Hewlett Packard
Enterprise

DEBUGGING AND PERFORMANCE PROFILING ON HPE CRAY SUPERCOMPUTERS WITH AMD GPUS

Steve Abbott, Kostas Makrides, & Trey White

May 02, 2022

INTRODUCTION

- Part 1 – Debugging
  - Entomology – what kind of bug do you have?
  - Tools to find your bug
  - Using runtime logging to understand your bug
  - Active debugging

- Part 2 – Profiling
  - Profiling with *Perftools*
  - Visualizing performance with *Apprentice2*
  - *Rocprof*, the other tracing tool
  - Bonus topic: Assembly!

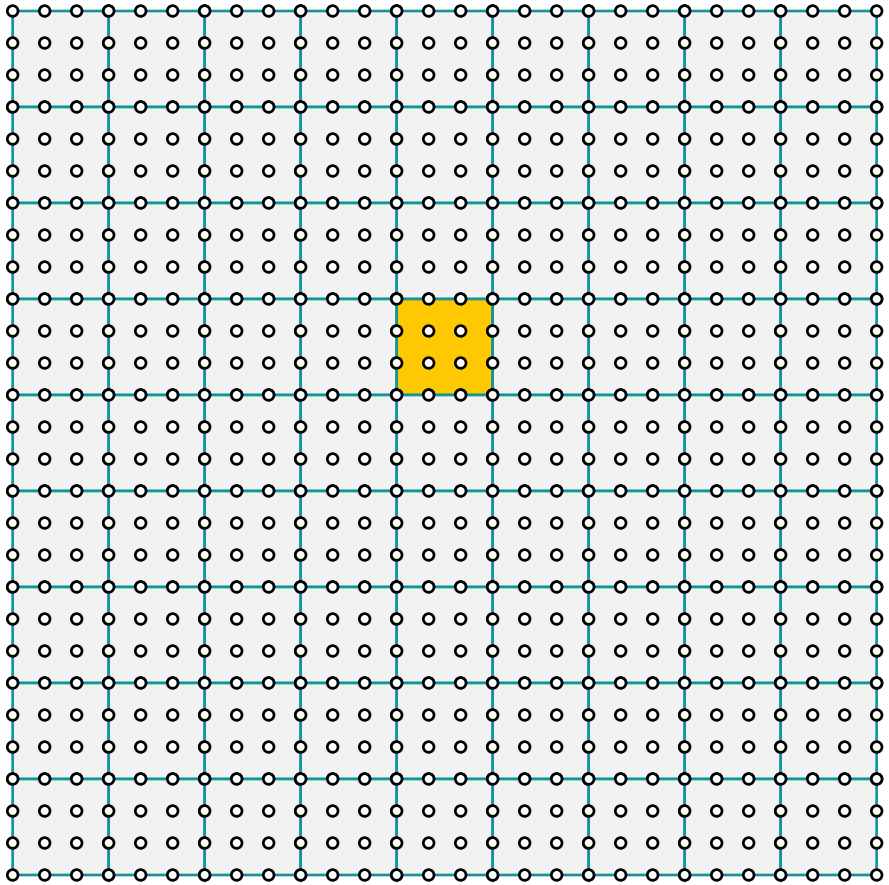
FACES: SPECTRAL-ELEMENT MICROBENCHMARK

- 3D grid of spectral elements
- That share faces that must be summed
- Partitioned across MPI tasks
- With contiguous buffers for MPI

https://upload.wikimedia.org/wikipedia/commons/c/ce/USDA-ARS_Guinea_Pig.jpg

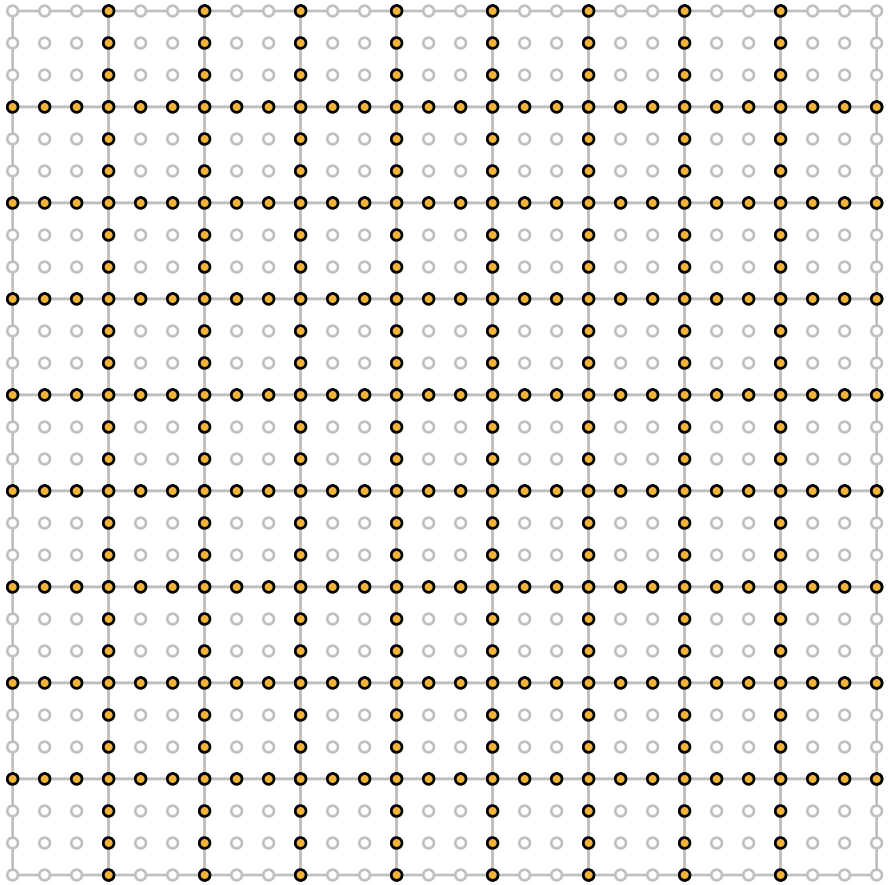
FACES: SPECTRAL-ELEMENT MICROBENCHMARK

- **3D grid of spectral elements**
- That share faces that must be summed
- Partitioned across MPI tasks
- With contiguous buffers for MPI



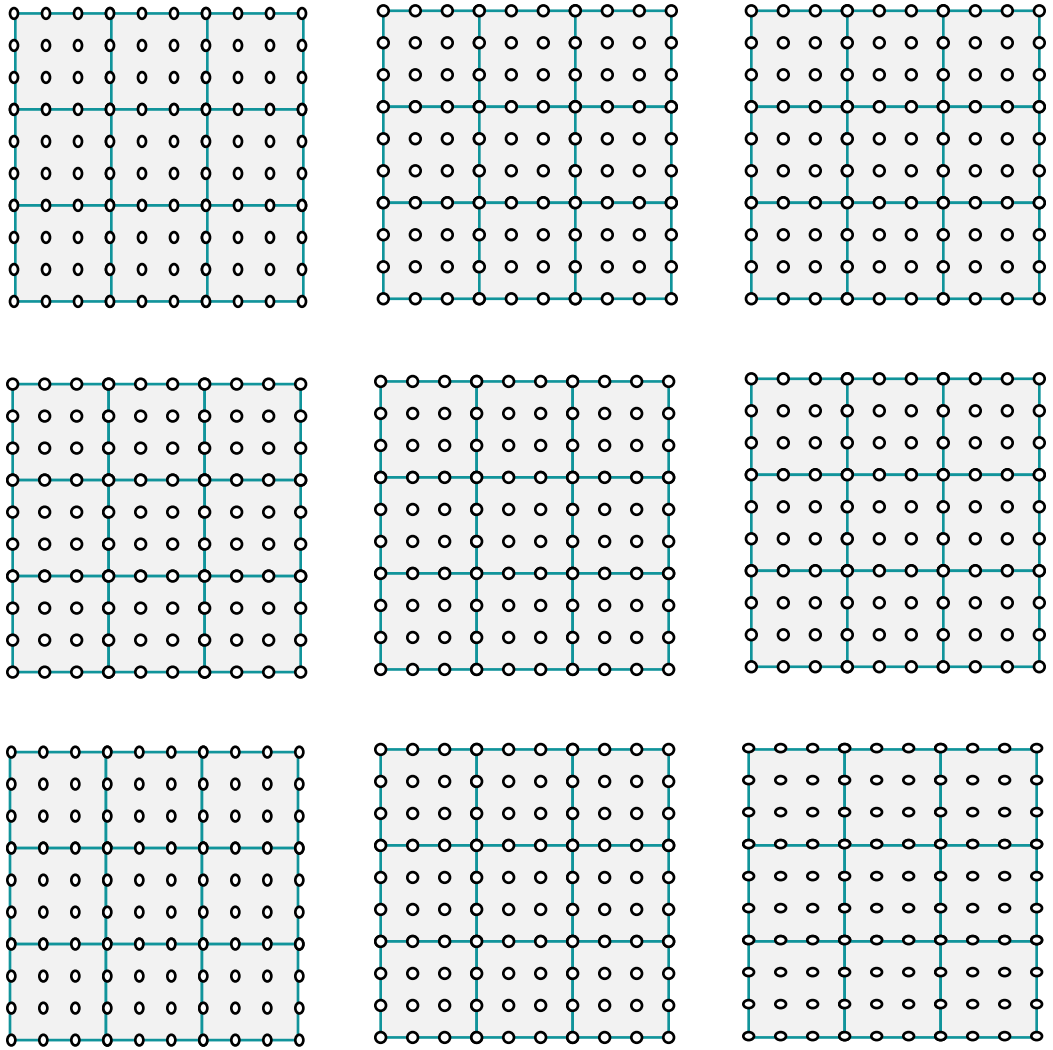
FACES: SPECTRAL-ELEMENT MICROBENCHMARK

- 3D grid of spectral elements
- **That share faces that must be summed**
- Partitioned across MPI tasks
- With contiguous buffers for MPI



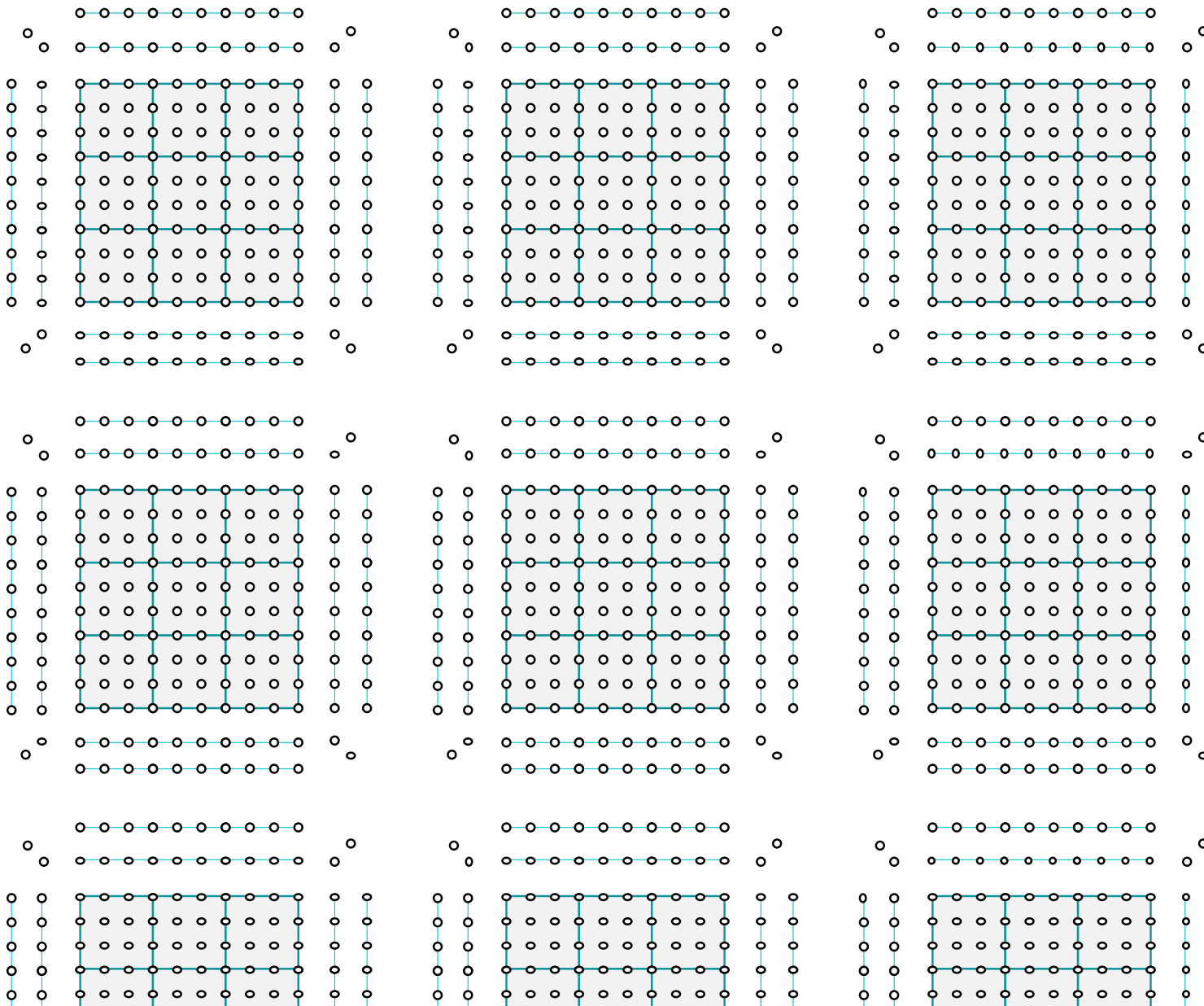
FACES: SPECTRAL-ELEMENT MICROBENCHMARK

- 3D grid of spectral elements
- That share faces that must be summed
- **Partitioned across MPI tasks**
- With contiguous buffers for MPI



FACES: SPECTRAL-ELEMENT MICROBENCHMARK

- 3D grid of spectral elements
- That share faces that must be summed
- Partitioned across MPI tasks
- **With contiguous buffers for MPI**



Hewlett Packard
Enterprise

PART 1: DEBUGGING ON HPE CRAY SUPERCOMPUTERS WITH AMD GPUS

Steve Abbott

May 02, 2022

ENTOMOLOGY
WHAT KIND OF BUG IS THAT?

THE MAJOR TYPES OF BUGS

- Crashing bugs
  - One or more processes in your application terminate
  - The most common kind
  - Generally (but not always) the easiest kind to solve
- Hangs
  - Deadlocks – everyone is stuck waiting for something that never happens
  - Livelocks – everyone is playing hot potato, calling different functions but not progressing
- Race conditions
  - One or more actors accessing the same data at the same time in a nondeterministic way
  - Shows up as changing results or sometimes crashes

CRASHING BUGS ON THE CPU

- Most crashing bugs will generate a signal
- "man 7 signal" can act as a cheat sheet

Signal Abbreviation (Number)	Signal Name	What it means
SIGSEGV (11)	Segmentation Fault, AKA Seg Fault	You attempted to access memory that technically exists on the machine but is outside the virtual address space the kernel gave you
SIGBUS (10,7)	Bus error	You attempted to access memory that cannot possibly be accessed
SIGABT (6)	Abort	Your application, or a library it uses, realized something was wrong and crashed intentionally
SIGFPE (8)	Floating Point Exception	You did some dangerous floating point math *and* asked to be notified about it

CRASHING BUGS ON THE AMD GPU

- Most crashing bugs will raise an exception on the CPU
- The runtime will map the exception to the analogous signal and raise it

What you'll see*	Signal	What it means
Memory access fault by GPU node-5 (Agent handle: 0x528e80) on address 0x7f223b7ad000. Reason: Page not present or supervisor privilege.	SIGSEGV	You tried to access memory that the GPU *could* access but isn't allowed to
HSA_STATUS_ERROR_MEMORY_FAULT: Agent attempted to access an inaccessible address. code: 0x2b	SIGSEGV	You tried to access memory that the GPU can't access
HSA_STATUS_ERROR_MEMORY_APERTURE_VIOLATION: The agent attempted to access memory beyond the largest legal address. code: 0x29	SIGBUS	You tried to access memory that the GPU cannot possibly access
HSA_STATUS_ERROR_EXCEPTION: An HSAIL operation resulted in a hardware exception. code: 0x1016	SIGABT	The code realized something was wrong and bailed out

* HSA errors will be prefaced by something like:

```
:0:rocdevice.cpp            :2589: 109972314012 us: Device::callbackQueue aborting with error :
```

MPI ERRORS

- Debugging MPI bugs could be another full talk, so just a few notes
- When MPI detects an error, it will invoke its own error handler and use that
- For the most part, MPI doesn’t care if the buffers are GPU's or CPU's
- Out of bounds memory accesses will be hit with CPU like signals, even if they land in GPU memory
  - The failure mode of crashing bugs is determined by WHO does the accessing, not where the accessing is
- MPI calls return instructive error codes!
  - Most codes don’t bother checking these, since most MPI’s default to abort on error
  - You can always set MPICH_ABORT_ON_ERROR=0 and actually do some error checking

HEISENBUGS

- A “Heisenbug” is a bug that disappears when you go looking for it

Why do bugs move around?

- Rebuilding with different optimization levels might pad memory differently, causing a bad access to land in benign memory instead of segfault
- Running in a debugger might add additional synchronization

In all cases, think about *what* the bug moving is telling you!

TOOLS FOR FINDING YOUR BUG

STEP 0: CHECK YOUR ERROR CODES!!

- If library authors went to the trouble of returning error codes, you should check them!
- A drop in macro you can use:

```
#define HIP_RC(hipCall) { \
  hipError_t e = hipCall; \
  if (e != hipSuccess) { \
    e = hipGetLastError();                                          \
    fprintf(stdout, "%s:%d -- %s returned %d:%s\n ",                \
            __FILE__, __LINE__, #hipCall ,  e, hipGetErrorString(e)); abort();}}
```

- Gives nice errors on failure:

```
simple_hmm.c:21 -- hipMalloc(&device, -1 * sizeof(int) * 1024) returned
2:hipErrorOutOfMemory
```

CORE FILES FOR POST-MORTEM ANALYSIS

- Most crashing signals will drop a core containing the process memory when hit
- See "man 7 signal" for tables

```
SIGSEGV      11       Core    Invalid memory reference
```

- Your user limits need to allow it

```
faces-tests> ulimit -c
unlimited
```

```
faces-tests>sh run-mi250x.sh 2 1 1 1
srun: job 192216 queued and waiting for resources
srun: job 192216 has been allocated resources
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
2 1 1 tasks
15 14 13 local elements of size 12
10 face inits x 10 element inits x 100 shares
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
Initialized Mugs: 15 x 14 x 13 elements of order 11 on 2 x 1 x 1 tasks
srun: error: x1000c2s2b0n0: task 1: Segmentation fault
srun: error: x1000c2s2b0n0: task 0: Segmentation fault (core dumped)
faces-tests>du -h core
137M    core
```

LOADING A CORE FROM A CPU CRASH

```
faces-tests>gdb faces core
GNU gdb (GDB; SUSE Linux Enterprise 15) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
(gdb) bt
#0  Mugs::share (this=<optimized out>, u=...) at Mugs.cpp:382
#1  0x000000000025586b in main (argc=<optimized out>, argv=<optimized out>) at main.cpp:152
(gdb) l
377           for (int jz = 0; jz < mz_; jz++) {
378             for (int iz = 0; iz < n_; iz++) {
379               u(0,0,iz,0,0,jz) += rzedge_(iz,jz,0);
380               u(nm1,0,iz,mxm1,0,jz) += rzedge_(iz,jz,1);
381               u(0,nm1,iz,0,mym1,jz) += rzedge_(iz,jz,2);
382               u(nm1,nm1,iz,mxm1,mym1,jz+10) += rzedge_(iz,jz,3);
383             }
384           }
385         }
386
(gdb) p u
$1 = (Array<double, 6> &) @0x7fffb45cd688: {sizes_ = {12, 12, 12, 15, 14, 13}, strides_ = {12,
    144, 1728, 25920, 362880, 4717440}, values_ = 0x34a6770}
(gdb) p jz
$2 = 4
```

LOADING A CORE FROM A GPU CRASH

```
faces-tests>sh run-mi250x.sh 2 1 1 1
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
2 1 1 tasks
15 14 13 local elements of size 12
10 face inits x 10 element inits x 100 shares
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
Initialized Mugs: 15 x 14 x 13 elements of order 11 on 2 x 1 x 1 tasks
Initialized Faces: 15 x 14 x 13 elements of order 11 on 2 x 1 x 1 tasks
:0:rocdevice.cpp            :2603: 185159763839 us: 35283: [tid:0x7f7b27df1700] Device::callbackQueue aborting with error
: HSA_STATUS_ERROR_MEMORY_FAULT: Agent attempted to access an inaccessible address. code: 0x2b
:0:rocdevice.cpp            :2603: 185159763855 us: 35284: [tid:0x7f13c61c7700] Device::callbackQueue aborting with error
: HSA_STATUS_ERROR_MEMORY_FAULT: Agent attempted to access an inaccessible address. code: 0x2b
srun: error: x1000c2s2b0n0: task 0: Aborted
srun: error: x1000c2s2b0n0: task 1: Aborted (core dumped)
```

```
faces-tests>rocgdb faces core
GNU gdb (rocm-rel-5.0-72) 11.1
```

LOADING A CORE FROM A GPU CRASH

```
faces-tests>rocgdb faces core
GNU gdb (rocm-rel-5.0-72) 11.1
(gdb) bt
#0  0x00007f13d428f18b in raise () from /lib64/libc.so.6
#1  0x00007f13d4290585 in abort () from /lib64/libc.so.6
#2  0x00007f13d981c889 in ?? () from /global/opt/rocm-5.0.2/lib/libamdhip64.so.5
#3  0x00007f13cc69420c in rocr::AMD::AqlQueue::ExceptionHandler(long, void*) ()
   from /global/opt/rocm-5.0.2/lib/libhsa-runtime64.so.1
#4  0x00007f13cc6d146b in rocr::core::Runtime::AsyncEventsLoop(void*) ()
   from /global/opt/rocm-5.0.2/lib/libhsa-runtime64.so.1
#5  0x00007f13cc6765c7 in rocr::os::ThreadTrampoline(void*) () from /global/opt/rocm-5.0.2/lib/libhsa-runtime64.so.1
#6  0x00007f13cc020a1a in start_thread () from /lib64/libpthread.so.0
#7  0x00007f13d4355d0f in clone () from /lib64/libc.so.6
(gdb) info thread
  Id   Target Id                         Frame
* 1    Thread 0x7f13c61c7700 (LWP 35289) 0x00007f13d428f18b in raise () from /lib64/libc.so.6
  2    Thread 0x7f13da63be00 (LWP 35284) warning: Section `.reg-xstate/35284' in core file too small.

0x00007f13cc6be0fc in rocr::core::InterruptSignal::WaitRelaxed(hsa_signal_condition_t, long, unsigned long, hsa_wait_state
_t) () from /global/opt/rocm-5.0.2/lib/libhsa-runtime64.so.1
  3    Thread 0x7f13abfff700 (LWP 35292) warning: Section `.reg-xstate/35292' in core file too small.
0x00007f13d434a099 in poll () from /lib64/libc.so.6
```

AMD GPU memory state is not currently part of the core dump!

LIMITATIONS OF CORE DUMPS

- Are the size of the process's occupied CPU memory
- Depending on system will either:
  - Only dump one core file -> maybe not enough information
  - Dump one core file for every failing process -> takes up a lot of space and is slow
- Don't contain AMD GPU memory state
- Are only postmortem

ABNORMAL TERMINATION PROCESSING (ATP)

Useful for crashes and *sometimes* hangs

- To use:
  - module load atp
  - Rebuild *or* just relink against libAtpSigHandler
  - The workload manager *does* need to be configured by admins to invoke ATP

```
faces-tests>HSA_XNACK=1 ATP_CORE_FILE_DIRECTORY=/lus/scratch/sabbott/faces-cores
ATP_GDB_BINARY=/opt/rocm-4.5.2/bin/rocgdb ATP_ENABLED=1 sh run-mi250x.sh 4 4 4 2
```

- What's on my command line?
  - **HSA_XNACK** – change an AMD GPU page fault setting (this just changes the type of error I get)
  - **ATP_CORE_FILE_DIRECTORY** – If ATP identifies useful core files, where should it put them?
  - **ATP_GDB_BINARY** – ATP will autodetect which gdb flavor it needs to load, but you can be explicit
  - **ATP_ENABLED** – Have ATP handle your signals

WHEN ATP IS INVOKED

My crash isn't subtle and hits all the nodes, so in the gasp of 64 dying ranks we see:

```
Memory access fault by GPU node-7 (Agent handle: 0x844280) on address 0x7fe0686bf
000. Reason: Unknown.
ATP analysis of Slurm job 192678.0 is starting...
```

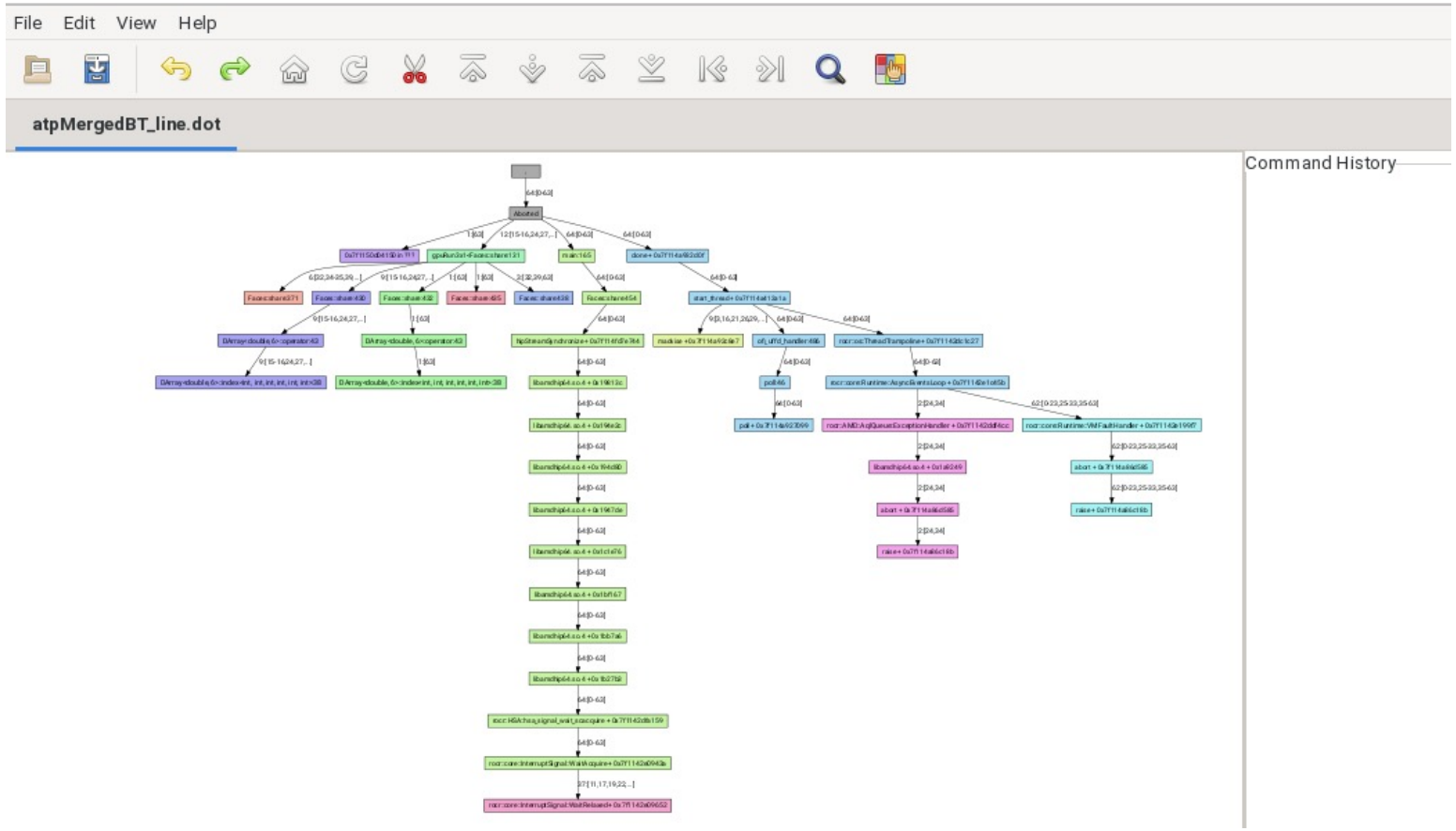
Then:

```
Processes died with the following statuses:
<0-63> Reason: 'Aborted' Address: 0xefb7 Assertion: ''

Producing core dumps for ranks 0 3 15 24 11 63
Failed to write core files. Ensure directory is accessible on backend: /lus/scrat
ch/sabbott/faces-cores
View application merged backtrace tree with: stat-view atpMergedBT.dot (function-
level) or atpMergedBT_line.dot (line-level)
You may need to: module load stat
```

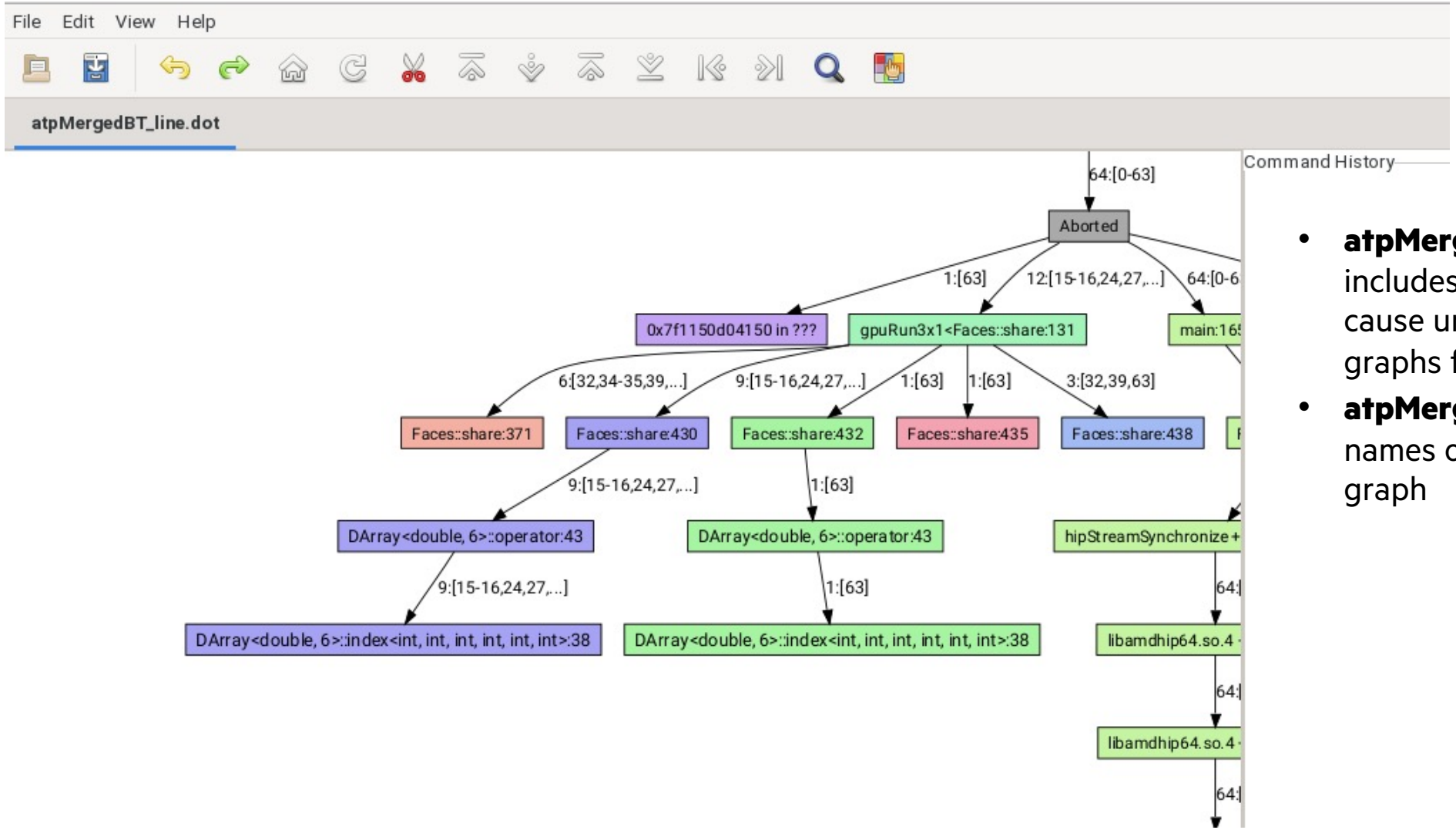
* I'm not sure why ATP couldn't dump my core files. I'm investigating

VIEWING THE TRACE

```
faces-tests> module load cray-stat
faces-tests> stat-view atpMergedBT_line.dot
```



GPU KERNEL POSITIONS

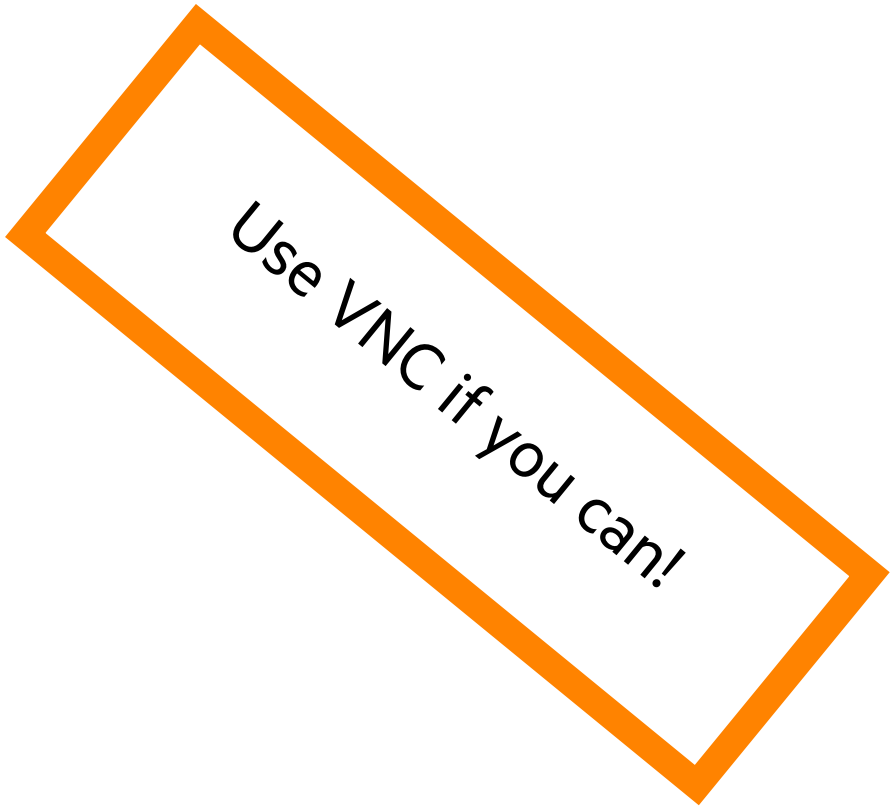


- **atpMergedBT_line.dot** – includes line numbers, and can cause unhelpfully complicated graphs for large applications
- **atpMergedBT.dot** – function names only, makes a cleaner graph

THE STACK TRACE ANALYSIS TOOL (STAT)

Useful for hangs

- Nothing special required, just
  `module load cray-stat`




```
faces-tests> sh run-mi250x.sh 4 4 4 2
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
2 with node rank 2 using device 2 (8 devices per node) (asked for 2)
3 with node rank 3 using device 3 (8 devices per node) (asked for 3)


62 with node rank 30 using device 6 (8 devices per node) (asked for 6)
63 with node rank 31 using device 7 (8 devices per node) (asked for 7)
Initialized Mugs: 15 x 14 x 13 elements of order 11 on 4 x 4 x 4 tasks
Initialized Faces: 15 x 14 x 13 elements of order 11 on 4 x 4 x 4 tasks
```

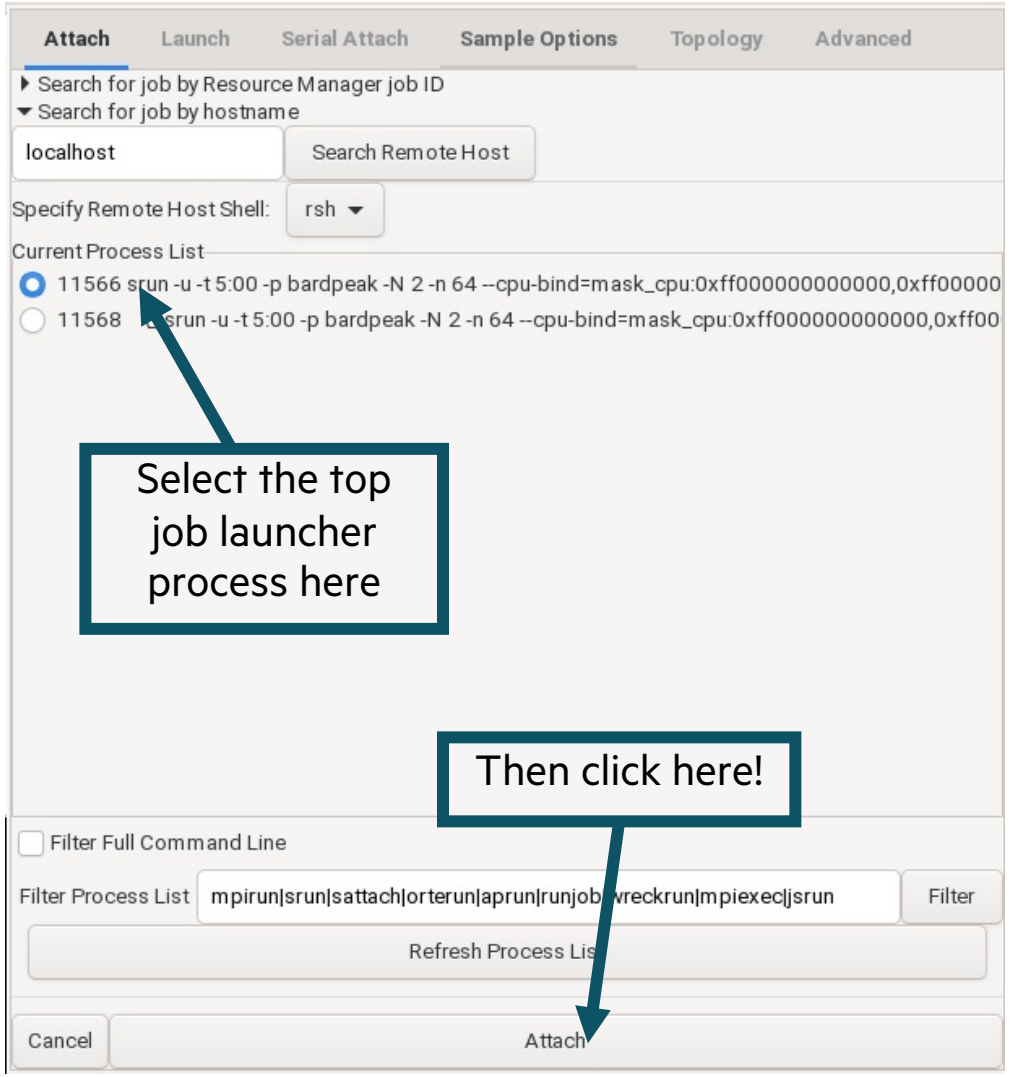
ATTACHING WITH STAT-GUI

```
STAT_GDB=/opt/rocm-4.5.2/bin/rocgdb
stat-gui -G -w -i
```

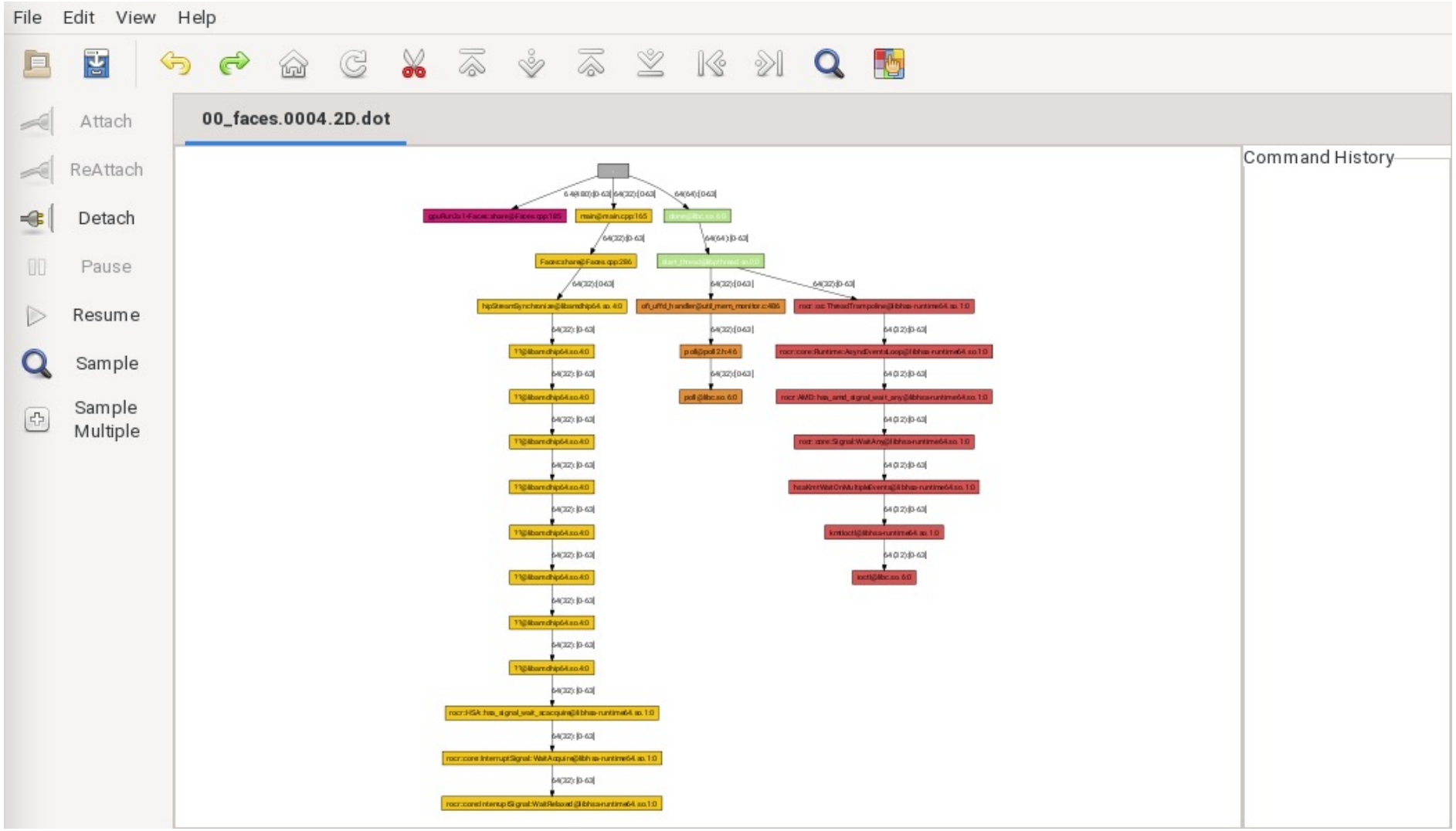
What's on my command line?

- **STAT_GDB** – Pick which gdb stat should use
- **stat-gui** – The stat command that launches an interactive window
- **-G** – Use the gdb backend to attach and trace
- **-w** – Trace threads, including GPU threads
- **-i** – Sample line numbers (use with caution)

All the above can be configured through the "Sample Options" and "Advanced" tabs too!

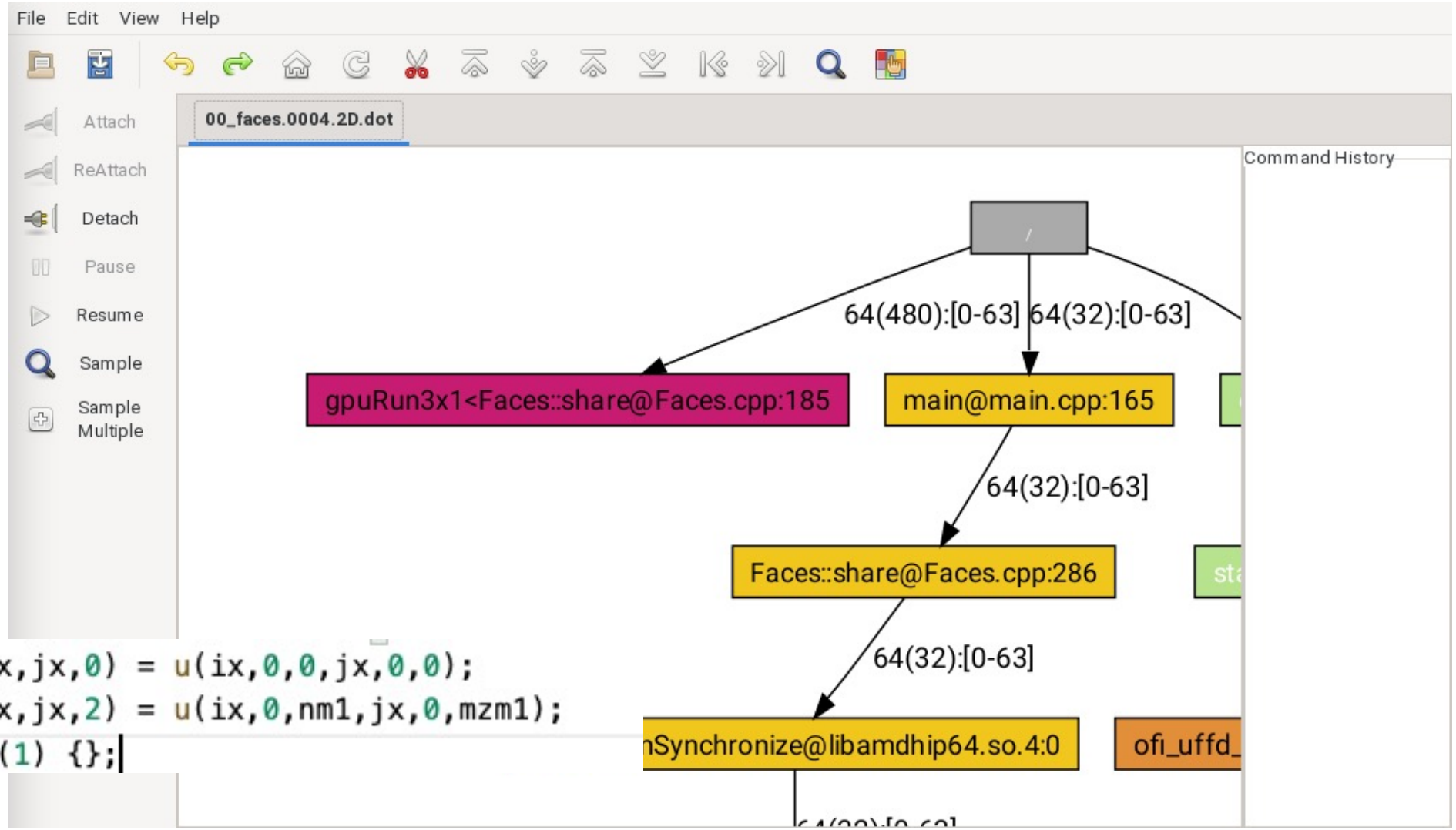


STAT TRACES


File
Edit
View
Help
Attach
ReAttach
Detach
Pause
Resume
Sample
Sample Multiple
00_faces.0004.2D.dot
Command History

ZOOMING IN TO THE GPU THREAD

Yes, that is the real "bug" location



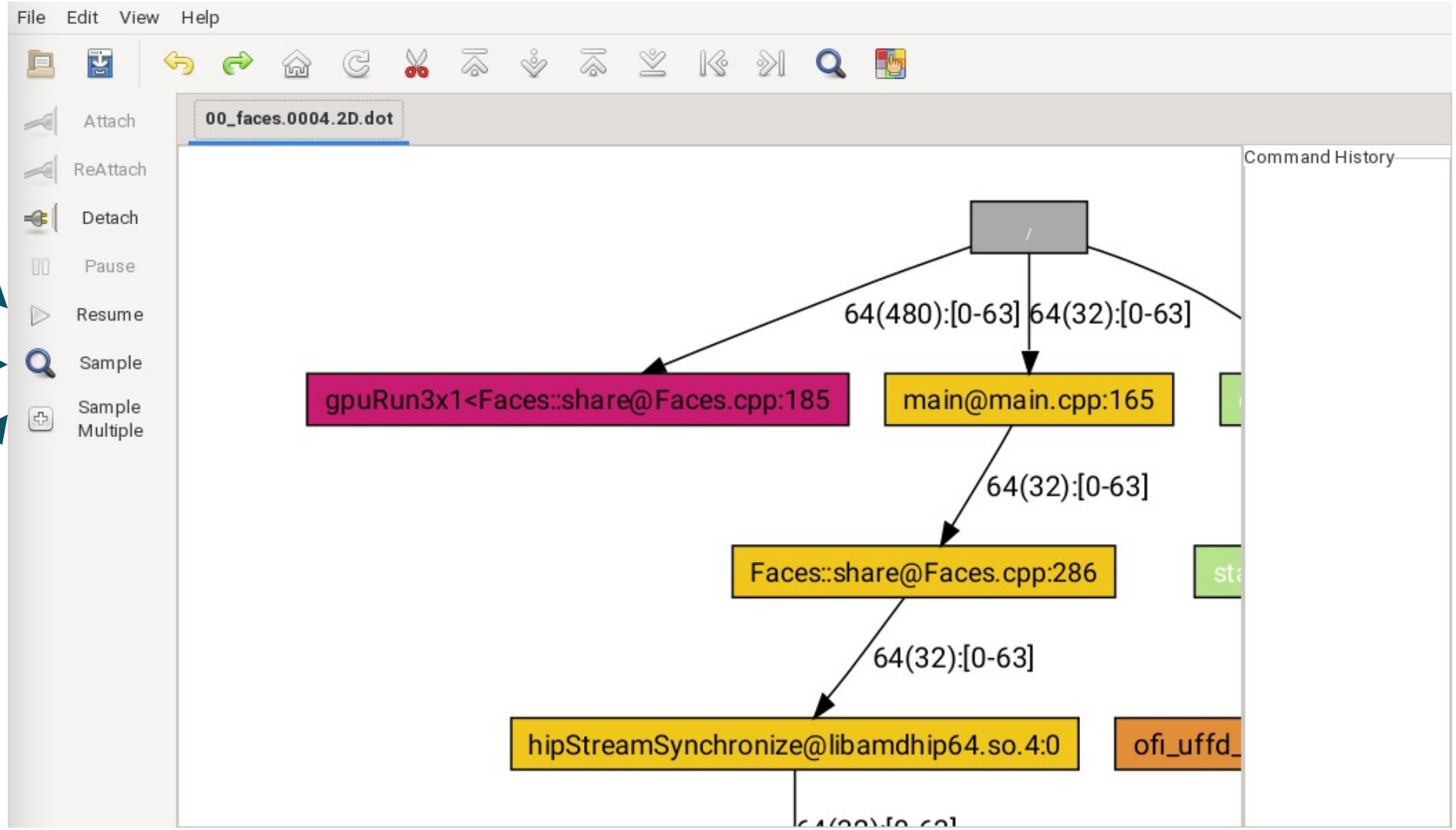

```
184     xes(ix,jx,0) = u(ix,0,0,jx,0,0);
185     xes(ix,jx,2) = u(ix,0,nm1,jx,0,mzm1);
186     while(1) {};
```

SOME OTHER THINGS YOU CAN DO

Un-pause but stay attached

Take another sample (maybe with different options)

Take some time sliced samples



SIPPING FROM THE FIREHOSE: USING RUNTIME DEBUG INFORMATION

THE CRAY OPENMP TARGET RUNTIME

- The Cray OpenMP and OpenACC runtimes will print debug information to stderr on demand
- **CRAY_ACC_DEBUG=1**
  - Concise, a good way to tell your offload regions are running
  - Probably not useful for more complex debugging
- **CRAY_ACC_DEBUG=2**
  - Designed to be user friendly and where you should start
  - Shows what the runtime is doing but not nitty gritty details
- **CRAY_ACC_DEBUG=3**
  - Very verbose, not designed for everyday users but very powerful in expert hands
  - If you need to look at memory addresses, this is your level

THREE VIEWS OF AN EXPLOSION

```
faces-tests> MPICH_GPU_SUPPORT_ENABLED=1 CRAY_ACC_DEBUG=0 srun -u -n 1 -N 1 -c 1 --
pty  --exclusive  ./faces-mi200 < opt.in &

&testfaces lx=1,ly=1,lz=1,mx=15,my=14,mz=13,n=12,niface=1,niel=10,nshare=100 /

 3*1 tasks
 15,  14,  13 local elements of size 12
 1 face inits x 10 element inits x 100 shares
 0 with node rank 0 using device 0 ( 8 devices per node )
 Initialized mugs: 15 x 14 x 13 elements of order 11 on 1 x 1 x 1 tasks
 Initialized faces: 15 x 14 x 13 elements of order 11 on 1 x 1 x 1 tasks
 0 FAIL 1.,  12,  5*1,  10101.010112,  1.28045515244161363E+34
 time 3.695112270999922 avg 3.695112270999922 min 3.695112270999922 max
```

What went wrong?

WITH CRAY_ACC_DEBUG=1

```
 Initialized faces: 15 x 14 x 13 elements of order 11 on 1 x 1 x 1 tasks
ACC: Transfer 7 items (to acc 2737280 bytes, to host 0 bytes) from faces.f90:109
ACC: Transfer 1 items (to acc 37739520 bytes, to host 0 bytes) from main.f90:53
ACC: Execute kernel main_$ck_L53_5 async(auto) from main.f90:53
ACC: Wait async(auto) from main.f90:53
ACC: Transfer 1 items (to acc 0 bytes, to host 37739520 bytes) from main.f90:53
ACC: Transfer 8 items (to acc 37739520 bytes, to host 0 bytes) from faces.f90:194
ACC: Join async(auto) to async(0) from faces.f90:237
```

```
ACC: Execute kernel share_faces$faces_$ck_L876_22 async(7) from faces.f90:876
ACC: Transfer 8 items (to acc 0 bytes, to host 0 bytes) async(7) from faces.f90:901
ACC: Synchronize
ACC: Wait async(auto) from faces.f90:908
ACC: Transfer 8 items (to acc 0 bytes, to host 0 bytes) from faces.f90:908
 0 FAIL 1.,  12,  5*1,  10101.010112,  1.28045515244161363E+34
 time 3.771113195999875 avg 3.771113195999875 min 3.771113195999875 max
```

WITH CRAY_ACC_DEBUG=2

```
ACC: Execute kernel share_faces$faces_$ck_L876_22 blocks:1 threads:1 async(7) from faces.f90:876
ACC: Start transfer 8 items async(7) from faces.f90:901
ACC:       free '$_acc_corner_T1002(:,:)' (128 bytes)
ACC:       release present 'u(:,:,:,:,:,:)' (37739520 bytes)
ACC:       free '$_acc_xedge_T1008(:,:,:,:)' (11520 bytes)
ACC:       free '$_acc_xface_T1014(:,:,:,:,:,:)' (838656 bytes)
ACC:       free '$_acc_yedge_T1006(:,:,:,:)' (10752 bytes)
ACC:       free '$_acc_yface_T1012(:,:,:,:,:,:)' (898560 bytes)
ACC:       free '$_acc_zedge_T1004(:,:,:,:)' (9984 bytes)
ACC:       free '$_acc_zface_T1010(:,:,:,:,:,:)' (967680 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC: Synchronize
ACC: Wait async(auto) from faces.f90:908
ACC: Start transfer 8 items from faces.f90:908
ACC:       release present 'corner_(:,:)' (128 bytes)
ACC:       free 'u(:,:,:,:,:,:)' (37739520 bytes)
ACC:       release present 'xedge_(:,:,:,:)' (11520 bytes)
ACC:       release present 'xface_(:,:,:,:,:,:)' (838656 bytes)
ACC:       release present 'yedge_(:,:,:,:)' (10752 bytes)
ACC:       release present 'yface_(:,:,:,:,:,:)' (898560 bytes)
ACC:       release present 'zedge_(:,:,:,:)' (9984 bytes)
ACC:       release present 'zface_(:,:,:,:,:,:)' (967680 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
 0 FAIL 1.,  12,  5*1,  10101.010112,  1.28045515244161363E+34
 time 3.9777042649998293 avg 3.9777042649998293 min 3.9777042649998293 max
```

WITH CRAY_ACC_DEBUG=3

We should probably copy back that state vector…

```
194       !$omp target data map(to:u) &
195       !$omp use_device_ptr(xface_,yface_,zface_,xedge_,yedge_,zedge_,corner_)
196
ACC:
ACC:    Trans 2
ACC:        Simple transfer of 'u(:,:,:,:,:)' (37739520 bytes)
ACC:             host ptr 10000e60580
ACC:             acc  ptr 0
ACC:             flags: FREE REL_PRESENT REG_PRESENT INIT_ACC_PTR
ACC:             host region 10000e60580 to 1000325e180 found in present table index 8 (ref count 1)
ACC:             last release acc 7f3c20000000 from present table index 8 (ref_count 1)
ACC:             last release of conditional present (acc 7f3c20000000, base 7f3c20000000)
ACC:             remove acc 7f3c20000000 from present table index 8
ACC:             new acc ptr 0
```

THE AMD OPENMP TARGET RUNTIME

- Builds and contributes to LLVM OpenMP Target runtime
- Uses the mechanisms at https://openmp.llvm.org/design/Runtimes.html#libomptarget-info
- Compile with “-g” to get sensible name
- Set `LIBOMPTARGET_INFO` to control *what* is printed, but not how much
  - This is a bitfield
  - See the link above for fine grained details
  - Set to -1 to get it all
- There is a separate “debug”, but that’s for library developers!
  - If you really need it, there’s a build in `${ROCM_PATH}/llvm/lib-debug`

```
faces-tests> LIBOMPTARGET_INFO=-1 srun -n 1 ./a.out
Libomptarget device 0 info: Entering OpenMP kernel at reduction.c:10:3 with 1 arguments:
Libomptarget device 0 info: tofrom(a)[8]
The result is correct on target = 499999500000!
Success!
```

WITH LIBOMPTARGET_DEBUG

```
faces-tests> LIBOMPTARGET_DEBUG=2 srun -n 1 ./a.out
Libomptarget --> Init target library!
Libomptarget --> Loading RTLs...
Libomptarget --> Loading library '/opt/rocm/llvm/lib-debug/libomptarget.rtl.x86_64.so'...
Libomptarget --> Successfully loaded library '/opt/rocm/llvm/lib-debug/libomptarget.rtl.x86_64.so'!
Libomptarget --> Registering RTL libomptarget.rtl.x86_64.so supporting 4 devices!
Libomptarget --> Loading library '/opt/rocm/llvm/lib-debug/libomptarget.rtl.amdgpu.so'...
Target AMDGPU RTL --> Start initializing HSA-ATMI
Target AMDGPU RTL --> There are 8 devices supporting HSA.
Target AMDGPU RTL --> Device 0: Initial groupsPerDevice 128 & threadsPerGroup 256
Target AMDGPU RTL --> Device 1: Initial groupsPerDevice 128 & threadsPerGroup 256
```

```
Target AMDGPU RTL --> Entry point 0 maps to __omp_offloading_6f2771a4_4b002663_main_l10
Libomptarget --> Entry  0: Base=0x00007ffea9917550, Begin=0x00007ffea9917550, Size=8, Type=0x23, Name=unknown
Libomptarget --> Looking up mapping(HstPtrBegin=0x00007ffea9917550, Size=8)...
Target AMDGPU RTL --> Tgt alloc data 8 bytes, (tgt:00007fa8aba00000).
Libomptarget --> Creating new map entry: HstBase=0x00007ffea9917550, HstBegin=0x00007ffea9917550, HstEnd=0x00007ffea9917558, TgtBeg
in=0x00007fa8aba00000
Libomptarget --> There are 8 bytes allocated at target address 0x00007fa8aba00000 - is new
Libomptarget --> Moving 8 bytes (hst:0x00007ffea9917550) -> (tgt:0x00007fa8aba00000)
Target AMDGPU RTL --> Submit data 8 bytes, (hst:00007ffea9917550) -> (tgt:00007fa8aba00000).
Libomptarget --> Looking up mapping(HstPtrBegin=0x00007ffea9917550, Size=8)...
Libomptarget --> Mapping exists with HstPtrBegin=0x00007ffea9917550, TgtPtrBegin=0x00007fa8aba00000, Size=8, RefCount=1
Libomptarget --> Obtained target argument 0x00007fa8aba00000 from host pointer 0x00007ffea9917550
Libomptarget --> Launching target execution __omp_offloading_6f2771a4_4b002663_main_l10 with pointer 0x000000000077f7d0 (index=0).
```

AMD HIP AND HSA RUNTIMES

- If the OpenMP runtimes are firehoses, the HIP runtime is an Ocean
- AMD_LOG_LEVEL environment variable (higher is inclusive of lower)
  - 0 – off
  - 1 – print errors
  - 2 – print warnings
  - 3 – print info
  - 4 - print detailed debugging information
- You can further fine tune *what* gets logged with AMD_LOG_MASK
  - See https://docs.amd.com/bundle/AMD_HIP_Programming_Guide/page/Programming_with_HIP.html if you need to do this

AN EXAMPLE WHERE AMD_LOG_LEVEL HELPS A LOT

```
faces-tests> sh run-mi250x.sh 1 1 1 1
"hipErrorNoBinaryForGpu: Unable to find code object for all current devices!"
srun: error: x1000c2s2b0n0: task 0: Aborted
faces-tests>
```

```
faces-tests> ROCR_VISIBLE_DEVICES=1 AMD_LOG_LEVEL=1 sh run-mi250x.sh 1 1 1 1
:1:rocdevice.cpp            :1573: 274572673160 us: HSA_AMD_AGENT_INFO_SVM_DIRECT_HOST_ACCESS query failed.
:1:hip_code_object.cpp      :460 : 274572673911 us: hipErrorNoBinaryForGpu: Unable to find code object for all current devices!
:1:hip_code_object.cpp      :461 : 274572673917 us:   Devices:
:1:hip_code_object.cpp      :464 : 274572673919 us:     amdgcn-amd-amdhsa--gfx90a:sramecc+:xnack- - [Not Found]
:1:hip_code_object.cpp      :468 : 274572673920 us:   Bundled Code Objects:
:1:hip_code_object.cpp      :485 : 274572673922 us:     host-x86_64-unknown-linux - [Unsupported]
:1:hip_code_object.cpp      :483 : 274572673923 us:     hipv4-amdgcn-amd-amdhsa--gfx908 - [code object v4 is amdgcn-amd-amdhsa--gfx908]
"hipErrorNoBinaryForGpu: Unable to find code object for all current devices!"
srun: error: x1000c2s2b0n0: task 0: Aborted
```

TURN UP THE FIREHOSE WITH CAUTION!

```
faces-tests> ROCR_VISIBLE_DEVICES=1 AMD_LOG_LEVEL=4 sh run-mi250x.sh 1 1 1 1
:3:rocdevice.cpp            :432 : 274800763181 us: Initializing HSA stack.
:3:comgrctx.cpp             :33  : 274800763231 us: Loading COMGR library.
:3:rocdevice.cpp            :204 : 274800763279 us: Numa selects cpu agent[3]=0x9605e0(fine=0x9607c0,coarse=0x960f40, kern_arg=0x96
1d40) for gpu agent=0x7fae76f70259
:1:rocdevice.cpp            :1573: 274800766022 us: HSA_AMD_AGENT_INFO_SVM_DIRECT_HOST_ACCESS query failed.
:3:rocdevice.cpp            :1577: 274800766030 us: HMM support: 1, xnack: 0, direct host access: 0

:4:rocdevice.cpp            :1873: 274800766067 us: Allocate hsa host memory 0x7fae77bca000, size 0x28
:4:rocdevice.cpp            :1873: 274800766237 us: Allocate hsa host memory 0x7fae53000000, size 0x101000
:4:rocdevice.cpp            :1873: 274800766382 us: Allocate hsa host memory 0x7fae52e00000, size 0x101000
:4:runtime.cpp              :82  : 274800766403 us: init
:3:hip_context.cpp          :49  : 274800766407 us: Direct Dispatch: 1
:1:hip_code_object.cpp      :460 : 274800766846 us: hipErrorNoBinaryForGpu: Unable to find code object for all current devices!
:1:hip_code_object.cpp      :461 : 274800766850 us:   Devices:
:1:hip_code_object.cpp      :464 : 274800766851 us:     amdgcn-amd-amdhsa--gfx90a:sramecc+:xnack- - [Not Found]
:1:hip_code_object.cpp      :468 : 274800766852 us:   Bundled Code Objects:
:1:hip_code_object.cpp      :485 : 274800766854 us:     host-x86_64-unknown-linux - [Unsupported]
:1:hip_code_object.cpp      :483 : 274800766855 us:     hipv4-amdgcn-amd-amdhsa--gfx908 - [code object v4 is amdgcn-amd-amdhsa--gfx
908]
"hipErrorNoBinaryForGpu: Unable to find code object for all current devices!"
srun: error: x1000c2s2b0n0: task 0: Aborted
faces-tests>
```

OTHER USEFUL ENVIRONMENT VARIABLES

Good for race conditions, and when you need to slow things down

- *Most* AMD flags are bitfields
- AMD_SERIALIZE_KERNEL
  - 1 = Synchronize *before* launches (i.e. make sure everything is done on the GPU)
  - 2 = Synchronize *after* launches (i.e. wait for kernel to finish before moving on)
  - 3 = Do both 1 and 2
- AMD_SERIALIZE_COPY
  - 1 = Synchronize *before* copies (i.e. make sure everything is done on the GPU)
  - 2 = Synchronize *after* copies (i.e. wait for copy to finish before moving on)
  - 3 = Do both 1 and 2
- For a writeup and other tips see debugging sections of:
  - https://docs.amd.com/bundle/AMD_HIP_Programming_Guide/page/Programming_with_HIP.html
- For raw flags, which may or may not do what you want:
  - https://github.com/ROCm-Developer-Tools/ROCclr/blob/develop/utils/flags.hpp

DIAGNOSING A SYNCHRONIZATION ERROR

```
faces-tests> sh run-mi250x.sh 4 4 4 4
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
2 with node rank 2 using device 2 (8 devices per node) (asked for 2)
```

```
48 FAIL 1 (11,4,0,0,0,0) 4.35055e+48 9.64172e+64 9.64172e+64 1
32 FAIL 1 (11,4,0,0,0,0) 5.55175e+48 9.64172e+64 9.64172e+64 1
30 FAIL 1 (11,4,0,0,0,0) 4.35134e+48 9.64172e+64 9.64172e+64 1
time 4.07344 avg 4.04031 min 4.12512 max
```

We’re running to completion but getting wrong results.

Can we figure out why by using environment variables?

CHECK FOR GPU AND CPU SYNCHRONIZATION ISSUES

```
faces-tests> AMD_SERIALIZE_KERNEL=3 AMD_SERIALIZE_COPY=3 sh run-mi250x.sh 4 4 4 4
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
16 with node rank 0 using device 0 (8 devices per node) (asked for 0)
32 with node rank 0 using device 0 (8 devices per node) (asked for 0)
```

```
7 PASS
13 PASS
15 PASS
time 5.80683 avg 5.78838 min 5.83031 max
```

This is correct, so we probably have some race involving the GPU.

I know faces doesn't do many Host<->Device copies, so can I rule that out?

CHECK JUST KERNEL SYNCHRONIZATION

```
faces-tests> AMD_SERIALIZE_KERNEL=3 sh run-mi250x.sh 4 4 4 4
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
2 with node rank 2 using device 2 (8 devices per node) (asked for 2)
```

```
21 PASS
20 PASS
28 PASS
time 5.84433 avg 5.82545 min 5.8607 max
```

We are probably missing a synch between two kernels or between the host and a kernel.

Can we learn more?

SYNCHRONIZE *BEFORE* KERNEL LAUNCHES

```
faces-tests> AMD_SERIALIZE_KERNEL=1 sh run-mi250x.sh 4 4 4 4
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
2 with node rank 2 using device 2 (8 devices per node) (asked for 2)
```

```
44 FAIL 1 (0,0,0,0,0,0) 3.64285e+47 9.74609e+64 9.74609e+64 1
60 FAIL 1 (0,0,0,0,0,0) 1.70276e+167 9.74609e+64 1.70276e+167 1
47 FAIL 1 (1,1,0,0,0,0) 4.18e+87 7.60591e+34 4.18e+87 1
time 4.02045 avg 3.98708 min 4.08269 max
```

This still fails.

We are probably *not* having two kernels racing.

SYNCHRONIZE *AFTER* KERNEL LAUNCHES

```
faces-tests> AMD_SERIALIZE_KERNEL=2 sh run-mi250x.sh 4 4 4 4
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
2 with node rank 2 using device 2 (8 devices per node) (asked for 2)
```

```
16 PASS
25 PASS
17 PASS
time 5.8051 avg 5.79262 min 5.82121 max
```

```
283     // send in use order
284
285     //CHECK(hipStreamSynchronize(stream_[0]));
286
287     MPI_Isend(zfs.data(0,0,0,0,0),nface_[2],MPI_DOUBLE,iface_[4],tag,MPI_COMM_WORLD,reqs_+0);
288     MPI_Isend(zfs.data(0,0,0,0,1),nface_[2],MPI_DOUBLE,iface_[5],tag,MPI_COMM_WORLD,reqs_+1);
```

Why did we comment that out again?

THE ART OF ACTIVE DEBUGGING

ROCGDB

- AMD has made significant enhancements to gdb for debugging on their GPUs
  - Each wavefront is represented as a single thread
  - Non-stop mode works across both CPU and GPU
  - Newest rocgdb+driver+compilers allow symbolic debugging and per-lane inspection
  - Documentation available in `${ROCM_PATH}/share/doc/rocgdb/`
- It has some shortcomings:
  - It's not multiprocess (or not more than gdb is)
  - The debugger version requires the driver version match for GPU debugging
  - The native thread representation can get a bit overwhelming

REMEMBER: to use gdb or rocgdb from slurm you need to `srun --pty` to get a pseudoterminal!

A ROCGDB EXAMPLE

```
faces-tests> sh run-mi250x-rocgdb.sh 1 1 1 1
GNU gdb (rocm-rel-4.5-164) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
```

```
Reading symbols from ./faces...
(gdb) break Faces.cpp:336
Breakpoint 1 at 0x25f820: file Faces.cpp, line 449.
(gdb) run < opt.in
Starting program: /lus/cflus02/sabbott/faces/hip/gpu_subtle/faces < opt.in
```

```
1 1 1 tasks
15 14 13 local elements of size 12
10 face inits x 10 element inits x 100 shares
Initialized Mugs: 15 x 14 x 13 elements of order 11 on 1 x 1 x 1 tasks
Initialized Faces: 15 x 14 x 13 elements of order 11 on 1 x 1 x 1 tasks
[Switching to thread 3, lane 1 (AMDGPU Lane 6:4:1:1/1 (0,0,1)[1,0,0])]

Thread 3 "faces" hit Breakpoint 1, with lanes [1-10], Faces::share(DArray<double, 6>&):
:{lambda(int, int, int, int)#2}::operator()(int, int, int, int) const (this=<optimized
out>, ia=<optimized out>, ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) a
t Faces.cpp:336
336             u(ix,0,nm1,jx,jy,mzm1) += u(ix,nm1,nm1,jx,jy-1,mzm1)+zfr(ix,0,jx,jy,1)+
zfr(ix,nm1,jx,jy-1,1);
(gdb)
```

VIEWING THREADS IN ROCGDB

```
(gdb) info threads
  Id   Target Id                                  Frame
  1    Thread 0x7fffed9afe00 (LWP 4835) "faces"  0x00007fffe0009652 in rocr::core::InterruptSignal::
WaitRelaxed(hsa_signal_condition_t, long, unsigned long, hsa_wait_state_t) ()
   from /opt/rocm/lib/libhsa-runtime64.so.1
  2    Thread 0x7fffda395700 (LWP 4844) "faces"  0x00007fffe76ef807 in ioctl ()
   from /lib64/libc.so.6
* 3    AMDGPU Wave 6:4:1:1 (0,0,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  4    AMDGPU Wave 6:4:1:2 (0,1,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  5    AMDGPU Wave 6:4:1:3 (0,2,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  6    AMDGPU Wave 6:4:1:4 (0,3,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  7    AMDGPU Wave 6:4:1:5 (0,4,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  8    AMDGPU Wave 6:4:1:6 (0,5,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  9    AMDGPU Wave 6:4:1:7 (0,6,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  10   AMDGPU Wave 6:4:1:8 (0,7,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
  11   AMDGPU Wave 6:4:1:9 (0,8,1)/0 "faces"     Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int)#2}::operator()(int, int, int, int) const (this=<optimized out>, ia=<optimized out>,
    ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
```

Regular gdb goodness works!

- `info threads`
- `thread <number>`
- `backtrace`
- `break`
- `watch`
- `layout`

LAYOUT ASM IN ROCGDB ON A GPU THREAD

```
   0x7fffc56ef614 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+276>      v_or_b32_e32 v1, s83, v0
   0x7fffc56ef618 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+280>      v_cmp_eq_u32_e32 vcc, 0, v1
   0x7fffc56ef61c <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+284>      v_cmp_gt_i32_e64 s[10:11], s82, 0
   0x7fffc56ef624 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+292>      s_and_b64 s[6:7], vcc, s[10:11]
   0x7fffc56ef628 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+296>      v_cmp_gt_i32_e64 s[0:1], s101, 0
   0x7fffc56ef630 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+304>      s_and_b64 s[6:7], s[6:7], s[0:1]
   0x7fffc56ef634 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+308>      s_xor_b64 s[6:7], s[6:7], -1
   0x7fffc56ef638 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+312>      s_and_saveexec_b64 s[8:9], s[6:7]
   0x7fffc56ef63c <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+316>      s_xor_b64 s[6:7], exec, s[8:9]
   0x7fffc56ef640 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+320>      s_cbranch_execz 487  # 0x7fffc56efde0 <_Z
   0x7fffc56ef644 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+324>      v_cmp_eq_u32_e64 s[8:9], s83, 0
   0x7fffc56ef64c <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+332>      v_cmp_lt_i32_e32 vcc, 0, v0
   0x7fffc56ef650 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+336>      s_and_b64 s[8:9], vcc, s[8:9]
   0x7fffc56ef654 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+340>      s_and_b64 s[0:1], s[8:9], s[0:1]
   0x7fffc56ef658 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+344>      s_xor_b64 s[0:1], s[0:1], -1
   0x7fffc56ef65c <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+348>      s_and_saveexec_b64 s[8:9], s[0:1]
   0x7fffc56ef660 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+352>      s_xor_b64 s[8:9], exec, s[8:9]
   0x7fffc56ef664 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+356>      s_cbranch_execz 303  # 0x7fffc56efb24 <_Z
   0x7fffc56ef668 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+360>      v_cmp_eq_u32_e64 s[0:1], 0, v0
   0x7fffc56ef670 <_Z9gpuRun3x1IZN5Faces5shareER6DArrayIdLi6EEEUliiiiE0_EvT_iiii+368>      v_cmp_gt_i32_e64 s[44:45], s83, 0
rocm AMDGPU Wave 6:4:1:4 In: gpuRun3x1<Faces::share                                 L336  PC: 0x7fffc56efd7c
(gdb)
```

GDB4HPC

- A parallel harness and aggregator around gdb/rocgdb/cuda-gdb
- Load the `gdb4hpc` module to have `gdb4hpc` in your path and the man pages available
  - `man gdb4hpc`
  - You can also find help at the `gdb4hpc` command line by utilizing the `help` command
    - `help` will give you a list of all the commands, and you can get more help about a particular command by augmenting the help command with the command of interest.
    - Ex. `>$ help info threads` will display information on the info threads command.
- You can still debug your application at non-zero optimization levels although you might not be getting all of the information that you desire when debugging.
- gdb4hpc supports both launching and attaching
  - I mostly launch so that's what we'll do here
  - See the man pages for attach info, or for how to integrate into you batch script

LAUNCHING WITH GDB4HPC

```
launch $a{16}
```
Launch process set “a” with 16 ranks

```
--gpu
```
We want to use a GPU debugger

```
--env="MPICH_GPU_SUPPORT_ENABLED=1"
```
gdb4hpc will use your environment, but set any additional values here

```
-g "-N 2 -p <partition> --cpu-bind=<masks>"
```
Pass job launcher arguments

```
-i opt.in
```
An input file to hand to stdin

```
./faces
```
The binary to debug

Remember to use `help launch` in gdb4hpc for more info!

GDB4HPC> HELP LAUNCH

dbg all> help launch

Summary: Launch an application.
Usage: launch <app_handle> <application>
[--args="<args>" OR -a "<args>"]
[--launcher="<launcher_name>" OR -l "<launcher_name>"]
[--launcher-args="<launcher_args>" OR -g "<launcher_args>"]
[--launcher-input=<path_to_file> OR -i <path_to_file>]
[--workdir=<path> OR -d<path>]
[--env="<name=value>"]
[--qsub=<batch_template> OR -q <batch_template>]
[--sbatch=<batch_template> OR -s <batch_template>]
[--gpu]
[--gdb=<gdb_app>]
[--non-mpi]
[--debug]

AN EXAMPLE GDB4HPC LAUNCH

```
faces-tests> gdb4hpc
gdb4hpc 4.13.10 - Cray Line Mode Parallel Debugger
With Cray Comparative Debugging Technology.
Copyright 2007-2021 Hewlett Packard Enterprise Development LP.
Copyright 1996-2016 University of Queensland. All Rights Reserved.

Type "help" for a list of commands.
Type "help <cmd>" for detailed help about a command.
dbg all> launch $a{16} --gpu --env="MPICH_GPU_SUPPORT_ENABLED=1" -g "-N 2 -p bp11" -i opt.in ./faces
Starting application, please wait...
Creating MRNet communication network...
sbcast: error: No compression library available, compression disabled.
sbcast: error: No compression library available, compression disabled.
Waiting for debug servers to attach to MRNet communications network...
Timeout in 400 seconds. Please wait for the attach to complete.
Number of dbgsrvs connected: [1];  Timeout Counter: [0]
Number of dbgsrvs connected: [1];  Timeout Counter: [1]
Number of dbgsrvs connected: [16];  Timeout Counter: [0]
Finalizing setup...
Launch complete.
a{0..15}: Initial breakpoint, main at /lus/cflus02/sabbott/faces/hip/gpu_subtle/main.cpp:103
dbg all>
```

THREAD AGGREGATION IN GDB4HPC

```
a{0..15}: Initial breakpoint, main at /lus/cflus02/sabbott/faces/hip/gpu_subtle/main.cpp:103
dbg all> c
<$a>: 0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
<$a>: 8 with node rank 0 using device 0 (8 devices per node) (asked for 0)
```

```
dbg all> info thread
a{8}: Debugger error: Gdb get thread info failed.
a{0..5,7,9..10,13}: *** The application is running
a{11..12,14..15}:   Id    Frame
a{11..12,14..15}: * 1-3   "faces" (running)
a{11..12,14..15}:   4-2313 AMDGPU "faces" void gpuRun2x3<Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int, int)#1}>(Faces::share(DArray<double, 6>&)::{lambda(int, int, int, int, int)#1}, int, int, int, in
t, int) [clone .kd] () from file:///lus/cflus02/sabbott/faces/hip/gpu_subtle/faces#offset=77824&size=267392
a{11..12,14..15}:
a{6}:   Id    Frame
a{6}: * 1-3   "faces" (running)
a{6}:   4-443 AMDGPU "faces" ?? ()
a{6}:
dbg all>
```

We're in non-stop mode by default, so some threads halting doesn't necessarily stop everything

gdb4hpc tries its best to aggregate information

(but sometimes aggregation does break down)

TECHNIQUE #1
Focus on what matters

- The gdb4hpc `focus` command lets you zoom into what you care about

Focus to ranges or comma separated lists of processes

```
dbg all> focus $a{2..3}
dbg a_temp> info thread
a{2..3}:   Id    Frame
a{2..3}:   1-2   "faces" (running)
a{2..3}: * 4 3 5-197 AMDGPU "faces" Faces::share(DArray<double, 6>&)::{lambda(int, int, int, int)#2}::operator()(int, int, int, int) co
nst (this=<optimized out>, ia=<optimized out>, ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
a{2..3}:
dbg a_temp> thread 4
dbg a_temp> bt
a{2}: #1  gpuRun3x1<Faces::share at /lus/cflus02/sabbott/faces/hip/base/gpu.hpp:131
a{2}: #0  Faces::share at /lus/cflus02/sabbott/faces/hip/base/Faces.cpp:336

a{3}: #1  gpuRun3x1<Faces::share at /lus/cflus02/sabbott/faces/hip/base/gpu.hpp:131
a{3}: #0  Faces::share at /lus/cflus02/sabbott/faces/hip/base/Faces.cpp:336

dbg a_temp> focus $all
dbg all> info thread
a{0..7}:   Id    Frame
a{0..7}:   1-2   "faces" (running)
a{0..7}: * 4 3 5-197 AMDGPU "faces" Faces::share(DArray<double, 6>&)::{lambda(int, int, int, int)#2}::operator()(int, int, int, int) co
nst (this=<optimized out>, ia=<optimized out>, ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
a{0..7}: _
```

And unfocus when you're done

TECHNIQUE #2

In non-stop mode you can halt it all

```
dbg all> info thread
a{0..7}:   Id    Frame
a{0..7}:   1-2   "faces" (running)
a{0..7}: * 4 3 5-197 AMDGPU "faces" Faces::share(DArray<double, 6>&)::{lambda(int, int, int, int)#2}::operator()(int, int, int, int) co
nst (this=<optimized out>, ia=<optimized out>, ib=<optimized out>, ja=<optimized out>, jb=<optimized out>) at Faces.cpp:336
a{0..7}:
dbg all> thread 1
dbg all> info locals
a{0..7}: Debugger error: Selected thread is running.
dbg all> halt -a
a{2..4,6..7}: Halt could not report a location
a{0..1,5}: Application halted in rocr::core::InterruptSignal::WaitRelaxed
dbg all> bt
a{0..7}: #13 main at /lus/cflus02/sabbott/faces/hip/base/main.cpp:165
a{0..7}: #12 Faces::share at /lus/cflus02/sabbott/faces/hip/base/Faces.cpp:454
```

We're in non-stop mode by default, so some threads halting doesn't necessarily stop everything

You can halt individual threads or processes, or just stop it all with -a

TECHNIQUE #3

Sometimes you just need gdbmode

gdb4hpc doesn't have commands for *everything* gdb can do

We can drop to "gdbmode" to get raw access to the backends

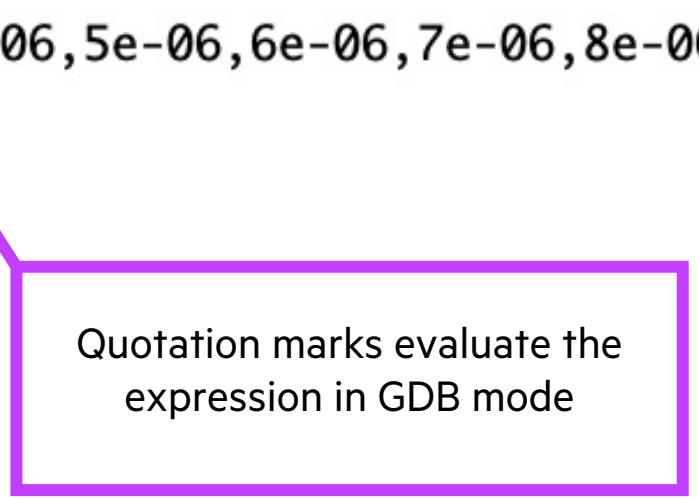
Make sure to end gdbmode before moving on!

```
dbg all> info args
Undefined info command: "args".  Try "help info".
dbg all> gdbmode
Entering gdb pass-thru mode. Type "end" to exit mode...

GNU gdb (rocm-rel-4.5-164) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
> info args

a{7}:
this = 0x7ffc338373e0
u = @0x7ffc338372d0: {strides_ = {12, 144, 1728, 25920, 363008, 4719104}, values_ = 0x
7f7530000000, first_ = 0x7ffc338372d0}

> end

Ending gdb pass-thru mode. If program location has changed (i.e. continue) debugger is
 in an unknown state.
dbg all>
```

TECHNIQUE #4

You can do mini-gdbmode inline for some things

```
dbg all> focus $a{1}
dbg a_temp> p u
a{1}: {strides_ = [12,144,1728,25920,363008,4719104], values_ = {*values_ = 14.000011}
, first_ = (DArray<double, 6> *) [1]}
dbg a_temp> p u->values_[0]@10
syntax error, unexpected INT, expecting STRING
dbg a_temp> p "u->values_[0]@10"
a{1}: [14.000011,1e-06,2e-06,3e-06,4e-06,5e-06,6e-06,7e-06,8e-06,9e-06]
dbg a_temp>
```



TECHNIQUE #5

You don't have to `focus` to focus

Use "::" operator to specify a process set as part of an expression

```
dbg all> p $a{2..3}::"u->values_[0]@10"
a{2}: [1300.0011,1300.001102,1300.001104,1300.001106,1300.001108,1300.00111,1300.00111
2,1300.001114,1300.001116,1300.001118]
a{3}: [2628.002222,1300.001102,1300.001104,1300.001106,1300.001108,1300.00111,1300.001
112,1300.001114,1300.001116,1300.001118]
dbg all>
```

DEBUGGING TAKEAWAYS

- Debugging is easy when you're introducing synthetic bugs to show off tools
- Understand what your bug *could* be before you go looking for it
- Understand what tools are at your disposal and what they can be used for
- Try to remember that every debugging session is a learning experience
  - If you knew what the bug was, you wouldn't need to debug
- GPUs are quickly becoming first class citizens in the debugging world
- There are tools we *didn't* talk about here
  - Address sanitizers (CPU and GPU)
  - Thread sanitizers
  - Visualizers

THANK YOU

Steve Abbott
stephen.abbott@hpe.com