allows
- You can set “pending” breakpoints on lines and functions
- What does “pending” mean?
  - You can set a breakpoint on code that does not exist *yet*
  - But you *know* will be loaded at runtime
  - Which happens with CUDA and dynamically loaded libraries

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

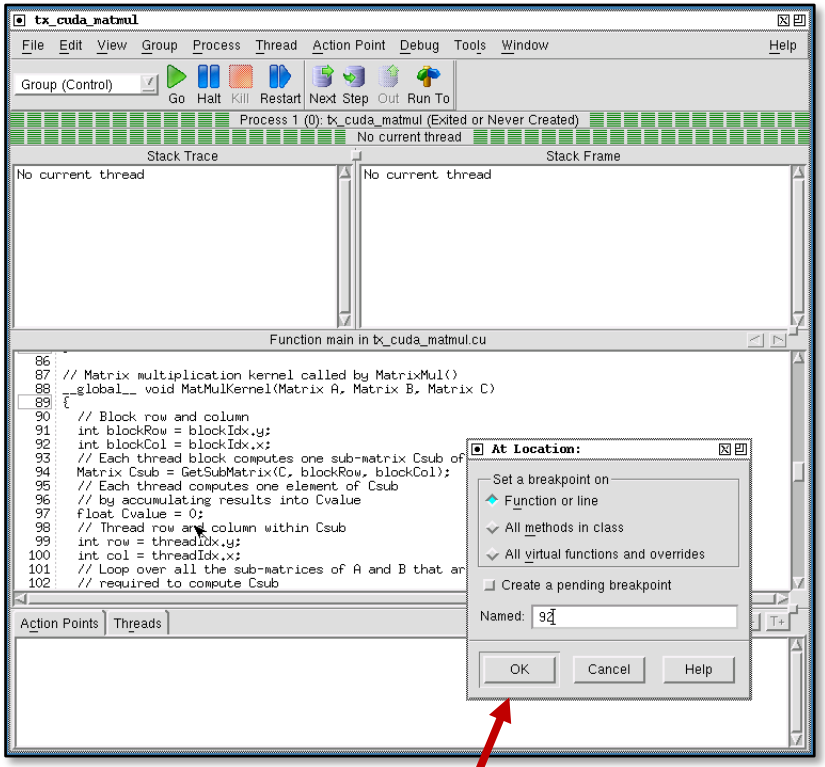
CUB: Pending line number breakpoints



Right-click to set the breakpoint, which will be planted *when* the kernel is launched

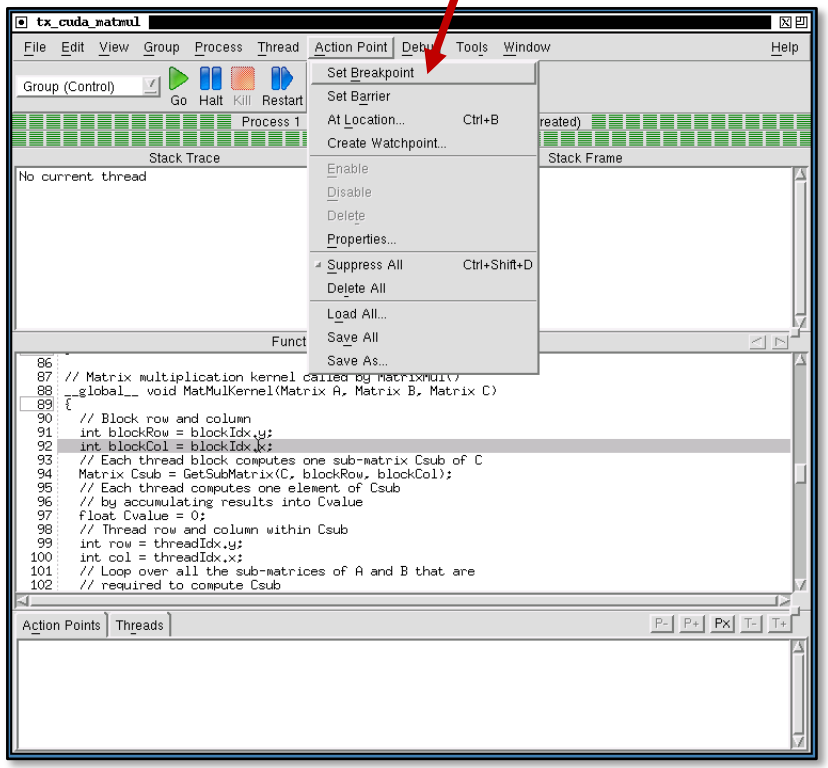
© 2018 Rogue Wave Software, Inc. All Rights Reserved.

CUB: Pending line number breakpoints



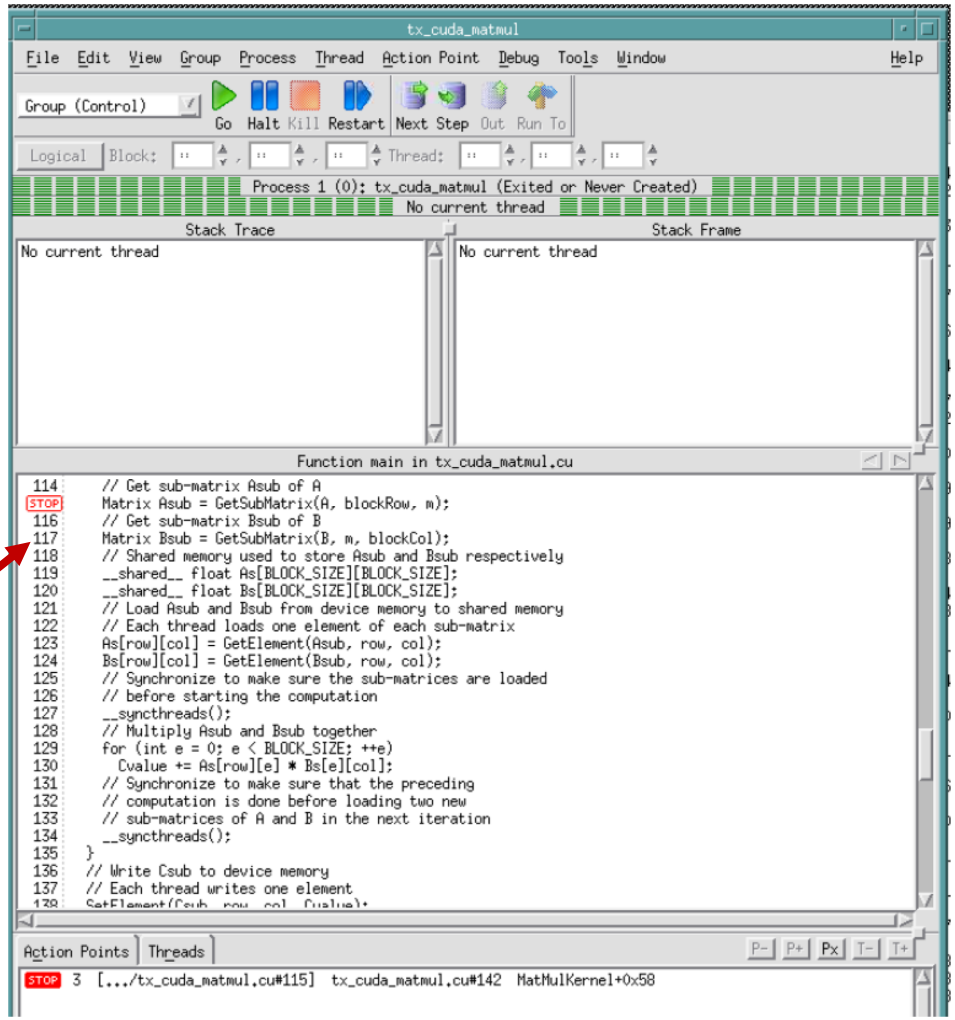
Or use
Action Point > At Location
(Ctrl+B) > line number

Or select the line and use
Action Point > Set Breakpoint



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

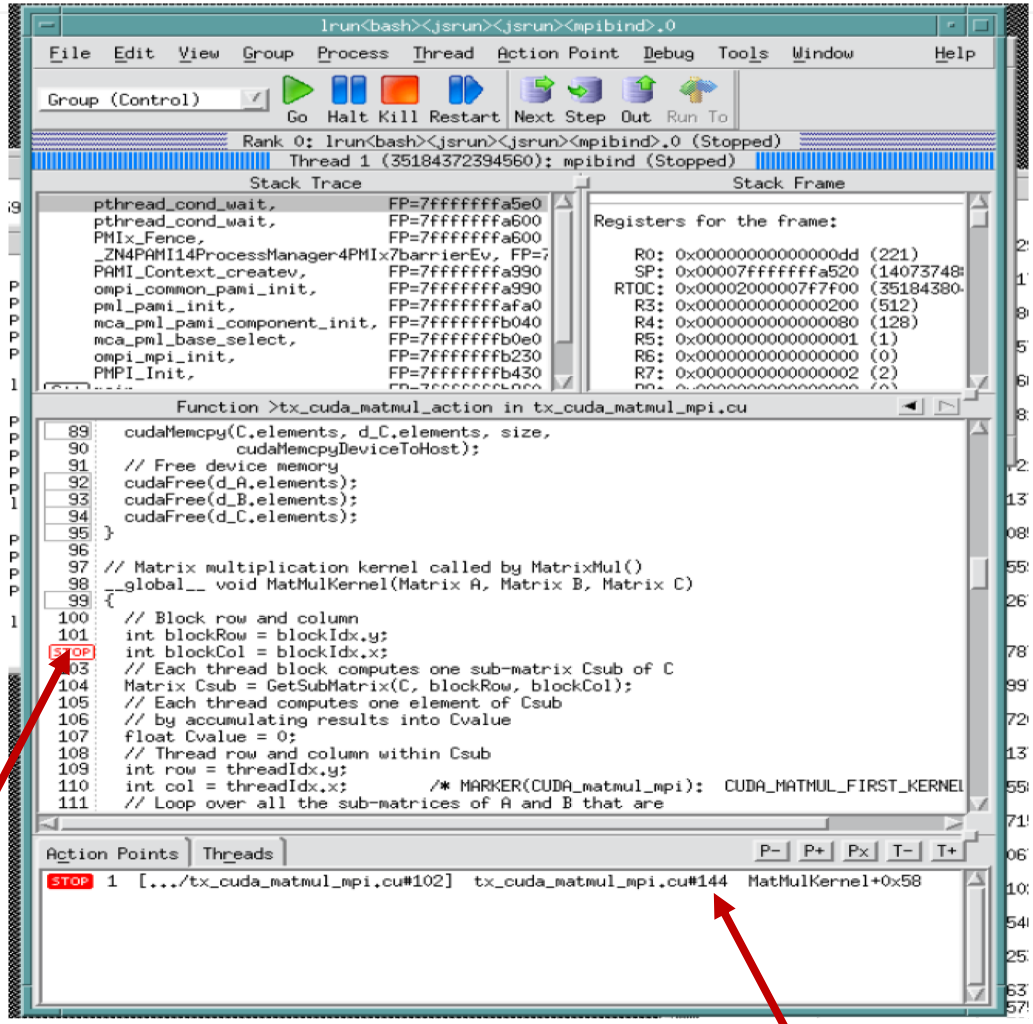
CUB: Or, just click on the line number



The lack of a breakpoint box does not stop one from setting a pending breakpoint

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

CUB: Pending breakpoints “slide” too

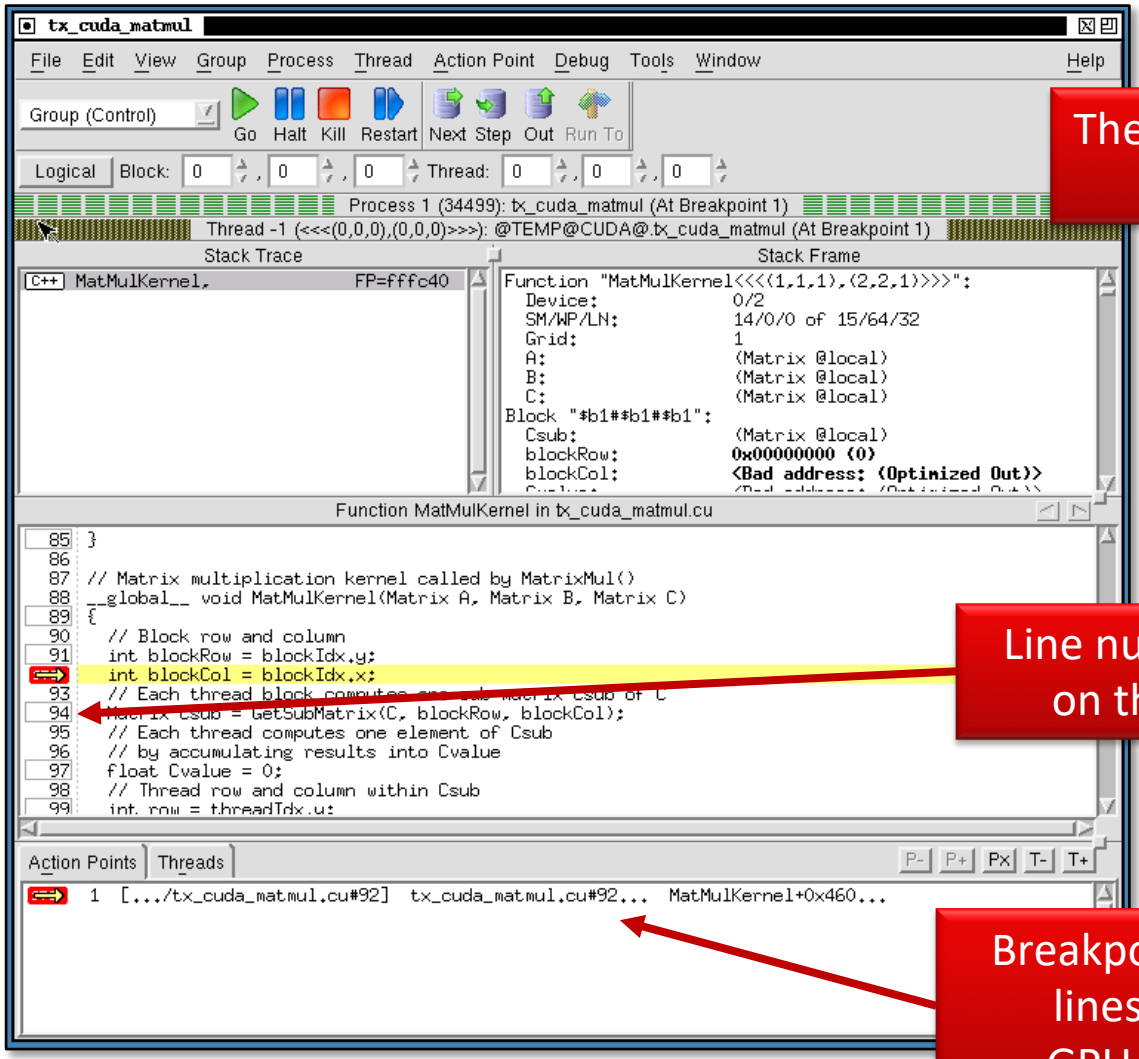


Created at #102

Slides to line #144 on the CPU

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

CUB: Planted at #92 on kernel launch



The “Stop on kernel launch” is disabled by default

Line number gridgets appear on the GPU source lines

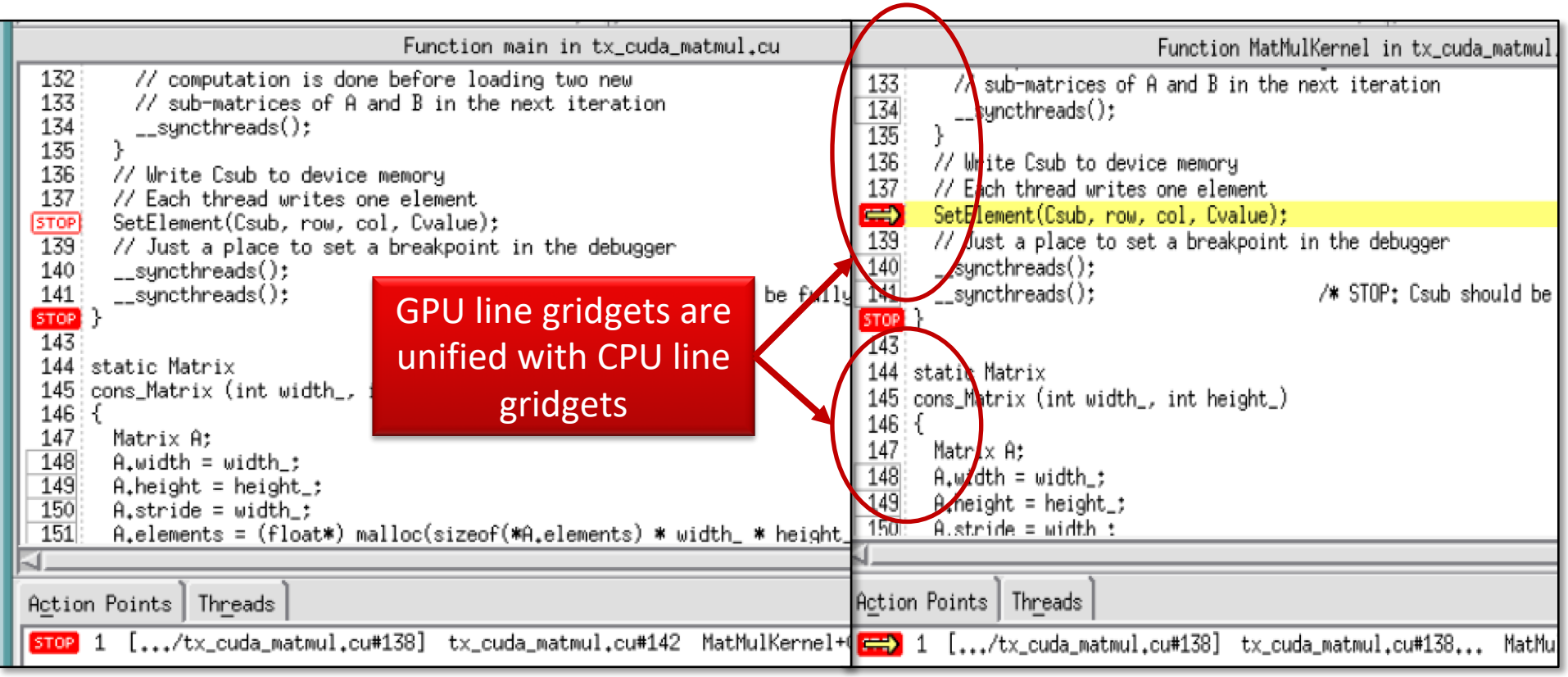
Breakpoint now maps to lines and blocks in GPU *and* CPU code

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

CUB: Source line gridgets are unified too

Before kernel launch

After kernel launch



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView Agenda

- TotalView Debugging on Sierra CORAL with GPUs and OpenMP
- CUDA Unified Breakpoints Project
- TotalView NewGui
- Dskip Support
- OpenMP Debugging (OMPD) Standardization Effort
- TotalView Roadmap
- TotalView Technical Support

Peter Thompson

Principle Tech Support Engineer, TotalView

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Invoking the New Interface

- totalview –newui
- codedynamics - This requires a separate license
- File->Preferences -> To set preferred launch window

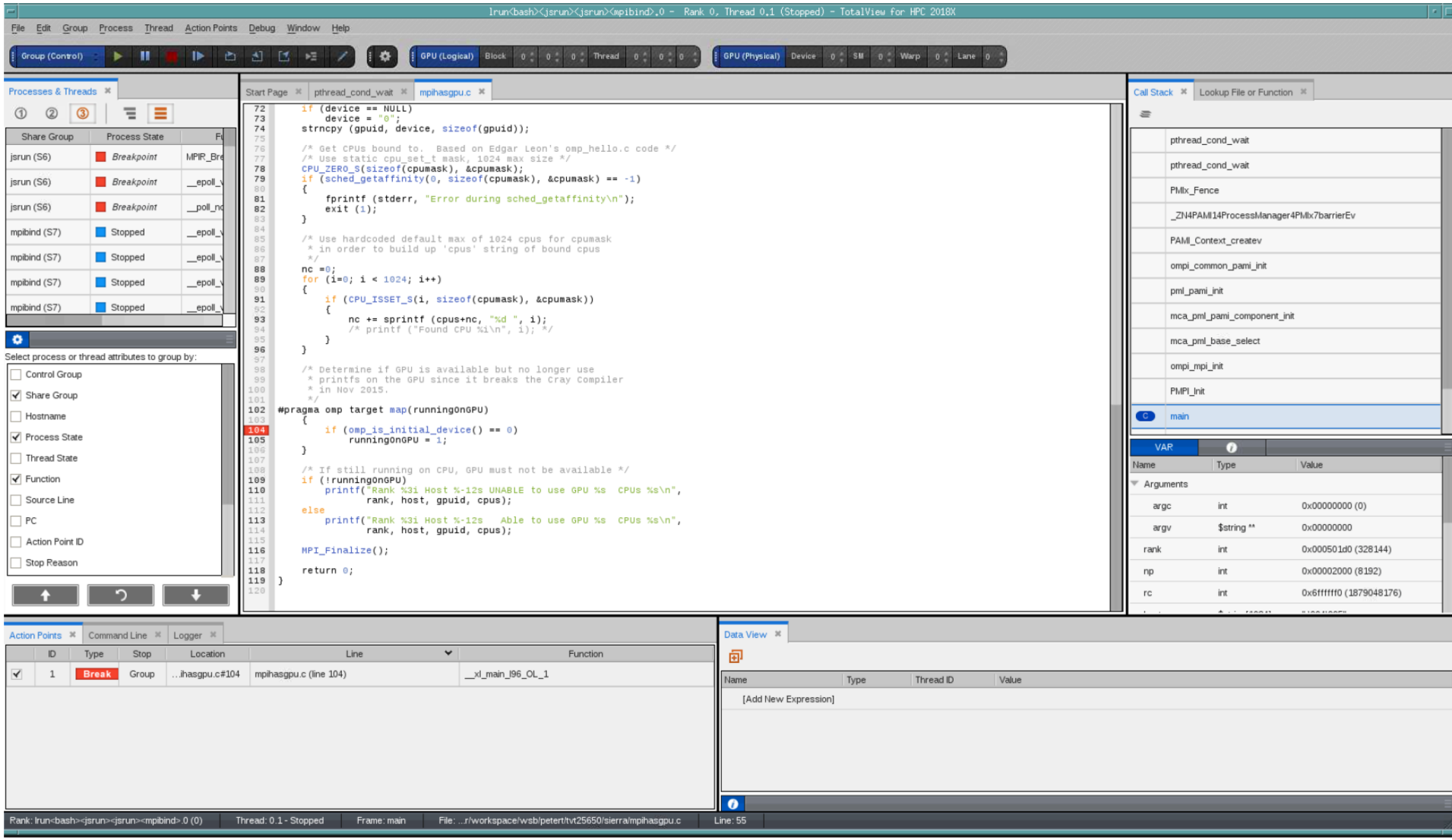
© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Changes to the New Interface
Since the last time you saw it

- Better support for HPC and CUDA debugging
- CUDA Unified breakpoints
- GPU tool bars added to the display with spin boxes
- Choice of Logical/Physical or Both

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

It’s just about ready for prime time!



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Python Development Trends

- Increased usage of Python to build applications that call out to C++
- Provides access to
  - High-performance routines
  - Leverage existing algorithms and libraries
  - Utilize advanced multi-threaded capabilities
  - Computational steering
- Calling between languages easily enabled using technologies such as SWIG, ctypes, Cython, CFFI, et al
- Debugging mixed language applications is not easy

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Python Debugging

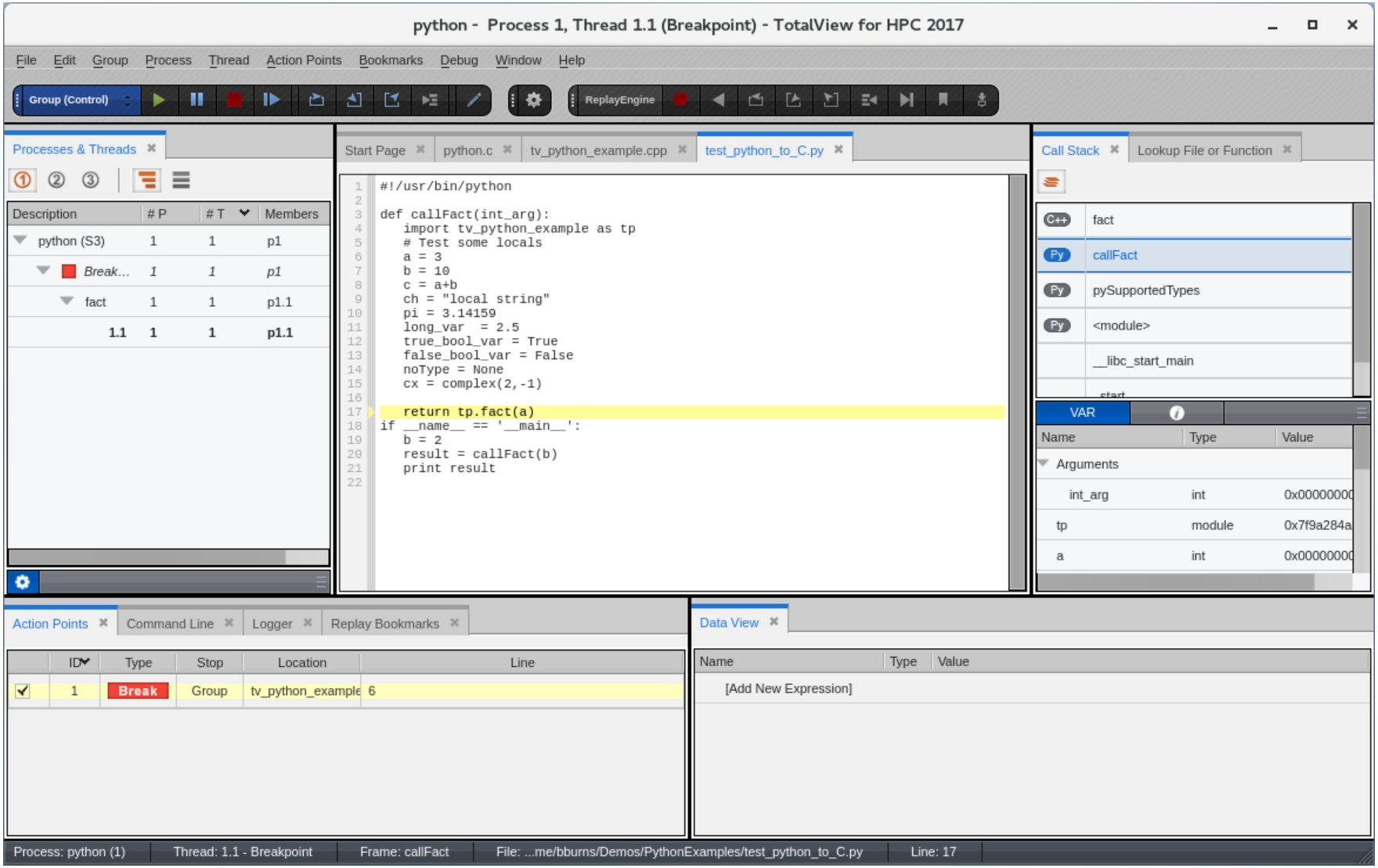
- Debugging one language is difficult enough
- Understanding the flow of execution across language barriers is hard
- Examining and comparing data in both languages is challenging

- What TotalView provides:
  - Easy python debugging session setup
  - Fully integrated Python and C/C++ call stack
    - ”Glue” layers between the languages removed
  - Easily examine and compare variables in Python and C++
  - Modest system requirements
  - Utilize reverse debugging and memory debugging
  - Python 2.7 (Python 3 coming soon)

- What TotalView does not provide (yet):
  - Setting breakpoints and stepping within Python code

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Python Debugging Demo



© 2018 Rogue Wave Software, Inc. All Rights Reserved.

dskip support

- Response to a user request (we do them for people other than the lab too!) looking for gdb like ”skip’ facility to skip through or over libraries that are not interesting to the user.
- Implemented as a CLI command to set up rules for single stepping skipping
- Use regular expressions to specify functions to skip and files can be matched with a file name or glob pattern
  - dskip through function Callback
  - dskip –rfunction [^EnumS<(.*)>::EnumS\(\1\)]
  - dskip file /g/g22/petert/devel/testfuncs.cxx
  - dskip over –gfile *.cxx

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

Skipping Over or Through

- Skipping over is the default unless qualified with the ‘through’ option
- Over
  - EnumS<Color> v (Red) ;
    - A step/dstep here would avoid stepping into the constructor for Enums
- Through
  - Callback (MyFunc)
    - A step/dstep at this point would step into Myfunc, passing through the code in Callback
- Full documentation can be found here

https://docs.roguewave.com/totalview/current/html/#page/Reference_Guide%2Fdskip.html

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

OpenMP Debugging (OMPD)

- OMPD allows debugging tools to extract the OMP runtime state of a live process or core file
  - Map from underlying pthread/device threads to OpenMP thread
  - OpenMP threads in a parallel/task region
  - OpenMP thread parent / child relationship
  - OpenMP internal control variables (ICVs)

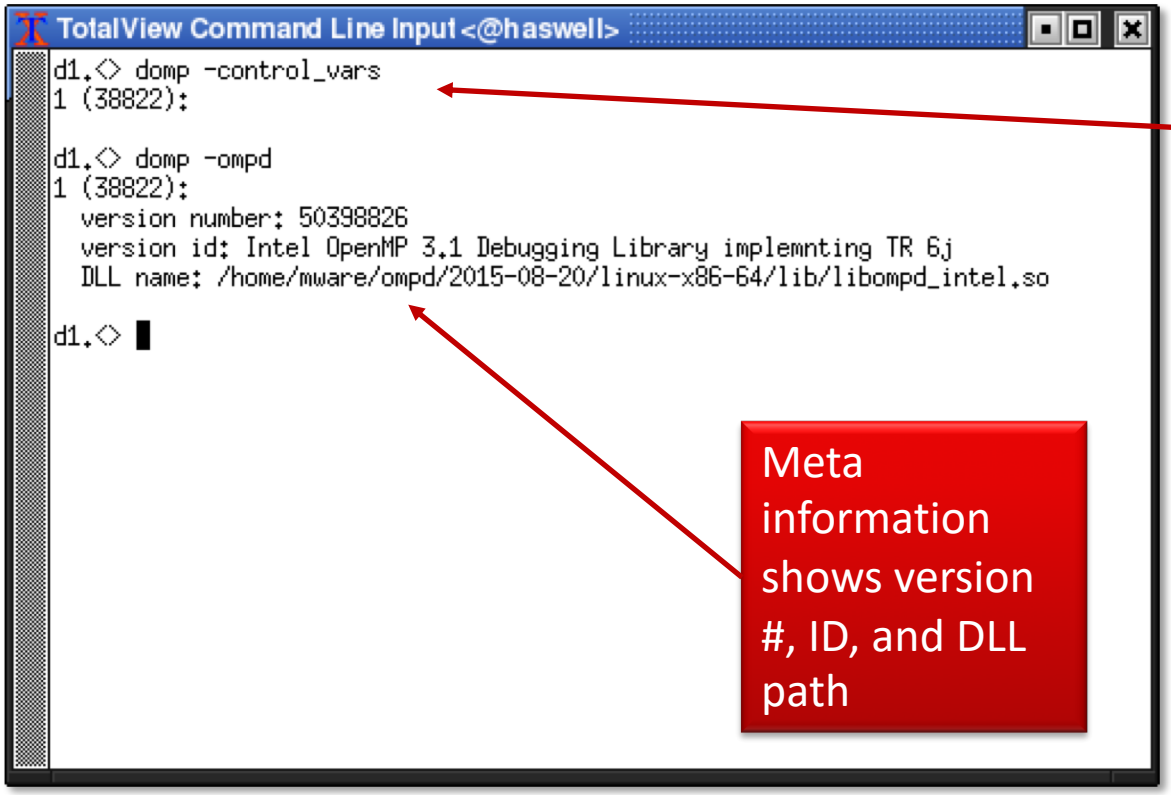
- Primary contributors
  - Joachim Protze, Martin Schulz: RWTH Aachen University
  - Ignacio Laguna, Dong Ahn, Marty McFadden: LLNL
  - John DelSignore: Rogue Wave Software

- 2.5+ year effort
  - OMPD library prototyped in the Intel OMP runtime for host threads
  - TotalView and an LLNL-modified GDB
  - Compiler vendors are just now starting to think about OMPD support

- OMPD is planned for inclusion in the OpenMP 5.0 standard

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

OMP Control Vars & Meta Info

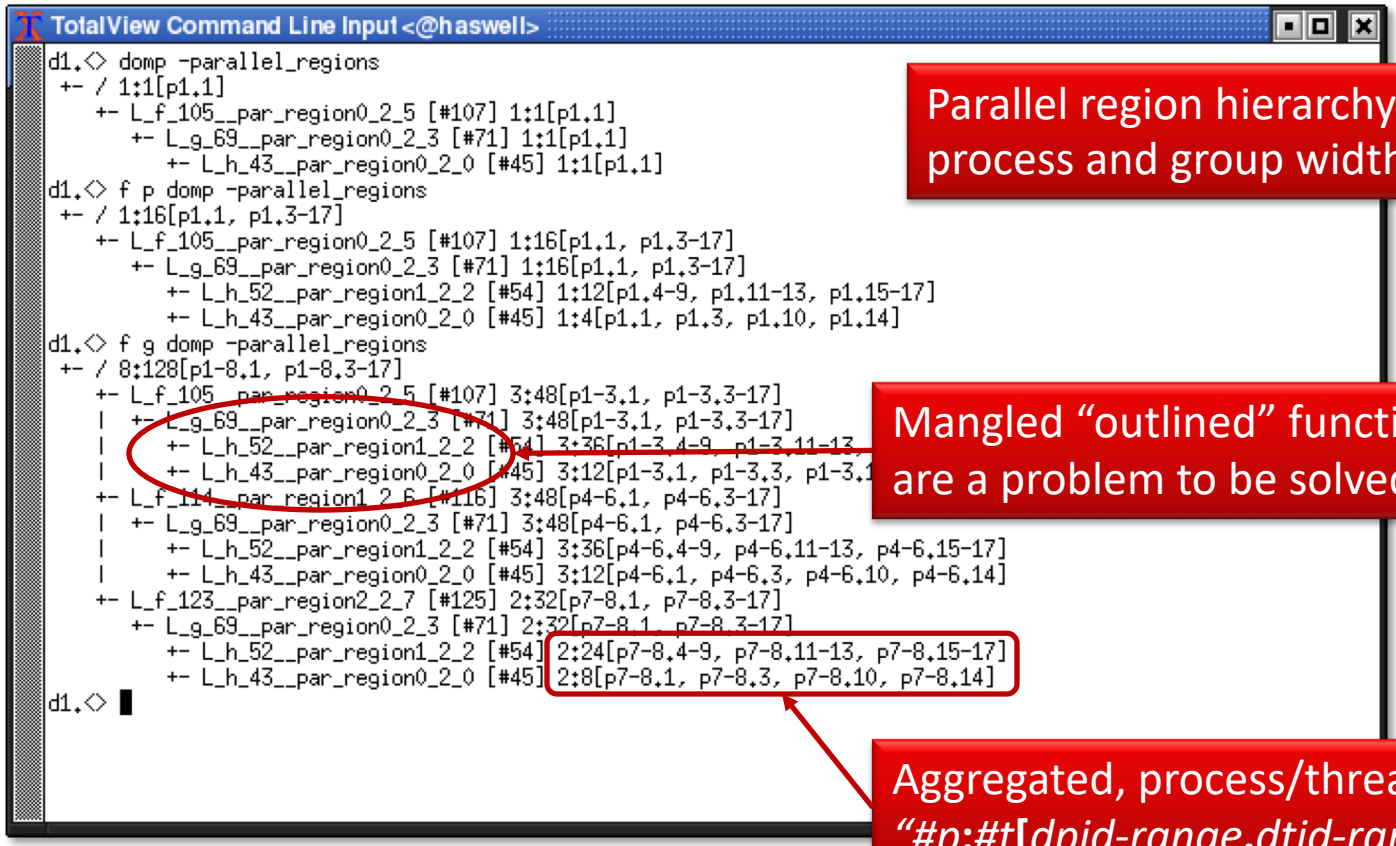


Intel OMPD DLL currently returns no control variable information

Meta information shows version #, ID, and DLL path

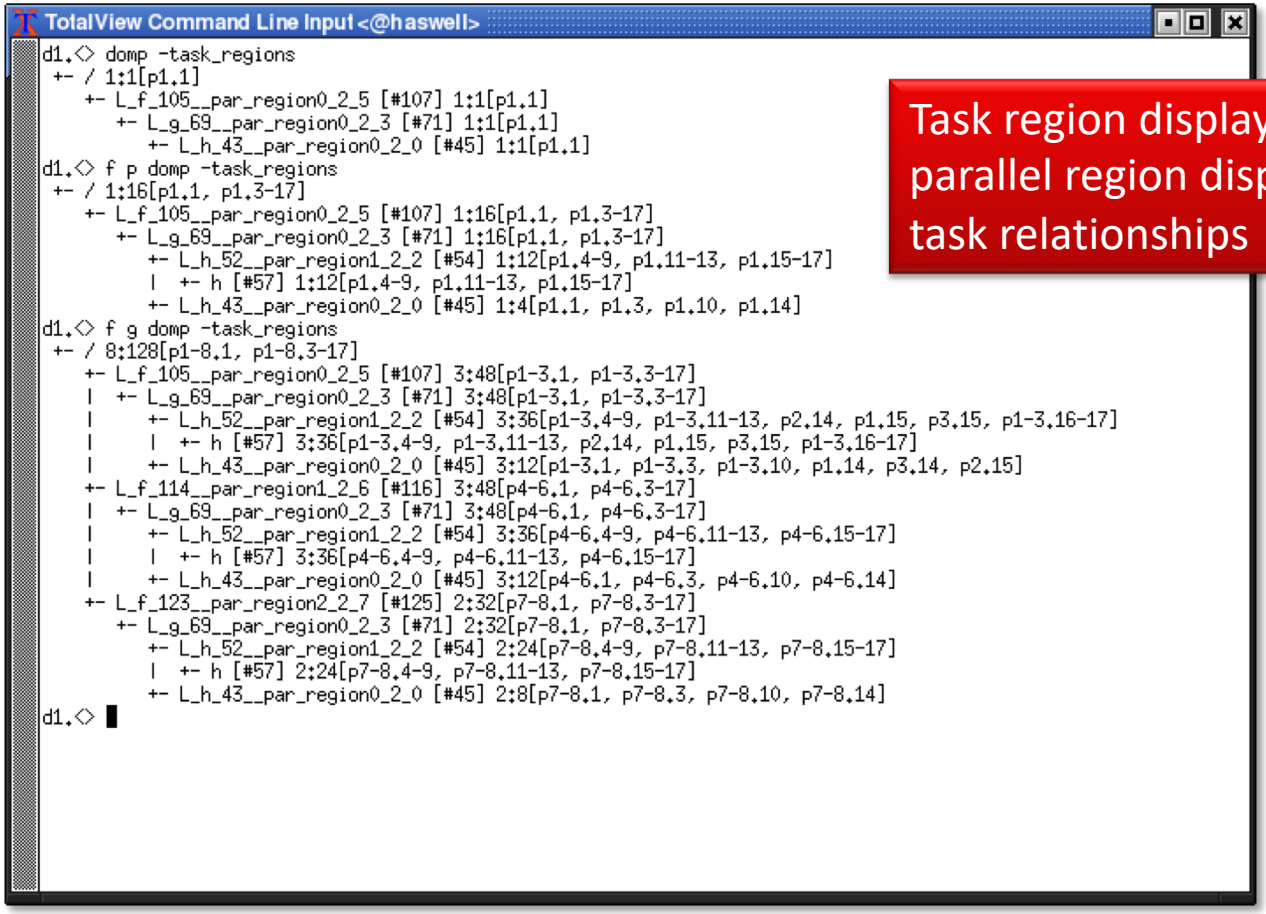
© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Parallel Regions



© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Task Regions



Task region display is similar to parallel region display, but shows the task relationships

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Threads

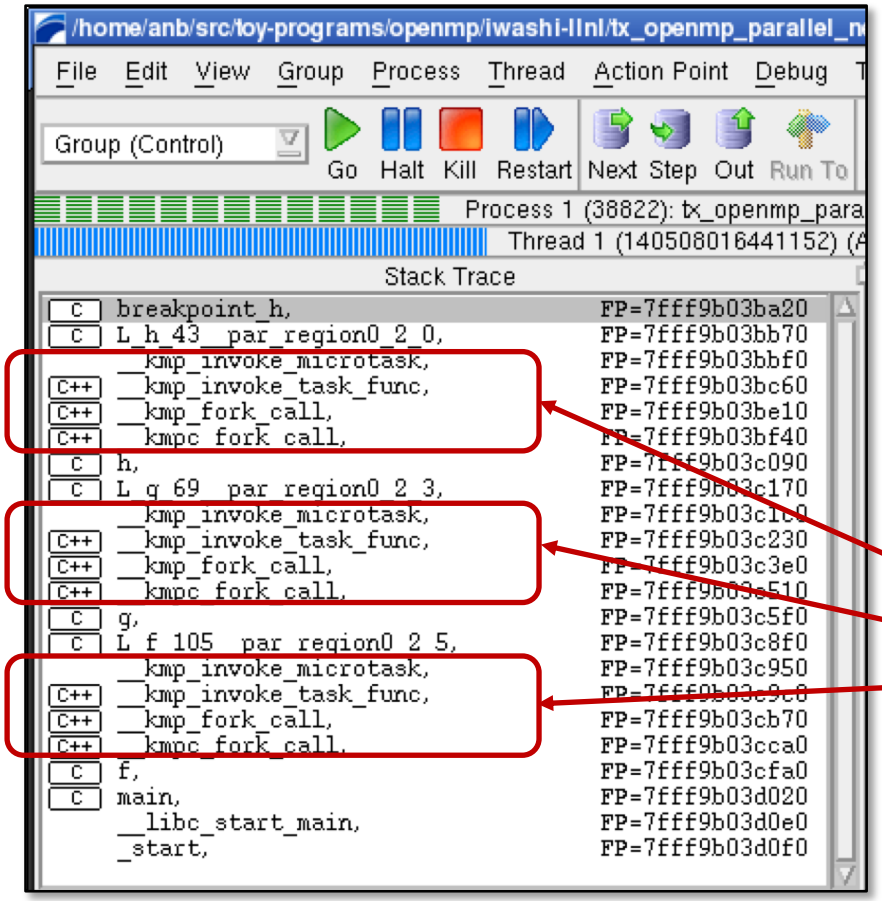
Thread-centric views of information available from OMPD

TotalView Command Line Input <@haswell>

```
d1.<> domp -threads
1 (37523): 16 OpenMP threads
     tv_id          os_thread_id thread_num          state  parallel_id         task_id
   ========================================================================================
         1       140302052603712          0    work_parallel           4          37 ip-

d1.<> domp -threads -regions
1 (37523): 16 OpenMP threads
     tv_id          os_thread_id thread_num          state  parallel_id         task_id
   ========================================================================================
         1       140302052603712          0    work_parallel           4          37 ip-
                                                                      1           2          11 ip-
                                                                      2           1           8 ip-
                                                                      3           0           4 ip-

d1.<> domp -threads -regions -functions
1 (37523): 16 OpenMP threads
     tv_id          os_thread_id thread_num          state  parallel_id         task_id              parallel_function                   task_function
   ================================================================================================================================================
         1       140302052603712          0    work_parallel           4          37 ip-    L_h_43__par_region0_2_0 [#45]    L_h_43__par_region0_2_0 [#45]
                                                                      1           2          11 ip-    L_g_69__par_region0_2_3 [#71]    L_g_69__par_region0_2_3 [#71]
                                                                      2           1           8 ip-  L_f_105__par_region0_2_5 [#107]  L_f_105__par_region0_2_5 [#107]
                                                                      3           0           4 ip-              0x0000000000000000               0x0000000000000000

d1.<> domp -threads -regions -functions -stack
1 (37523): 16 OpenMP threads
     tv_id          os_thread_id thread_num          state  parallel_id         task_id       exit_address     reentry_address              parallel_function                   task_function
   ================================================================================================================================================================================================
         1       140302052603712          0    work_parallel           4          37 ip- 0x00007ffffc58c870 0x0000000000000000    L_h_43__par_region0_2_0 [#45]    L_h_43__par_region0_2_0 [#45]
                                                                      1           2          11 ip- 0x00007ffffc58ce40 0x00007ffffc58cbc0    L_g_69__par_region0_2_3 [#71]    L_g_69__par_region0_2_3 [#71]
                                                                      2           1           8 ip- 0x00007ffffc58d610 0x00007ffffc58d190  L_f_105__par_region0_2_5 [#107]  L_f_105__par_region0_2_5 [#107]
                                                                      3           0           4 ip- 0x0000000000000000 0x00007ffffc58d960              0x0000000000000000               0x0000000000000000

d1.<>
```


© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Stack Filtering

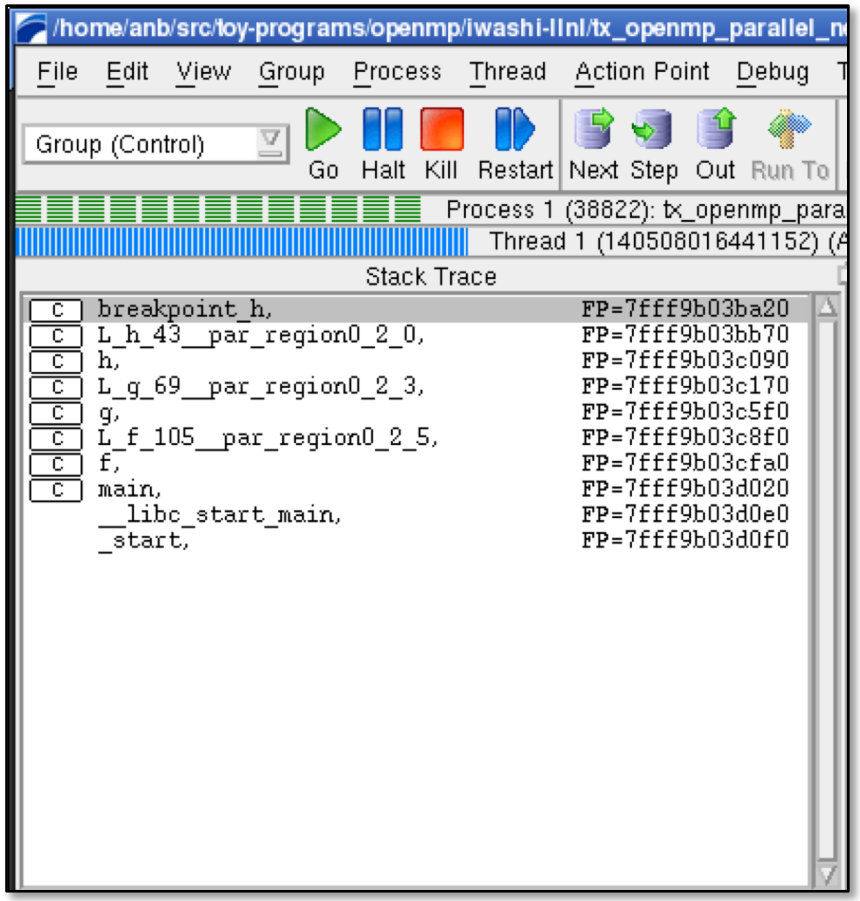


- "Raw", unfiltered stack displays the OMP RTL stack frames

OMP RTL frames typically uninteresting to users

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

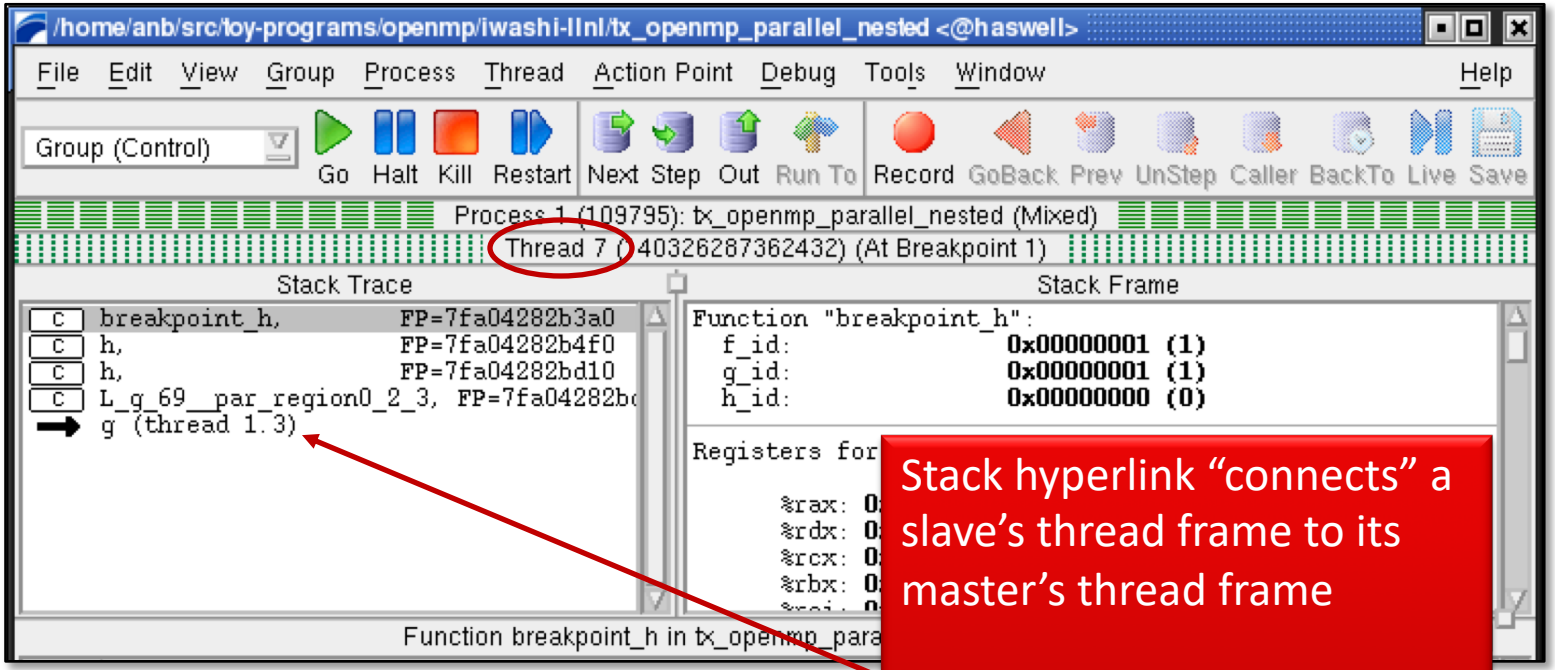
OMP Stack Filtering



- OMPD allows the debugger to *portably* find and filter-out OMP RTL frames

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Master/Slave Stack Linking

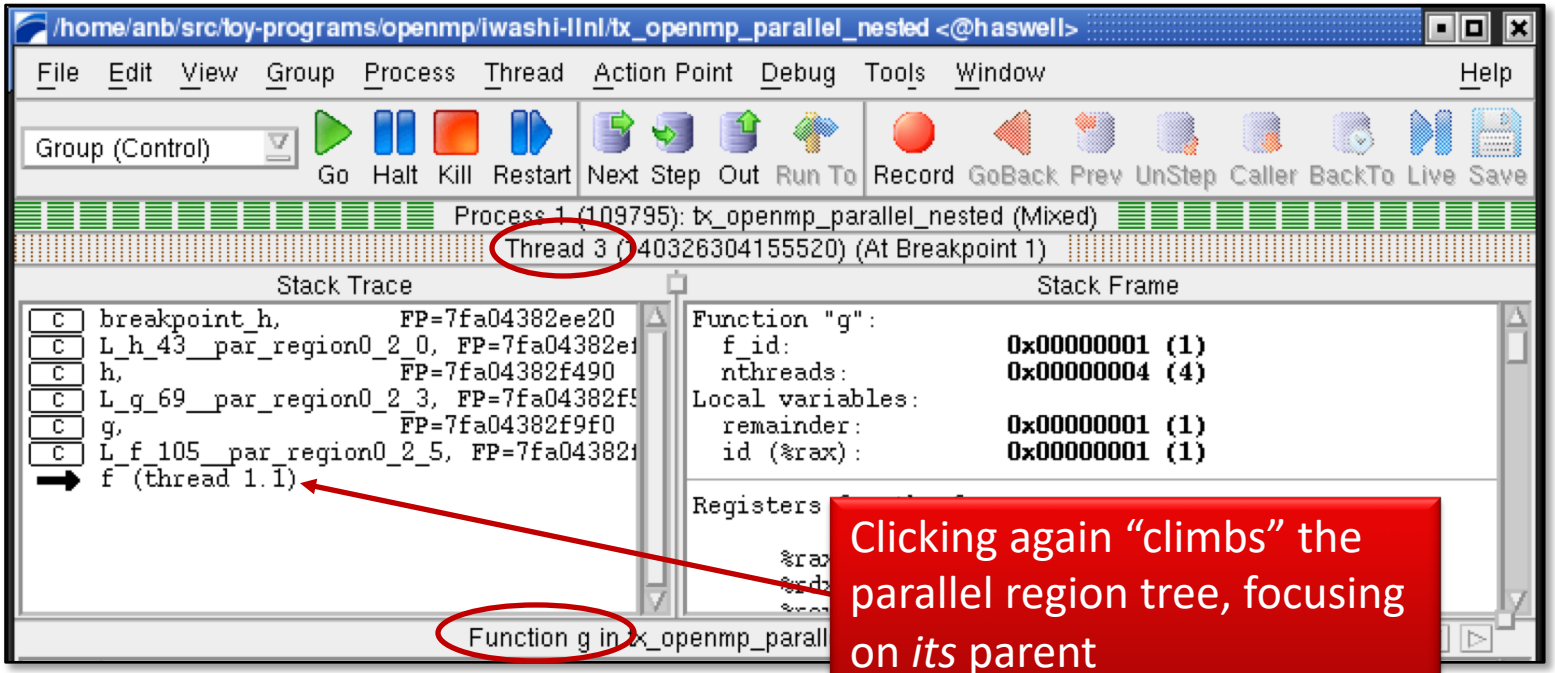


Stack hyperlink “connects” a slave’s thread frame to its master’s thread frame

Selecting the frame jumps to the parent thread and stack frame that invoked the parallel region

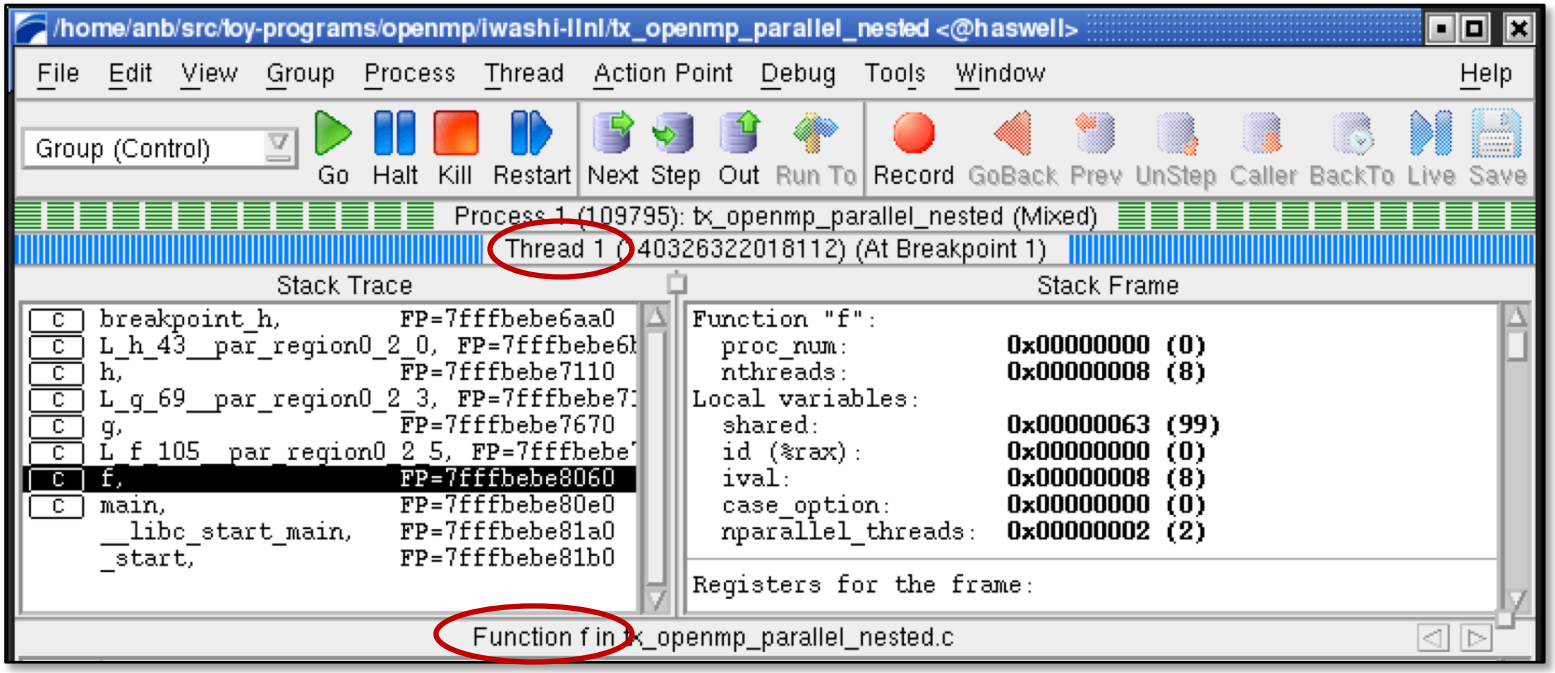
© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Master/Slave Stack Linking



© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Master/Slave Stack Linking



Now at the “root” of the OMP parallel region tree

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Mangled Outlined-Functions

- OMP compiler outlined-function name mangling is *not* standard
- DWARF *could* connect an outlined function to its containing function
  - E.g., DW_AT_omp_outlined <*containing-die*>
- Instead of

  “L_func_42__par_region0_1_2”
- Debugger could *reliably* show something like

  “func (parallel region 1 at file.c:42)”

- What’s needed
  - A DWARF OpenMP proposal
  - A compiler to produce the DWARF

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP Variable Information

- Users have asked for “OpenMP variable information”
  - E.g., private, shared, firstprivate, copyin, reduction, etc.
  - Compile-time attributes of the variable that the compiler knows
  - DWARF *could* represent these attributes
- Can’t the user just look at the source code?
  - Apparently it’s not that simple
  - Users are looking for confirmation
  - Packages like RAJA may hide OMP directives inside templates
- Needed
  - A DWARF OpenMP proposal
  - A compiler to produce the DWARF

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMP + GPUs

- Several OpenMP compilers now produce DWARF for GPU code
  - A baseline requirement for *any* OMP TARGET region debugging
- The OMPD 3.1 prototype supports only OMP 3.1 (e.g., no GPUs)
  - OMPD 5.0 specification also supports OMP 5.0 TARGET (e.g., GPUs)
  - OMPD 3.1 prototype (significantly) different than OMPD 5.0

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

OMPD Futures

- Help push toward OMPD 5.0 standardization
- Need OMPD library support from OpenMP vendors
  - IBM LOMP/LLVM support in the fall?
  - Intel support on the radar screen?
- TotalView modifications needed for OMPD 5.0
  - Adapt to OMPD 3.1 → OMPD 5.0 API changes
  - Add DEVICE and TARGET region support (e.g., NVIDIA GPUs to start)
  - Details will depend on the OMP runtime TARGET execution model
- Other possible enhancements
  - Extend TotalView's displays to factor in OpenMP constructs
  - Create new TotalView displays to show the overall structure of an OpenMP program
  - First-class OpenMP language support in DWARF

© 2018 ROGUE WAVE SOFTWARE, INC. ALL RIGHTS RESERVED.

TotalView Agenda

- TotalView Debugging on Sierra CORAL with GPUs and OpenMP
- CUDA Unified Breakpoints Project
- OpenMP Debugging (OMPD) Standardization Effort
- TotalView Roadmap
- TotalView Technical Support

Peter Thompson

Technical Support Specialist

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView Support Model for LLNL

- Help LLNL staff manage and qualify TotalView releases
- Provide LLNL users customized consultation services
- Provide resources for users at LLNL to search for effective debugging techniques on their own
- Provide an efficient and effective process for submitting, managing and prioritizing TotalView bugs and enhancements important to LLNL
- Roles and Responsibilities:
  - support LLNL users
  - coordinate releases and new versions on LLNL systems
  - test fixes and new features
  - monitor issues generated at and for LLNL
  - develop a Knowledge base
  - develop WebEx learning sessions

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

TotalView Support for LLNL

- Contact: tv-support@lists.llnl.gov
- Availability: 9am-5pm ET, response within 1 business day
- Planned on-site visits: Twice a year
- Security access: Limited to RZ

© 2018 Rogue Wave Software, Inc. All Rights Reserved.

RogueWave
SOFTWARE

Innovate with Confidence