Hewlett Packard
Enterprise

Debugging on HPE Cray Supercomputers With AMD GPUs

Mark Stock
And Steve Abbott

February 17, 2023

Outline: Debugging techniques and tools

- Lay the right groundwork *before* you encounter the bug
  - Add asserts and error code checking
- Leverage version control (you’re using it, right?)
  - Revert commits until the problem goes away – what changed since then?
- Run with smaller inputs or fewer parameters
- Understand the error message
- Set environmental variables to change execution
  - One-shot systems for serializing all GPU calls, or dumping real-time status
- (Inter)active debugging
  - Build a debug version of your code (and/or link with debug versions of libraries)
  - Run gdb / rocgdb / kokkosgdb / gdb4hpc / ccdb / DDT / Totalview, etc.
  - Examine a core file, attach to a running process, or interactively probe execution

Before the Bug:
Best Practices

Public domain

Set yourself up for success - Asserts

- Compile-time
  - Save yourself from problematic variable or type usage
  - Zero performance hit at runtime
  - `std::static_assert( sizeof(long) == 8, “Require long int to be 8 bytes” );`
- Run-time asserts
  - In C++
    ```
    #include <assert>
    assert(neverIsZero == 0 && “Somehow it equals zero”);
    ```
    And define `NDEBUG` during compilation to turn all of them off
  - In Fortran
    ```
    if (neverIsZero .eq. 0) stop “Somehow it equals zero”
    ```
  - If used outside of inner loops, performance hit is minimal
  - Runtime asserts in GPU code may impact performance

Set yourself up for success - Builds

- Build with `–g` or `–ggdb`
- Also consider `–v`
- CCE users can see the underlying command with `–craype–verbose`
- CMake users
  - Set `CMAKE_BUILD_TYPE=RelWithDebInfo`
  - Build with `VERBOSE=1 make`
- Note that the optimization level has a strong influence on many bugs
  - -O0 code is larger & uses more resources – BUT necessary for deep GPU debugging
  - -O1 makes good debug builds for CPU codes
  - -O2 is well-optimized, and the default for CMake, though many temporaries lost
  - -G2 for OpenMP+Offload with ftn

Set yourself up for success - Check error codes!!

- If library authors went to the trouble of returning error codes, you should check them!
- Drop-in macro is all you need:

```
#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(); }}
```

- Then wrap your synchronous HIP API calls with this:

```
HIP_RC( hipMalloc(&d_A, N * sizeof(int)) );
```

- Gives nice errors on failure:

```
simple_hmm.c:21 -- hipMalloc(&d_A, N * sizeof(int)) returned 2:hipErrorOutOfMemory
```

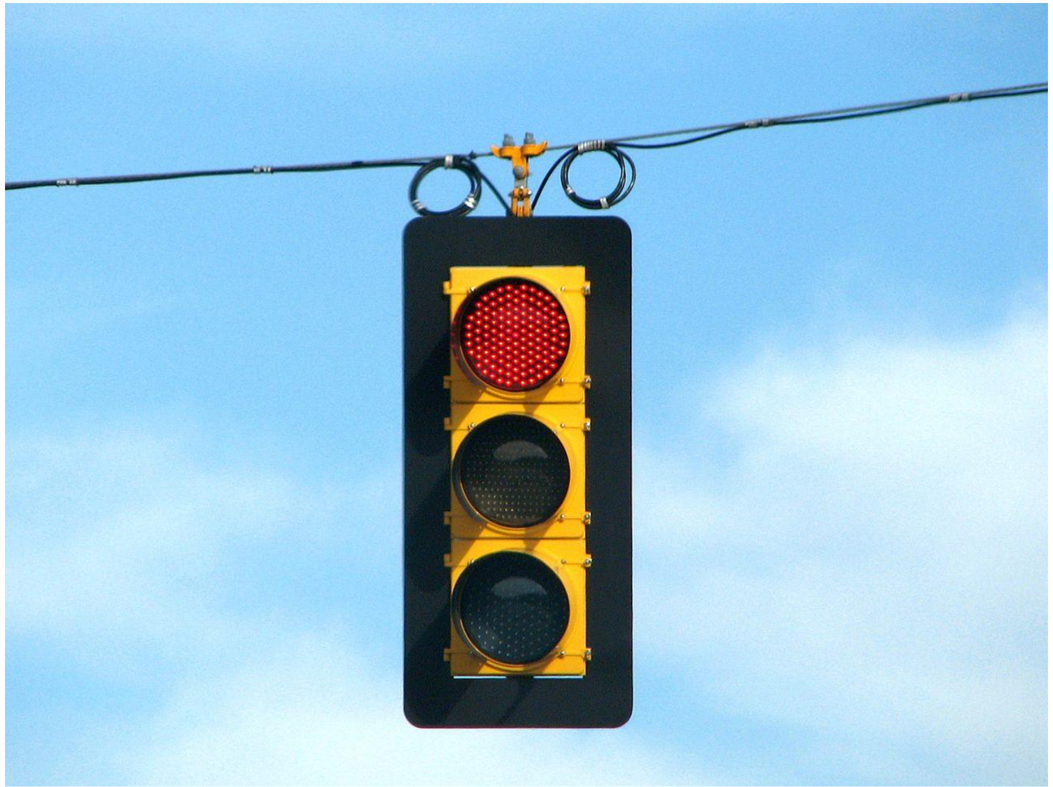
Check asynchronous calls, too

- Asynchronous calls like kernel launches don’t immediately emit error codes
- Use the same macro to check a subsequent (synchronous) call:

```
// launch kernel
hipLaunchKernelGGL(vector_addition, blk_in_grid, thr_per_blk , 0, 0, d_A, d_B, d_C);

// check for synchronous errors during kernel launch (like invalid execution params)
HIP_RC( hipGetLastError() );

// check for *asynchronous* errors during kernel execution
HIP_RC( hipStreamSynchronize(mystream) );
HIP_RC( hipDeviceSynchronize() );
```

- Note that these last two will wait on all previous async calls, so may suppress your bug!

Reading a Crash:
SIGSEGV Means Something



K Payravi

Reading OS signals

- Compiler writers are succinct and often precise

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

- See man 7 signal for a quick guide

Common error messages related to AMD GPUs

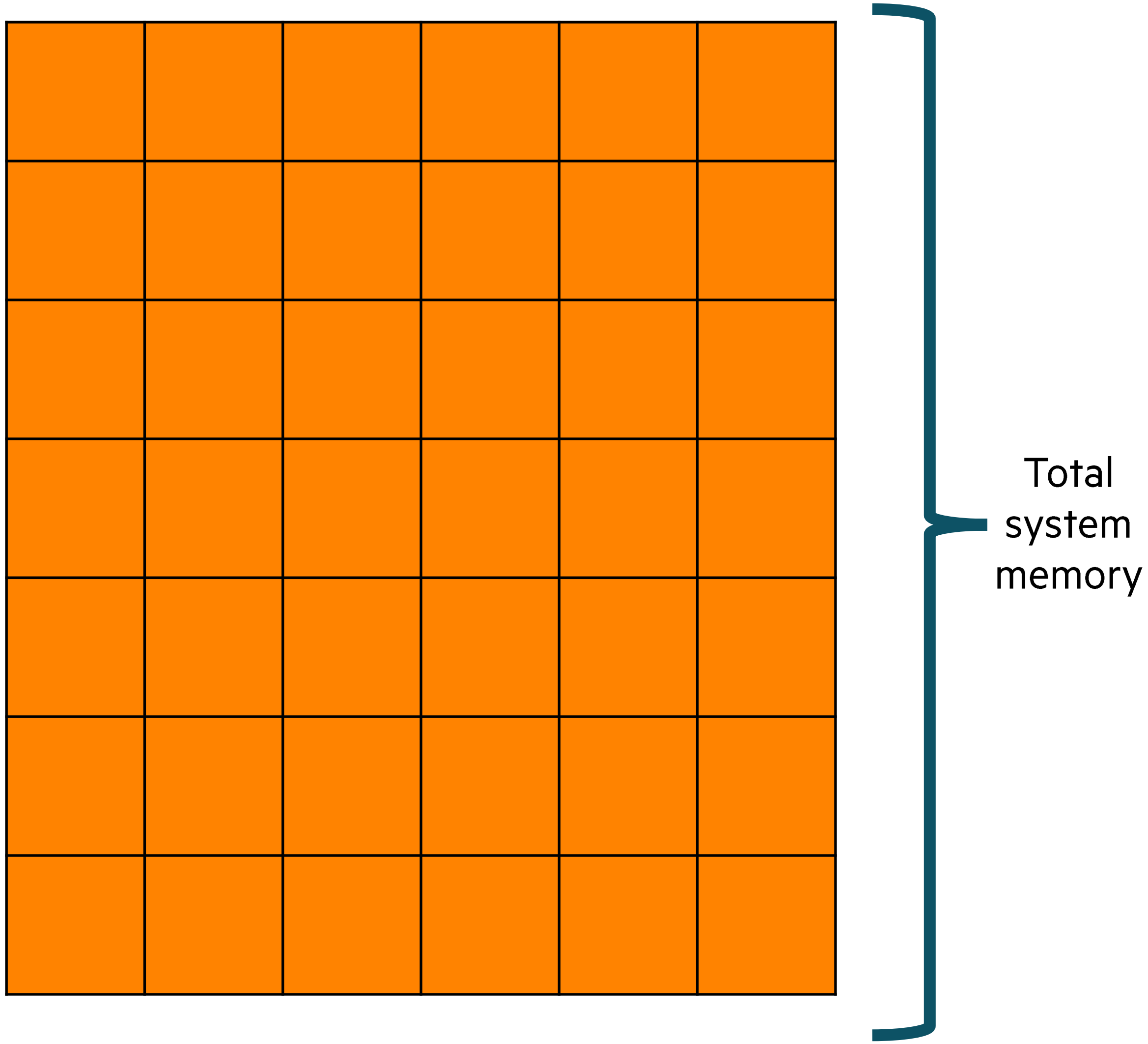
- When an error is hit on the GPU it raises an exception
- The runtime will map the exception to the analogous signal and drop it

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

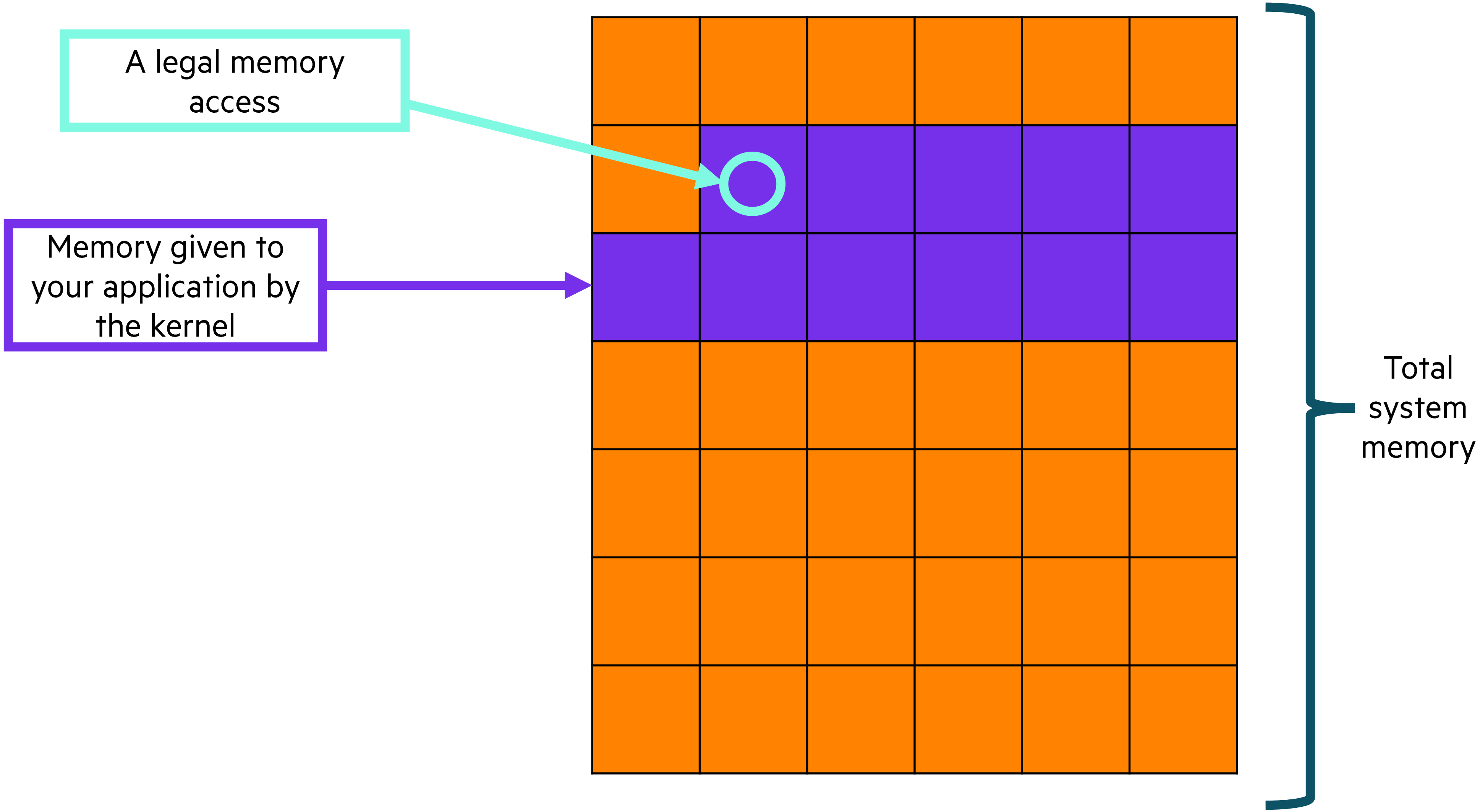
* HSA errors will be prefaced by something like:

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

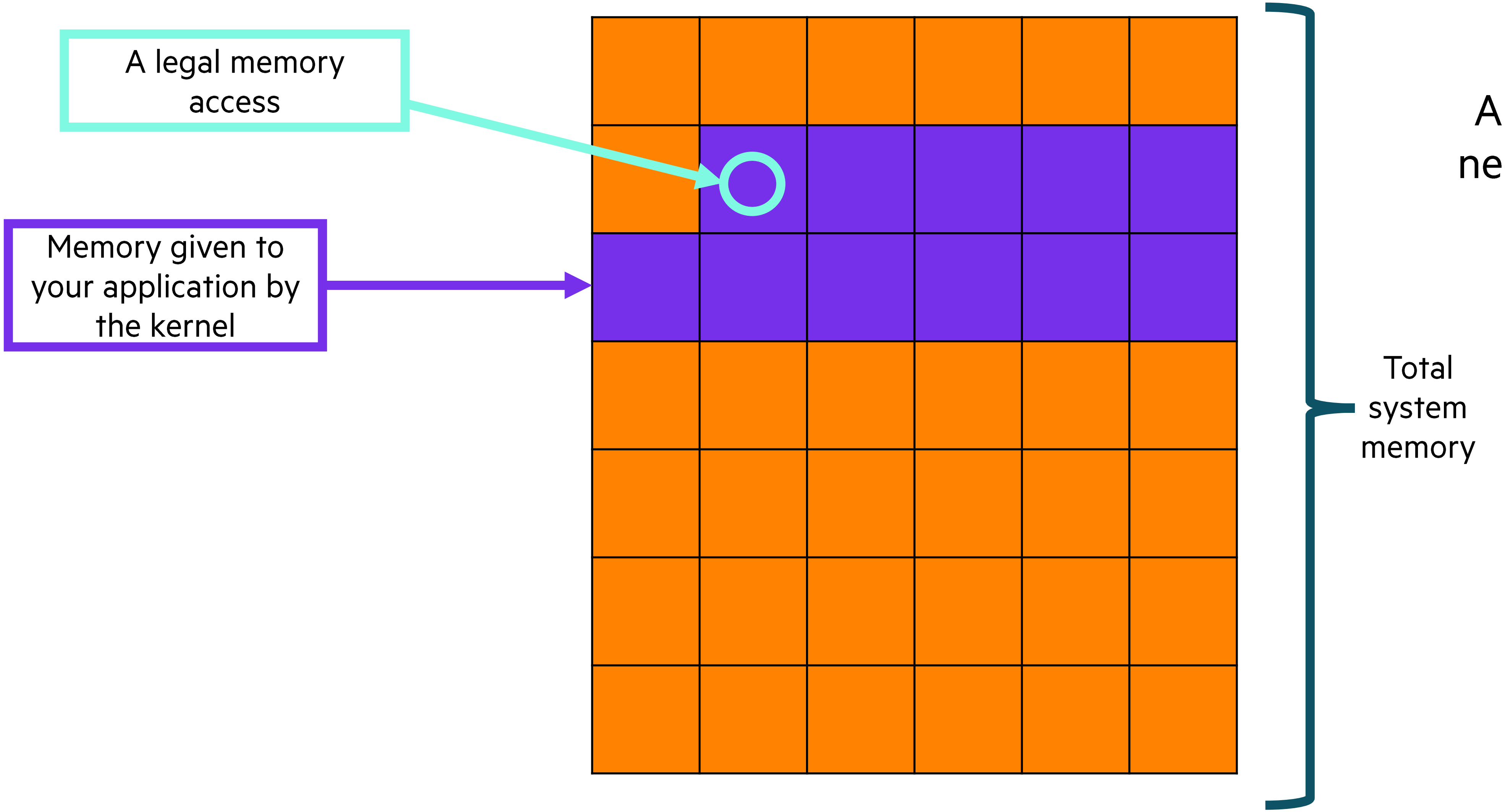
Segfault/SIGBUS visualized



Segfault/SIGBUS visualized

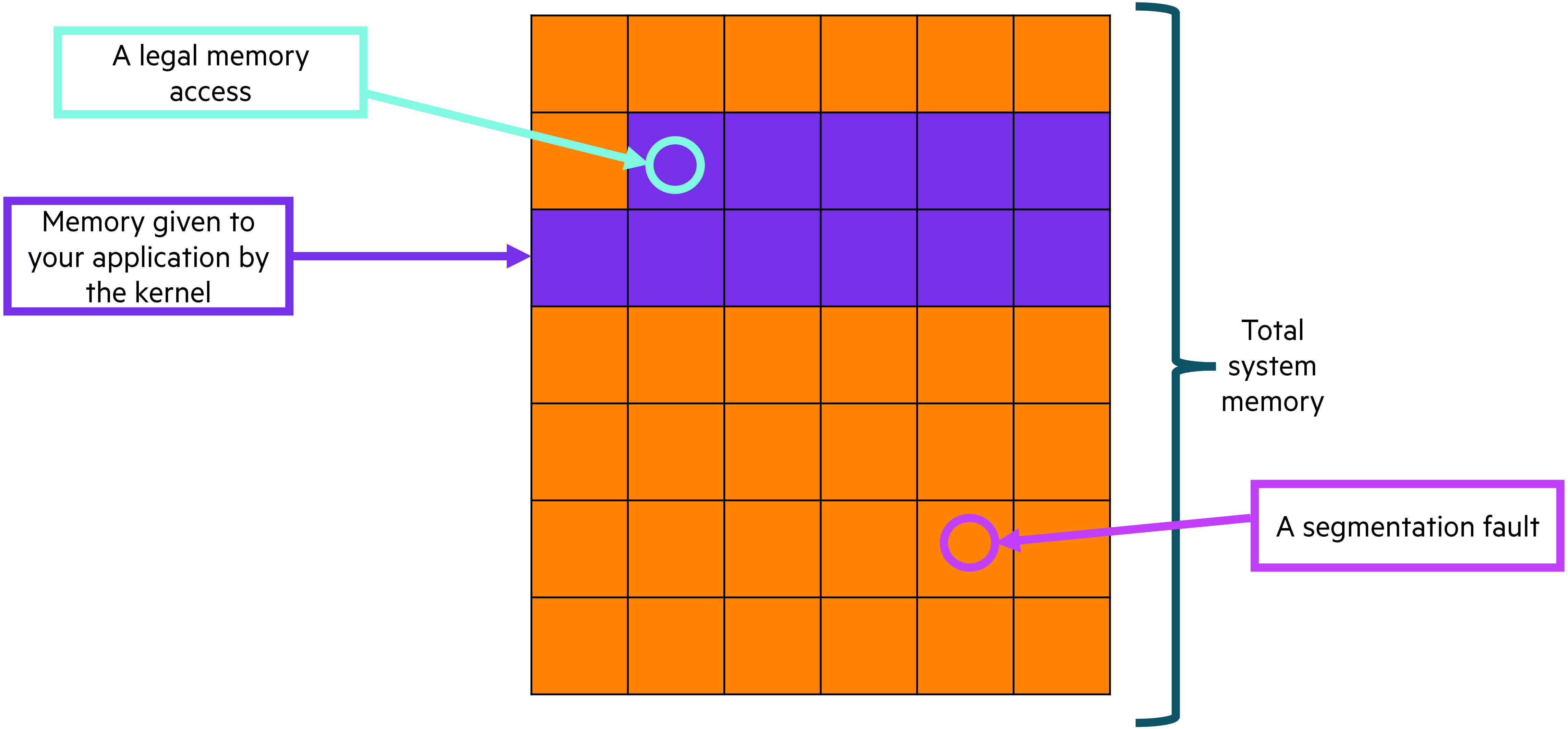


Segfault/SIGBUS visualized

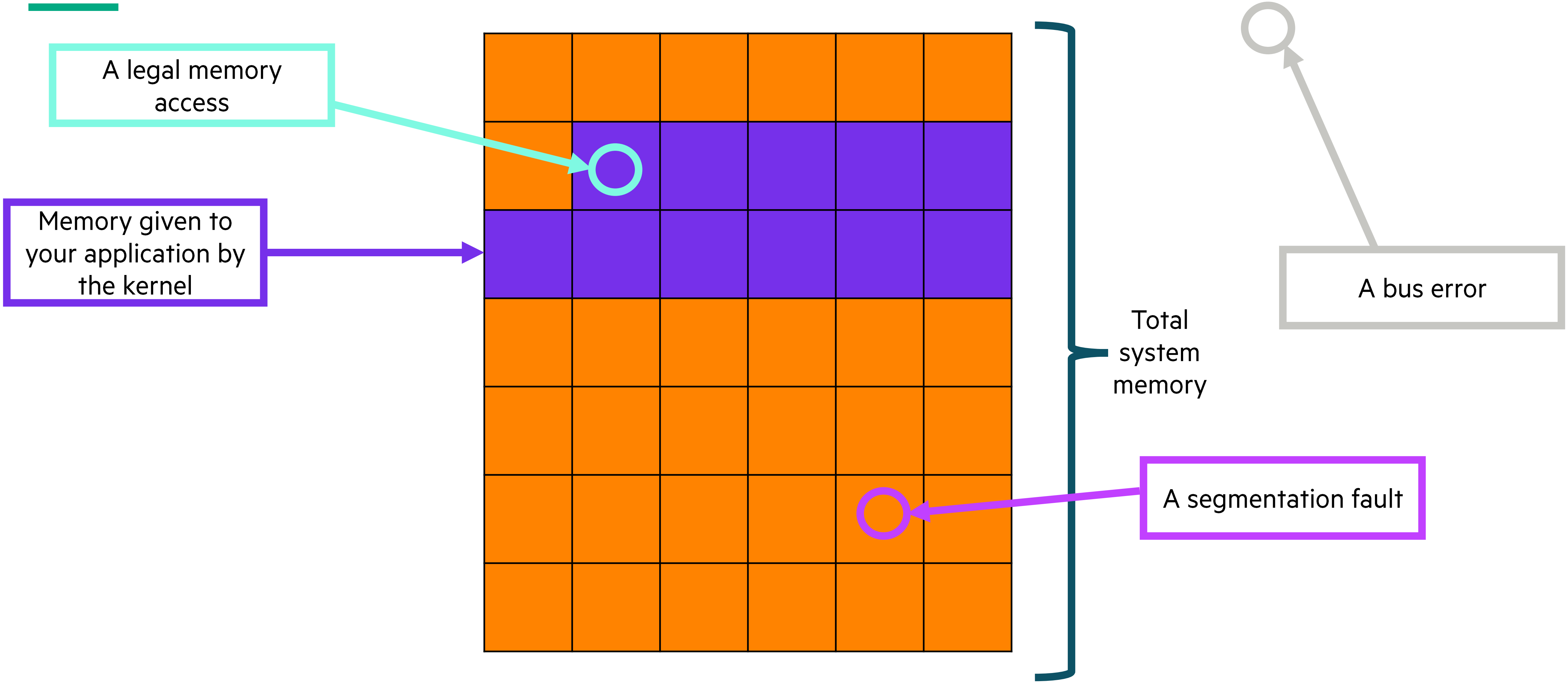


A legal access isn't necessarily a correct access!

Segfault/SIGBUS visualized



Segfault/SIGBUS visualized



Sipping From the Firehose: Runtime Debug Information

U.S. Bureau of Reclamation

The Cray OpenMP Offload runtime

- The Cray OpenMP (cc, CC, ftn) and OpenACC (ftn) runtimes will print debug information to stderr on demand; here's how to control the level of output

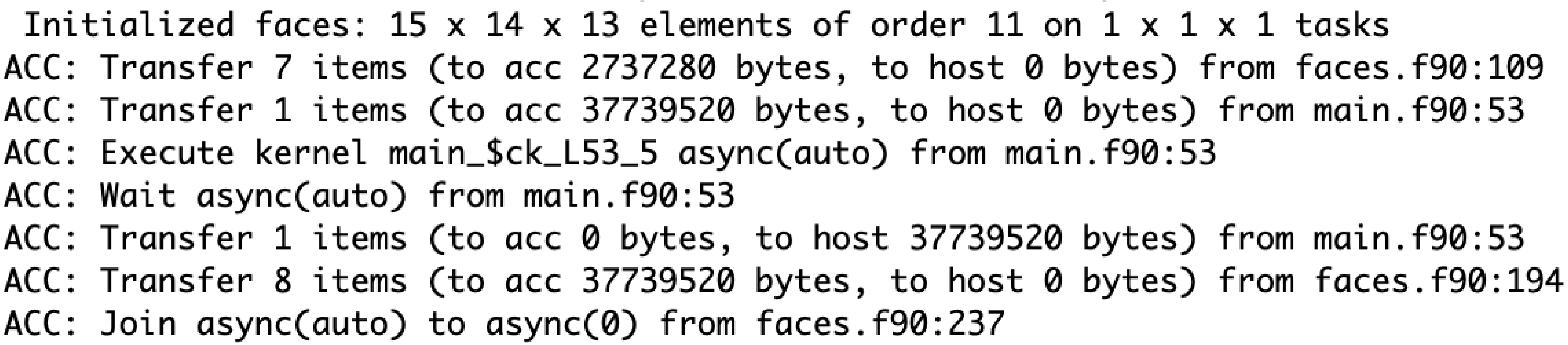
- **CRAY_ACC_DEBUG=1**
  - Concise, a good way to tell your offload regions are running
  - Probably not useful for more complex debugging
- **CRAY_ACC_DEBUG=2**
  - Designed to be user friendly and where you should start
  - Shows what the runtime is doing but not nitty gritty details
- **CRAY_ACC_DEBUG=3**
  - 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

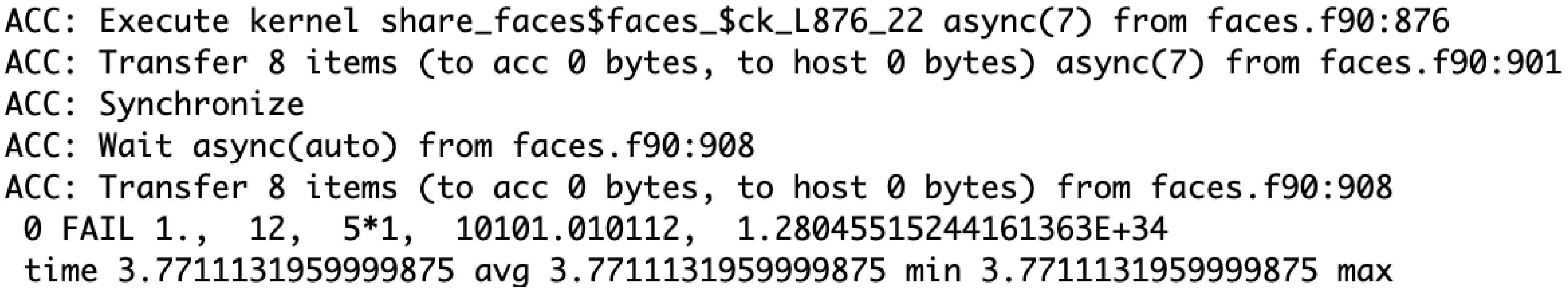
```
faces-tests> MPICH_GPU_SUPPORT_ENABLED=1 CRAY_ACC_DEBUG=0 srun -u -n 1 -N 1 -c 1 --
pty --exclusive ./faces-mi200 < opt.in &

&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,  1.2804551524416136E+34
 time 3.6951122709999922 avg 3.6951122709999922 min 3.6951122709999922 max
```

What went wrong?

With CRAY_ACC_DEBUG=1

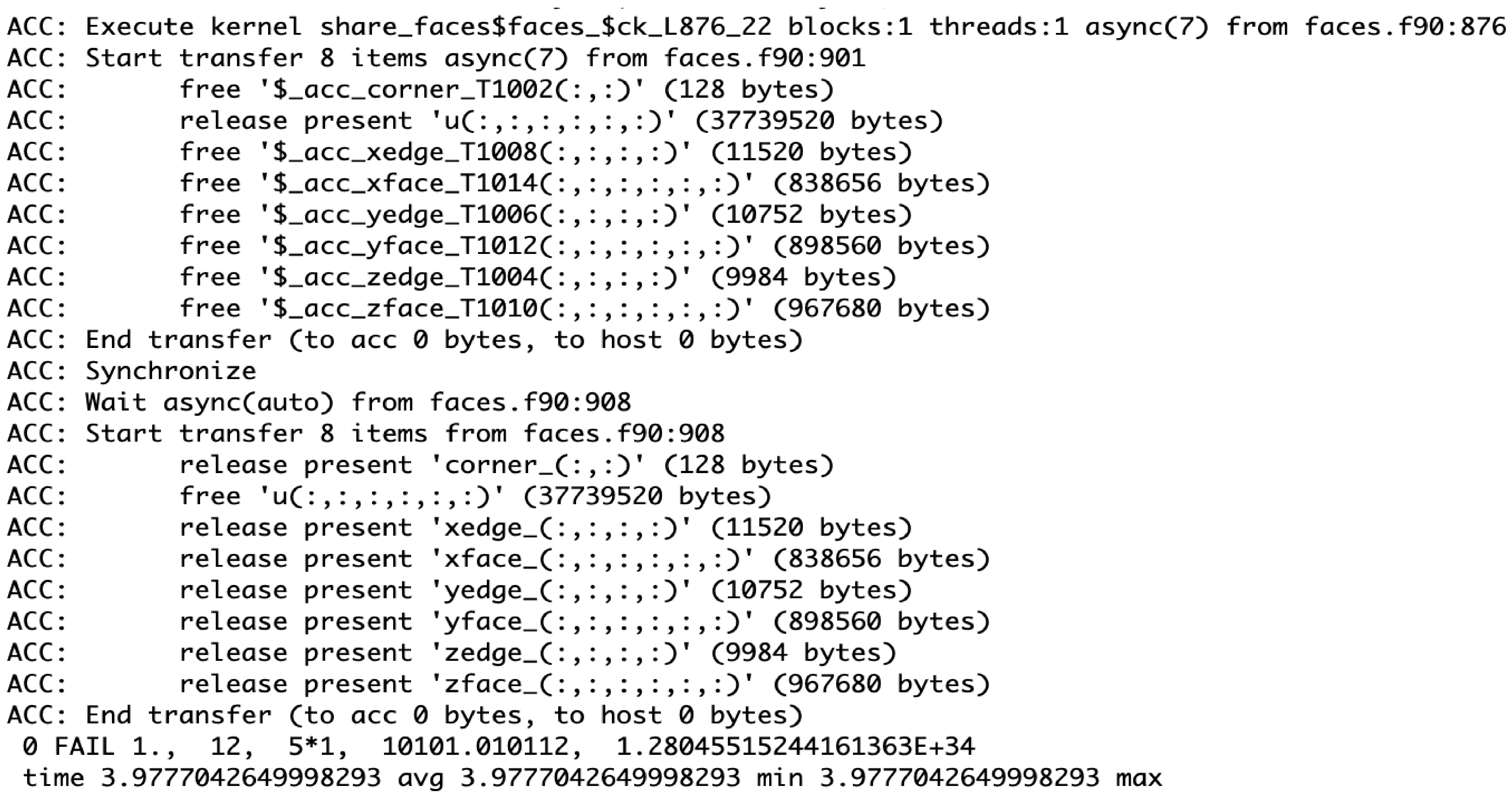


```
 Initialized faces: 15 x 14 x 13 elements of order 11 on 1 x 1 x 1 tasks
ACC: Transfer 7 items (to acc 2737280 bytes, to host 0 bytes) from faces.f90:109
ACC: Transfer 1 items (to acc 37739520 bytes, to host 0 bytes) from main.f90:53
ACC: Execute kernel main_$ck_L53_5 async(auto) from main.f90:53
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,  1.2804551524416136E+34
 time 3.7711131959999875 avg 3.7711131959999875 min 3.7711131959999875 max
```

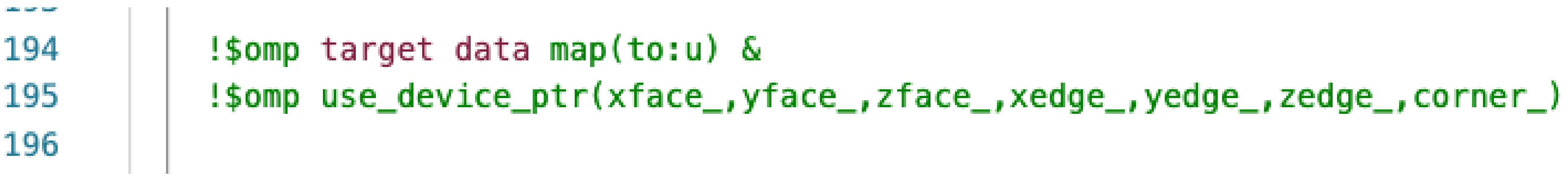
With CRAY_ACC_DEBUG=2



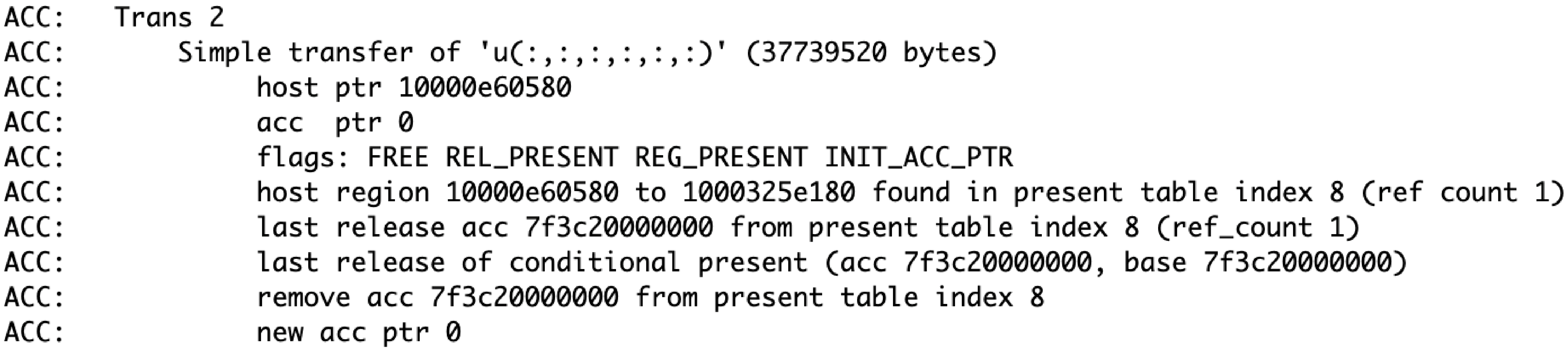
```
ACC: Execute kernel share_faces$faces_$ck_L876_22 blocks:1 threads:1 async(7) from faces.f90:876
ACC: Start transfer 8 items async(7) from faces.f90:901
ACC:       free '$_acc_corner_T1002(:,:)' (128 bytes)
ACC:       release present 'u(:,:,:,:,:,:)' (37739520 bytes)
ACC:       free '$_acc_xedge_T1008(:,:,:,:)' (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,  1.2804551524416136E+34
 time 3.9777042649998293 avg 3.9777042649998293 min 3.9777042649998293 max
```

With CRAY_ACC_DEBUG=3

We should probably copy back that state vector…



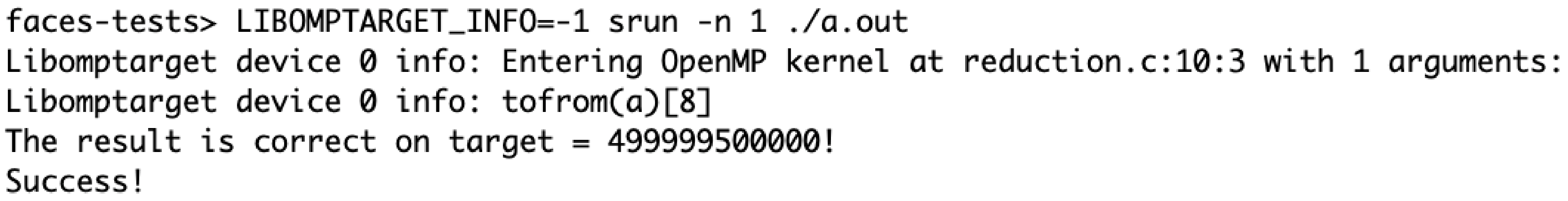
```
  !$omp target data map(to:u) &
  !$omp use_device_ptr(xface_,yface_,zface_,xedge_,yedge_,zedge_,corner_)

```



```
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
```

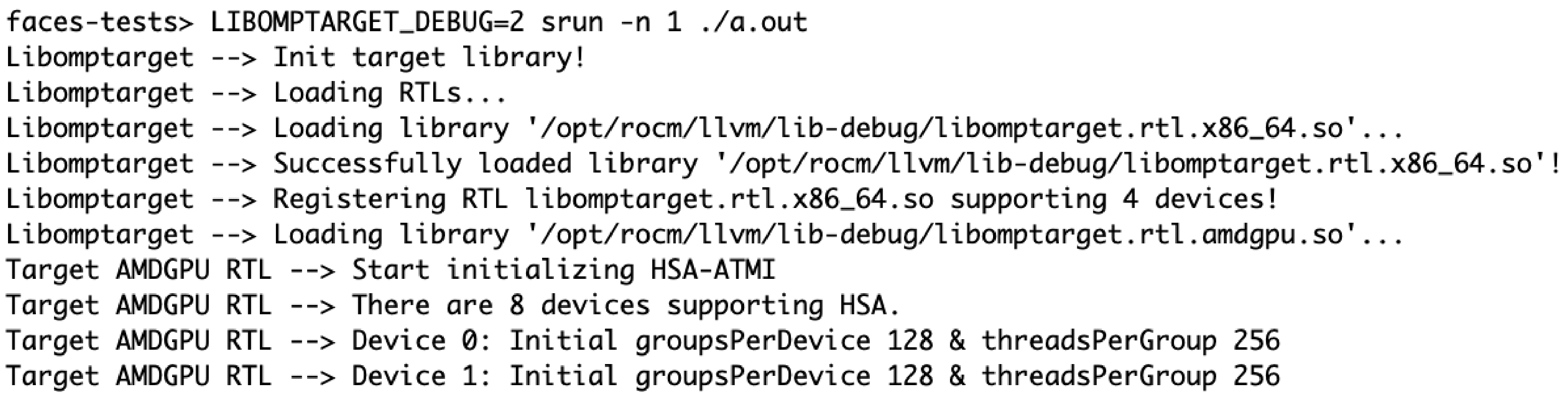
The AMD OpenMP Offload runtime

- 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 also `LIBOMPTARGET_DEBUG`, but that may be too much!
  - If you really need a debug compiler, there's a build in `${ROCM_PATH}/llvm/lib-debug`

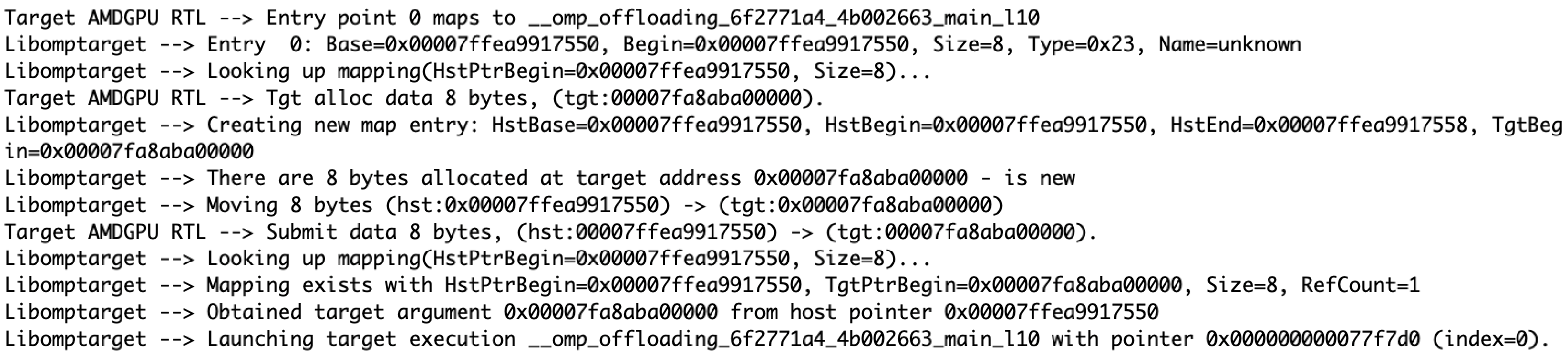


```
faces-tests> LIBOMPTARGET_INFO=-1 srun -n 1 ./a.out
Libomptarget device 0 info: Entering OpenMP kernel at reduction.c:10:3 with 1 arguments:
Libomptarget device 0 info: tofrom(a)[8]
The result is correct on target = 499999500000!
Success!
```

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
```

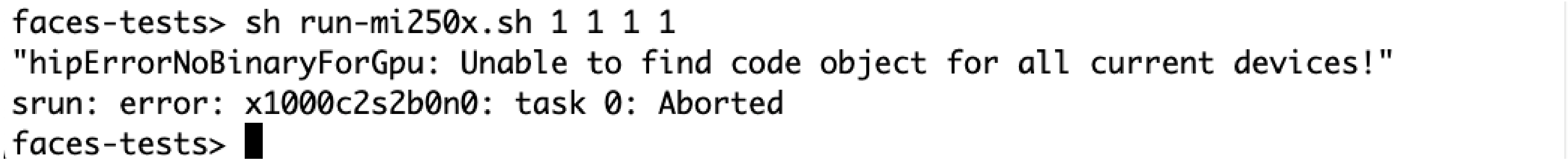


```
Target AMDGPU RTL --> Entry point 0 maps to __omp_offloading_6f2771a4_4b002663_main_l10
Libomptarget --> Entry  0: Base=0x00007ffea9917550, Begin=0x00007ffea9917550, Size=8, Type=0x23, Name=unknown
Libomptarget --> Looking up mapping(HstPtrBegin=0x00007ffea9917550, Size=8)...
Target AMDGPU RTL --> 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).
```

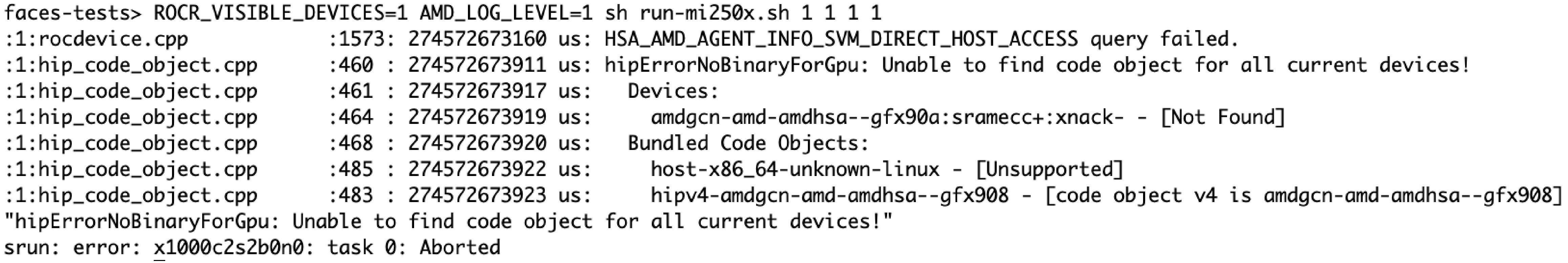
AMD HIP and HSA runtimes

- 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
  - 7 – only a lumberjack wants these logs
- You can fine tune *what* gets logged with `AMD_LOG_MASK`
  - Details at https://docs.amd.com/bundle/AMD_HIP_Programming_Guide/page/Programming_with_HIP.html

An example where AMD_LOG_LEVEL helps a lot

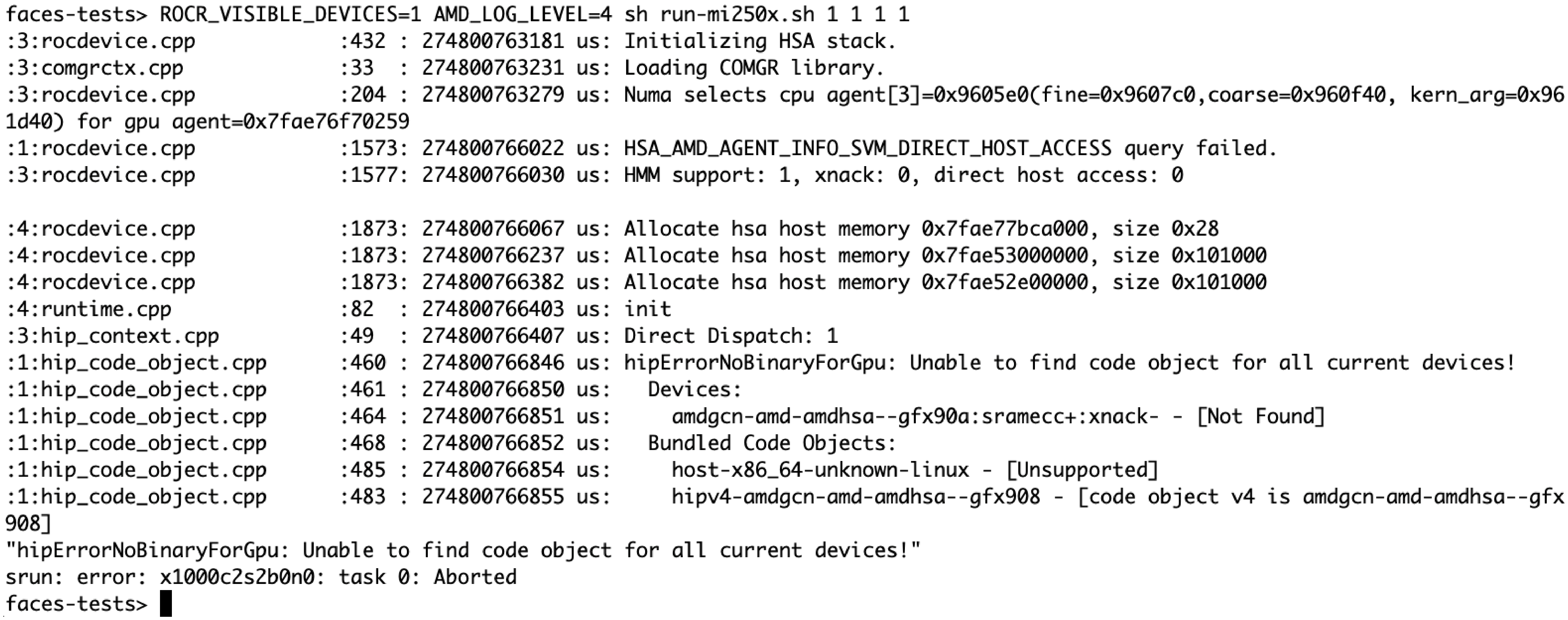


```
faces-tests> sh run-mi250x.sh 1 1 1 1
"hipErrorNoBinaryForGpu: Unable to find code object for all current devices!"
srun: error: x1000c2s2b0n0: task 0: Aborted
faces-tests> █
```



```
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!



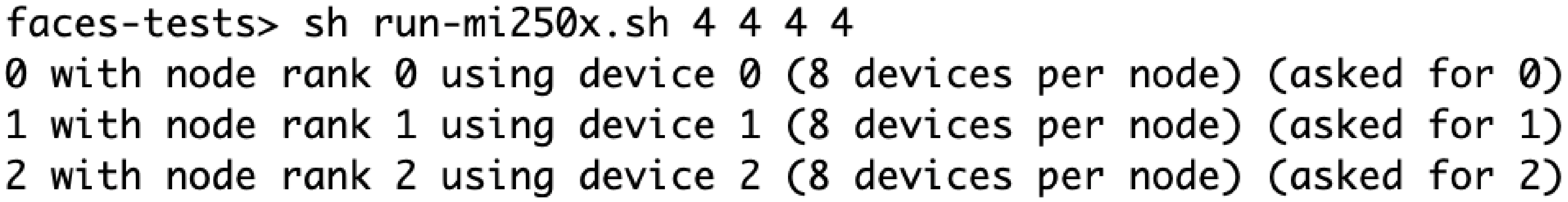
```
faces-tests> ROCR_VISIBLE_DEVICES=1 AMD_LOG_LEVEL=4 sh run-mi250x.sh 1 1 1 1
:3:rocdevice.cpp            :432 : 274800763181 us: Initializing HSA stack.
:3:comgrctx.cpp             :33  : 274800763231 us: Loading COMGR library.
:3:rocdevice.cpp            :204 : 274800763279 us: Numa selects cpu 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

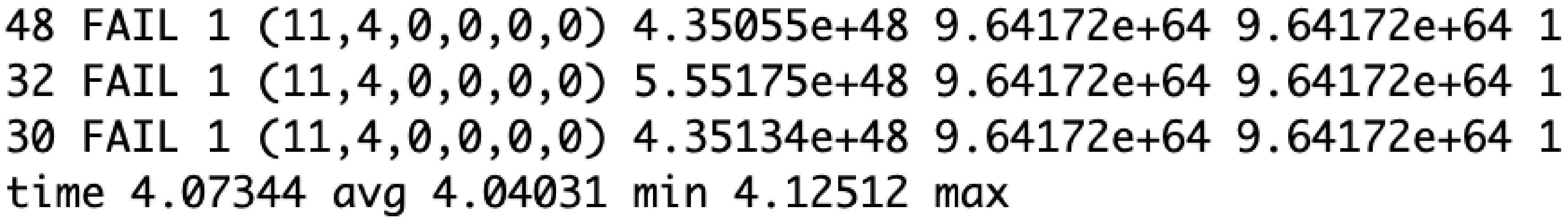
Good for race conditions, and when you need to slow things down

- `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



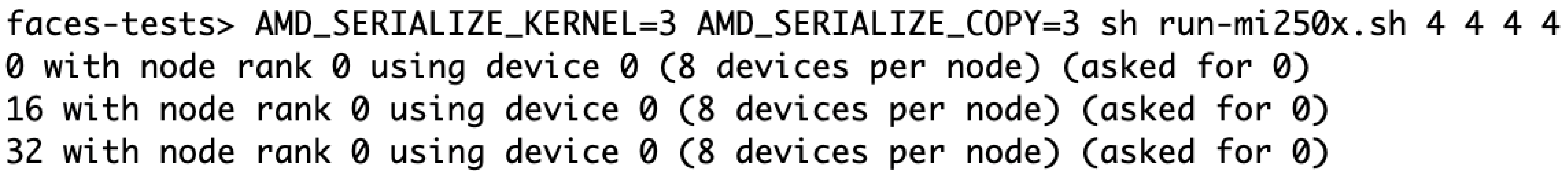
```
faces-tests> sh run-mi250x.sh 4 4 4 4
0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
1 with node rank 1 using device 1 (8 devices per node) (asked for 1)
2 with node rank 2 using device 2 (8 devices per node) (asked for 2)
```



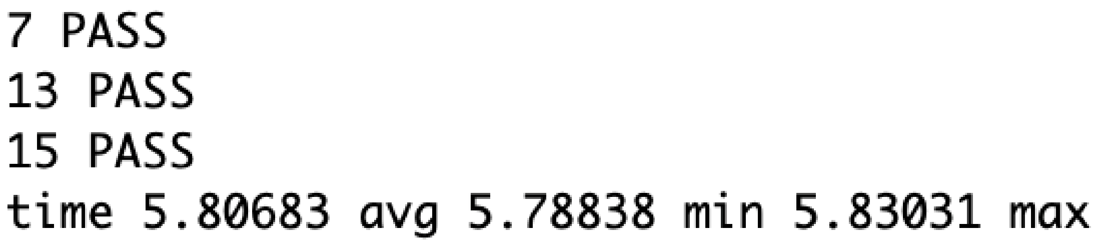
```
48 FAIL 1 (11,4,0,0,0,0) 4.35055e+48 9.64172e+64 9.64172e+64 1
32 FAIL 1 (11,4,0,0,0,0) 5.55175e+48 9.64172e+64 9.64172e+64 1
30 FAIL 1 (11,4,0,0,0,0) 4.35134e+48 9.64172e+64 9.64172e+64 1
time 4.07344 avg 4.04031 min 4.12512 max
```

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)
```

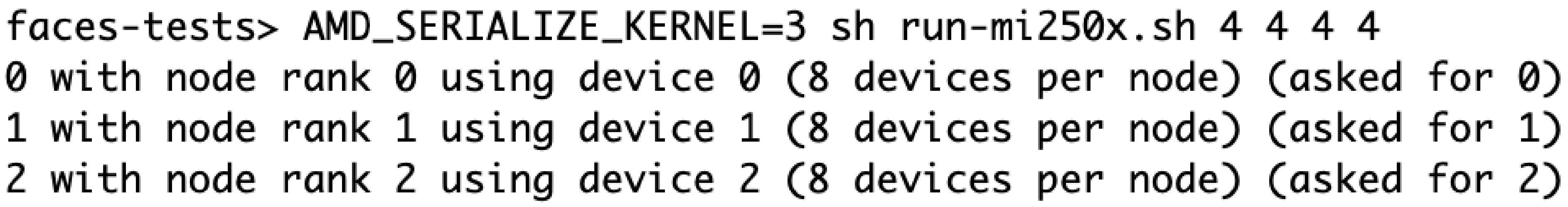


```
7 PASS
13 PASS
15 PASS
time 5.80683 avg 5.78838 min 5.83031 max
```

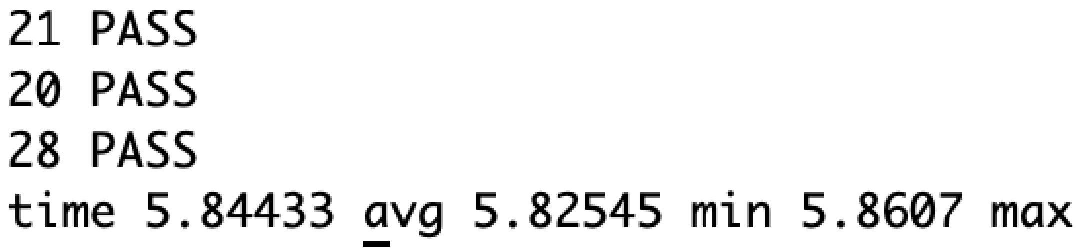
This is correct, so we probably have some race involving the GPU.

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

Check only 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)
```

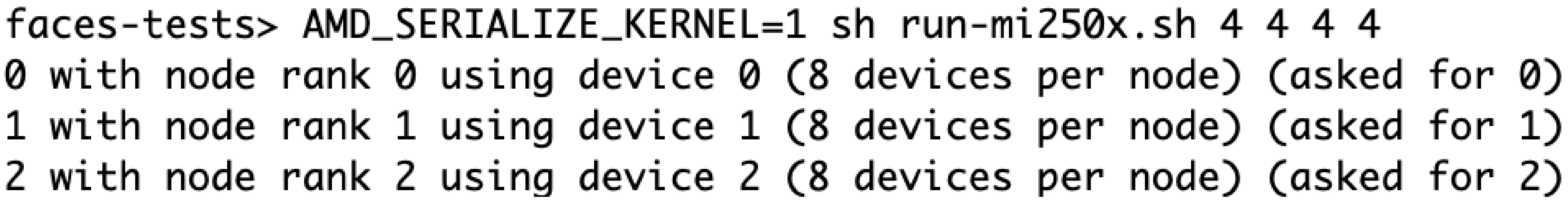


```
21 PASS
20 PASS
28 PASS
time 5.84433 avg 5.82545 min 5.8607 max
```

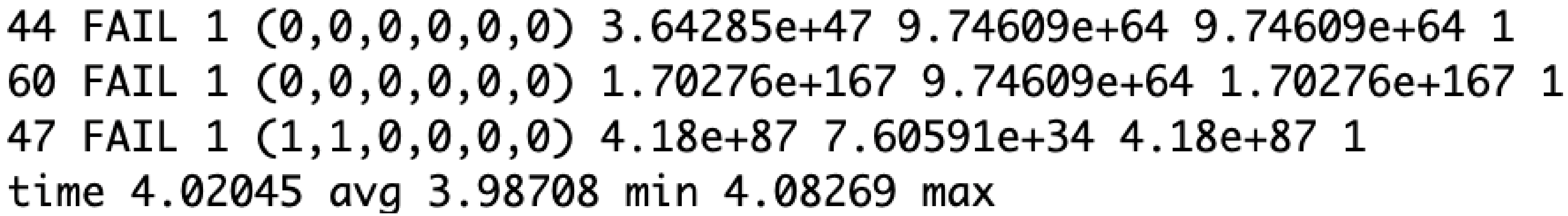
We are probably missing a synch between two kernels or between the host and a kernel.

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)
```

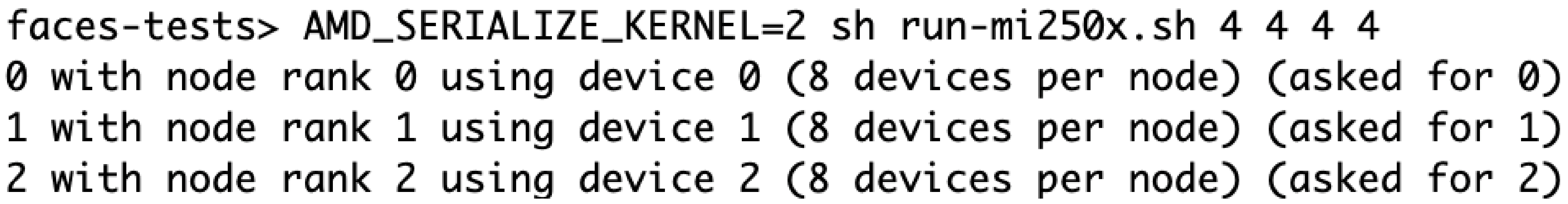


```
44 FAIL 1 (0,0,0,0,0,0) 3.64285e+47 9.74609e+64 9.74609e+64 1
60 FAIL 1 (0,0,0,0,0,0) 1.70276e+167 9.74609e+64 1.70276e+167 1
47 FAIL 1 (1,1,0,0,0,0) 4.18e+87 7.60591e+34 4.18e+87 1
time 4.02045 avg 3.98708 min 4.08269 max
```

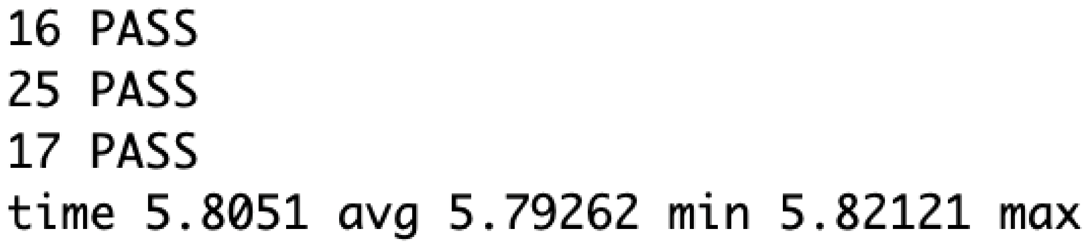
This still fails.

We are probably *not* having two kernels racing.

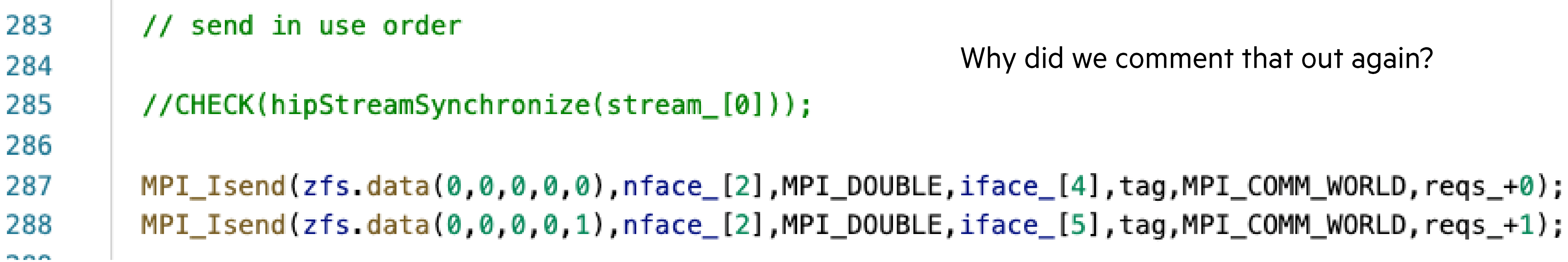
Synchronize *after* kernel launches



```
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
```



```
// send in use order

//CHECK(hipStreamSynchronize(stream_[0]));

MPI_Isend(zfs.data(0,0,0,0,0),nface_[2],MPI_DOUBLE,iface_[4],tag,MPI_COMM_WORLD,reqs_+0);
MPI_Isend(zfs.data(0,0,0,0,1),nface_[2],MPI_DOUBLE,iface_[5],tag,MPI_COMM_WORLD,reqs_+1);
```

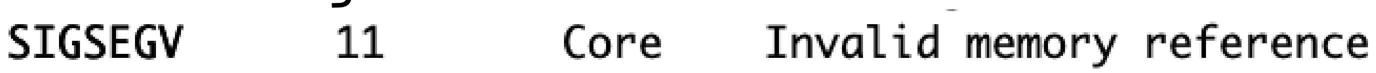
Why did we comment that out again?

Searching the Warehouse: Sifting Through Core Dumps

Axisadman, CC-BY-SA-3.0

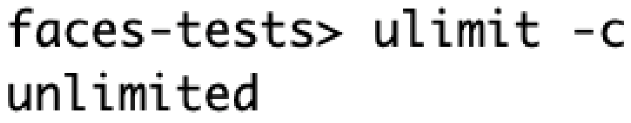
Core files for post-mortem analysis

- Most crashing signals will drop a (large) core file containing the process memory
- See `man 7 signal` for tables



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

- Your user limits need to allow core files



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

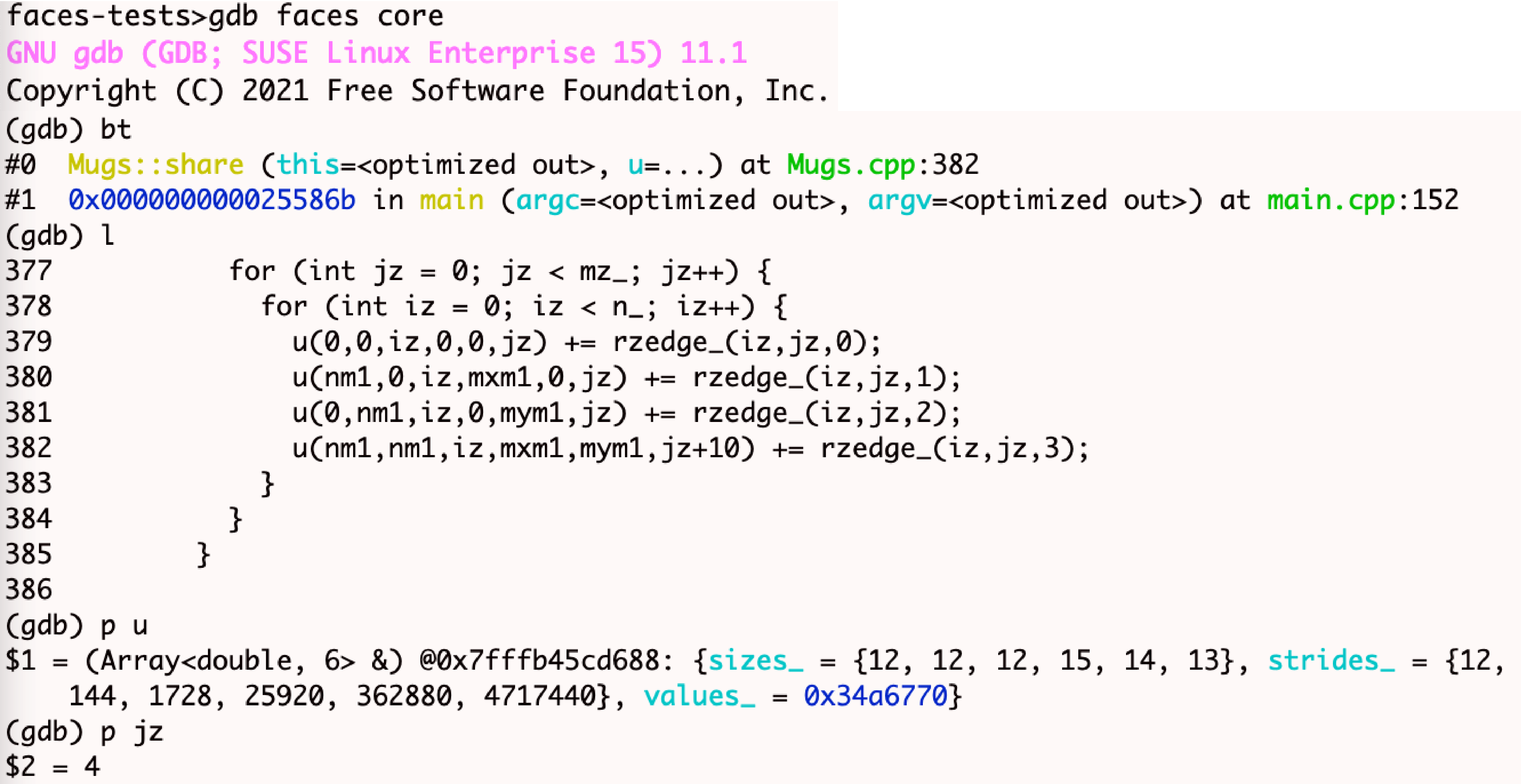
- Start a debug session with

```
gdb –core core
```

Limitations of core dumps

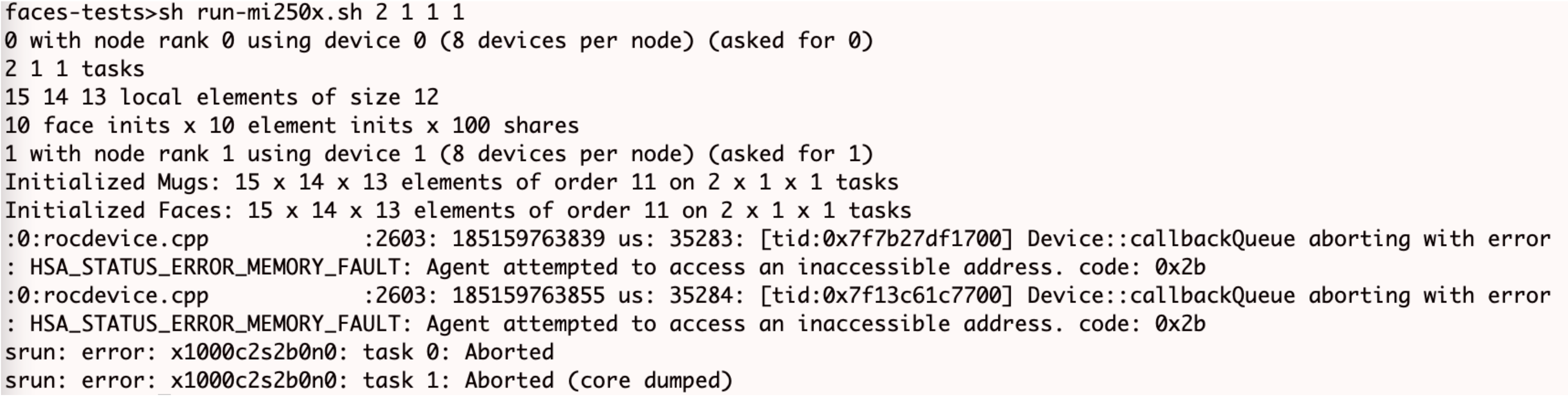
- Are the size of the process's occupied CPU memory
- Depending on system will either:
  - Only dump one core file -> maybe not enough information
  - Dump one core file for every failing process -> takes up *a lot* of space and is *slow*
- Don't contain AMD GPU memory state
- Are only postmortem

Loading a core from a CPU crash



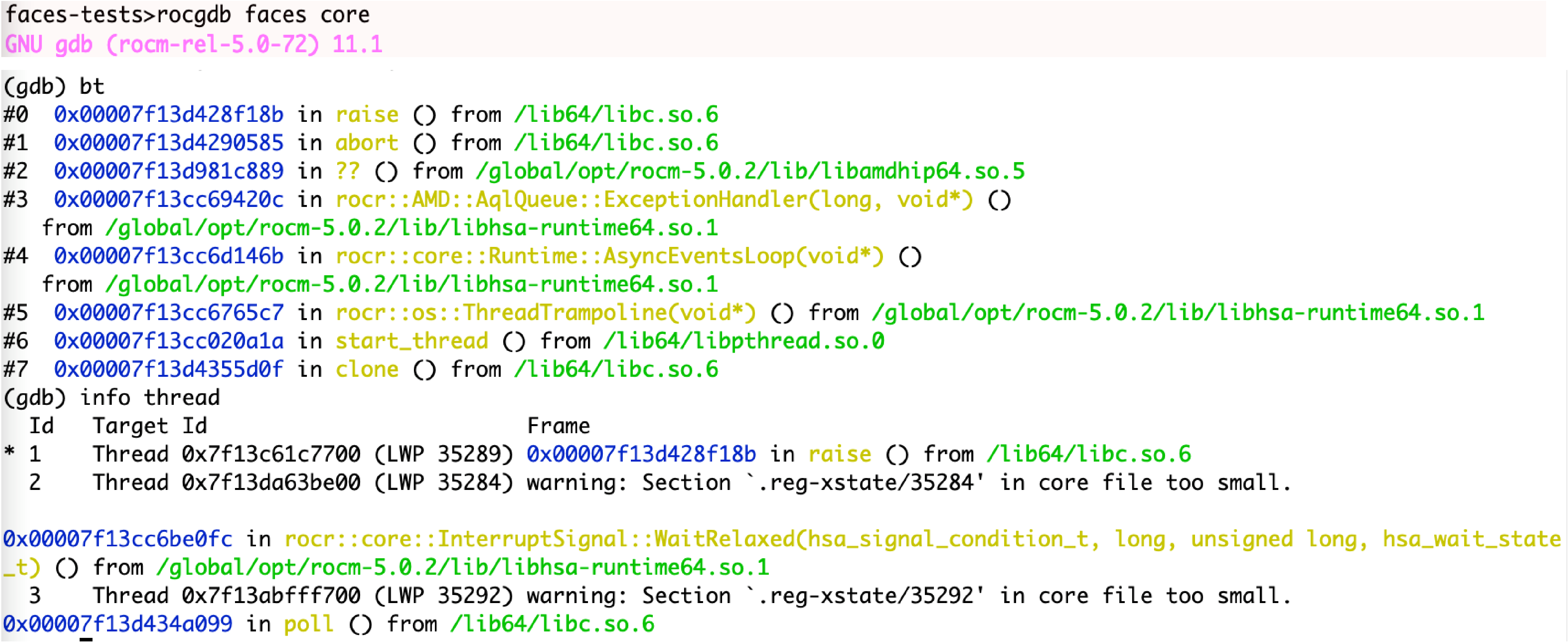
```
faces-tests>gdb faces core
GNU gdb (GDB; SUSE Linux Enterprise 15) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
(gdb) bt
#0  Mugs::share (this=<optimized out>, u=...) at Mugs.cpp:382
#1  0x000000000025586b in main (argc=<optimized out>, argv=<optimized out>) at main.cpp:152
(gdb) l
377           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)
```

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!

Interactive Debugging: Tools and Practice

Henk Monster, CCA3.0

gdb

- The “Gnu Debugger” helps locate the source of problems during CPU execution
  - Run it on a binary until the crash (`gdb mybinary`)
  - Set breakpoints and step through critical code
  - Attach to an already-running process (`gdb -pid` *`PID`*)
  - Investigate a core dump (`gdb -core core` , typically)
- Typical debugging session:

1.	`srun … --pty`	will get you onto the compute node
2.	`gdb ./mybinary`	runs gdb, inherits environment
3.	`break filename.cpp:124`	(optional) sets a break point
4.	`run [your command-line options]`	begins execution of your program
5.	*CRASH*, or breakpoint reached, gdb will go to the appropriate thread	
6.	`info threads`	see the top of each thread’s call stack, * marks the current thread
7.	`t 4`	jump to thread 4
8.	`bt`	“backtrace”: examine the stack, with summary
9.	`f 5`	jump to frame 5 in the call stack
10.	`info locals`	look at local variables and their values
11.	`print x[7]`	print the value of an array element

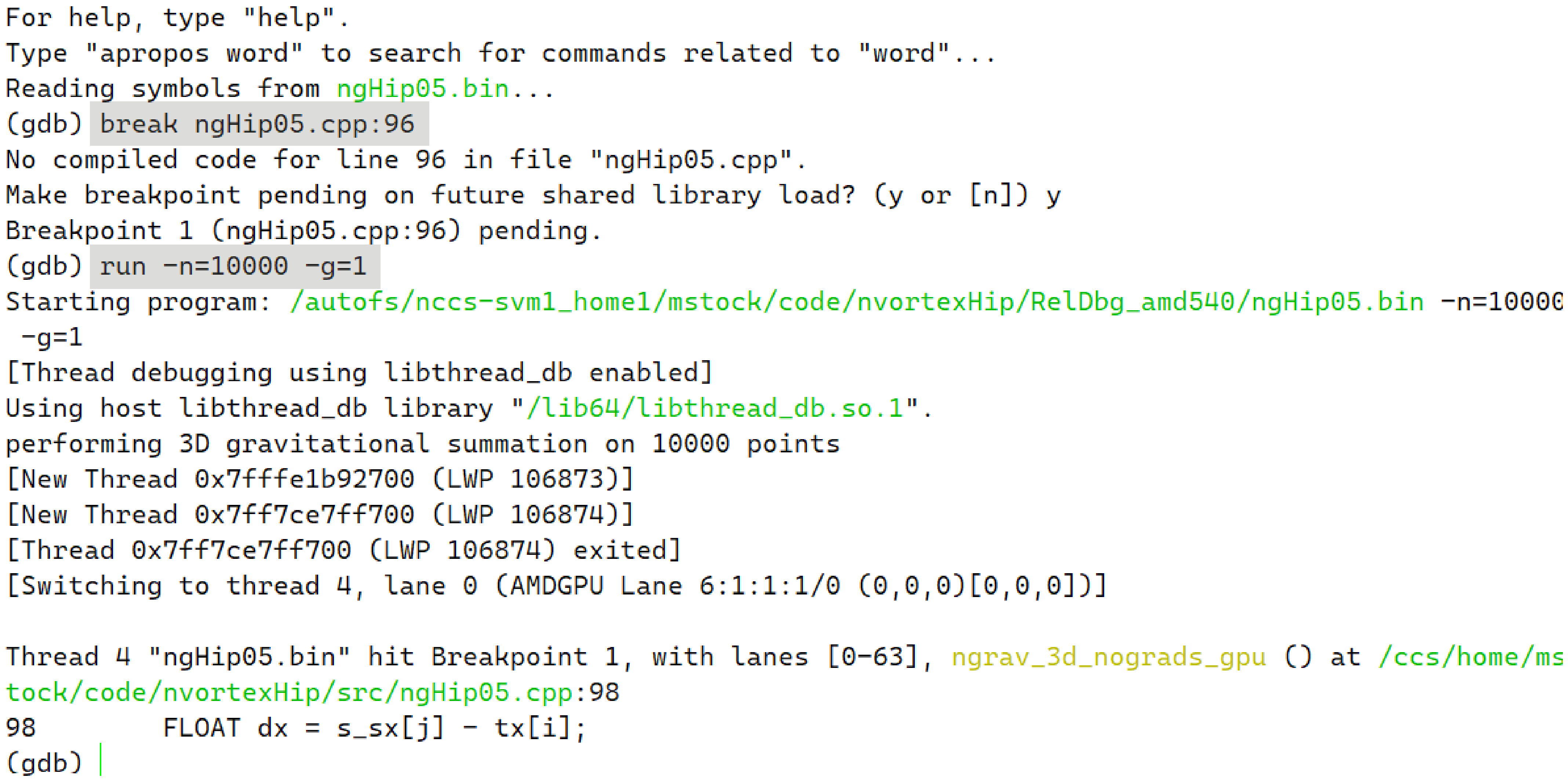
rocgdb

- Rocgdb is just gdb extended to debug HIP programs on AMD GPUs
- Notable enhancements or changes to standard gdb:
  - Each wavefront is represented as a single thread
  - A thread will have 64 lanes (like a CPU thread with 4 doubles in a SIMD register)
  - Non-stop mode works across both CPU and GPU
- It has some shortcomings:
  - To get locals you need to compile with no optimization `–O0`
  - It’s not multiprocess (or not more than gdb is)
  - The debugger version requires the driver version match for GPU debugging (so use rocm 5.3.0, not 5.4.0)
  - The native thread representation can get a bit overwhelming
- Documentation is sections 20 and 22.4.10 in rocgdb manual
  - On your system: `${ROCM_PATH}/share/doc/rocgdb/`
  - Online: https://docs.amd.com/bundle/ROCDebugger-User-Guide-v5.4/page/AMD-GPU.html

rocgdb - Details

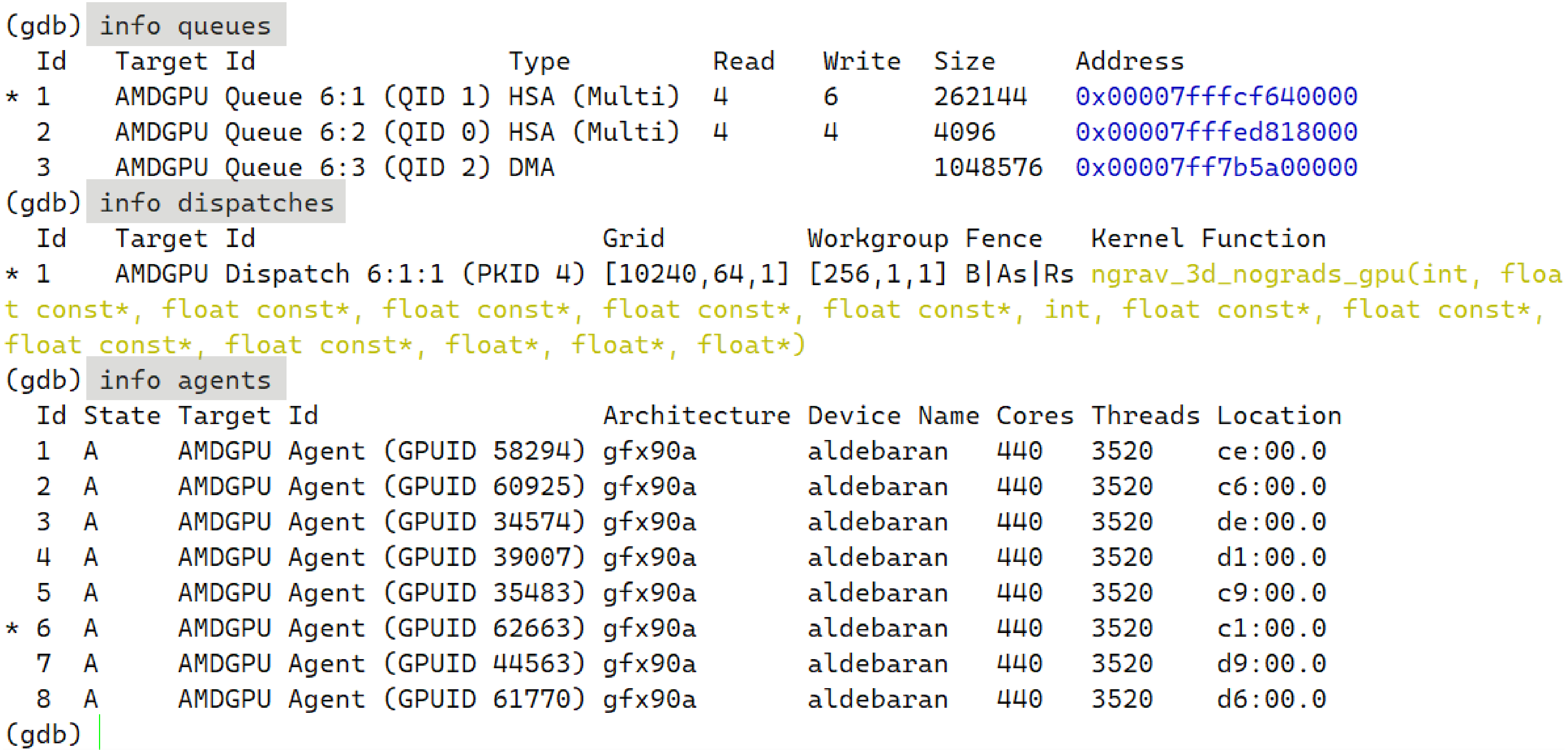
- The HIP runtime currently performs deferred code object loading by default. AMD GPU code objects are not loaded until the first kernel is launched.
  - Set breakpoints normally, just confirm [y]
  - Avoid a breakpoint that many threads or lanes will see
- Some convenience vars are available
  - `print $_wave_id` shows workgroup x,y,z and work group thread index
  - `print $_lane_workgroup_pos` shows work item x,y,z
- Examine GPU registers
  - `info registers [scalar | vector | system]`
  - `print/f $v20`
- Step and next will jump to another thread/lane that hit the breakpoint
  - Avoid this with `set scheduler-locking step`
- Most gdb commands should work normally

rocgdb – Set breakpoint and run



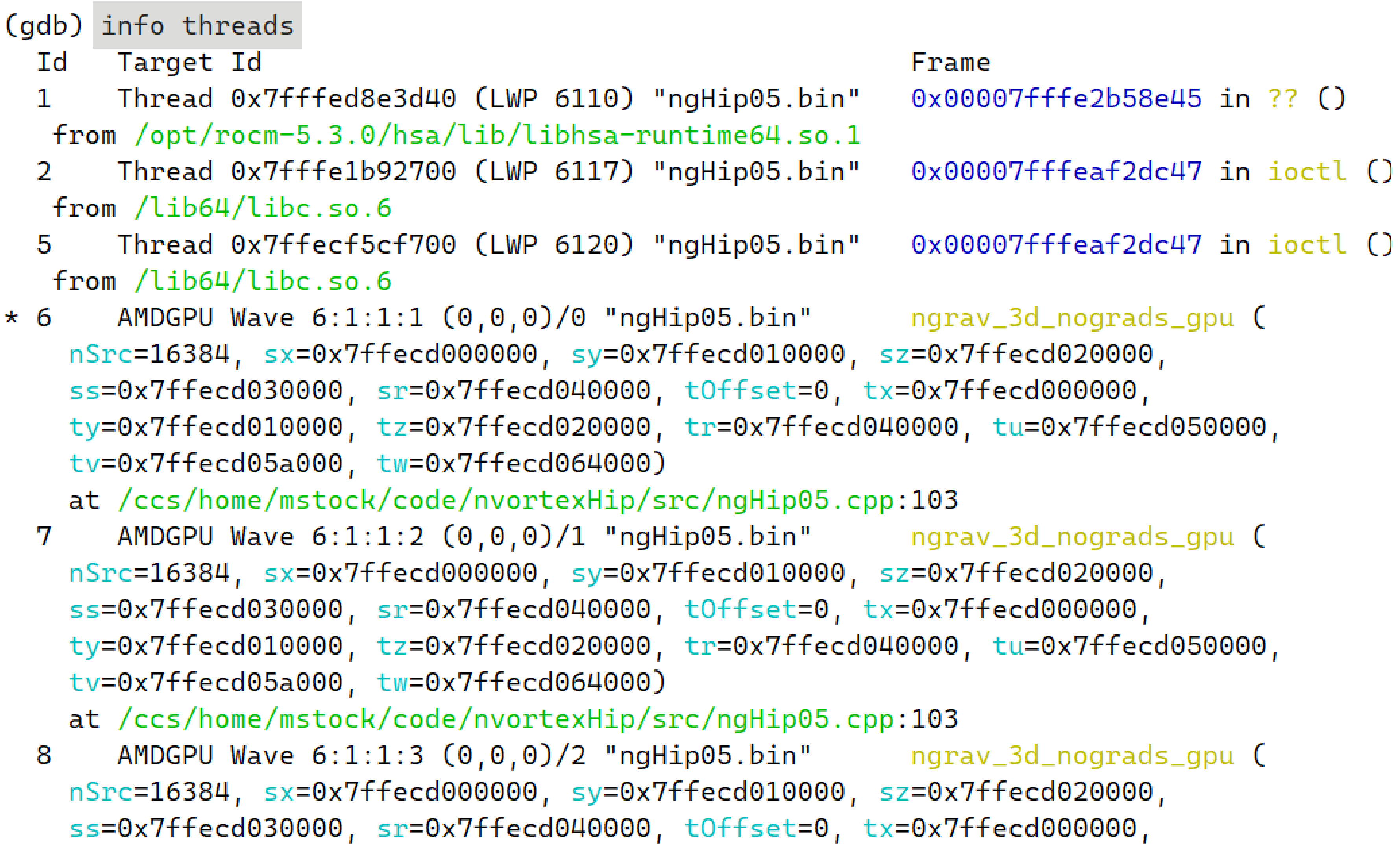
```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ngHip05.bin...
(gdb) break ngHip05.cpp:96
No compiled code for line 96 in file "ngHip05.cpp".
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (ngHip05.cpp:96) pending.
(gdb) run -n=10000 -g=1
Starting program: /autofs/nccs-svm1_home1/mstock/code/nvortexHip/RelDbg_amd540/ngHip05.bin -n=10000
 -g=1
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
performing 3D gravitational summation on 10000 points
[New Thread 0x7fffe1b92700 (LWP 106873)]
[New Thread 0x7ff7ce7ff700 (LWP 106874)]
[Thread 0x7ff7ce7ff700 (LWP 106874) exited]
[Switching to thread 4, lane 0 (AMDGPU Lane 6:1:1:1/0 (0,0,0)[0,0,0])]

Thread 4 "ngHip05.bin" hit Breakpoint 1, with lanes [0-63], ngrav_3d_nograds_gpu () at /ccs/home/ms
tock/code/nvortexHip/src/ngHip05.cpp:98
98          FLOAT dx = s_sx[j] - tx[i];
(gdb) 
```

rocgdb – Looking around



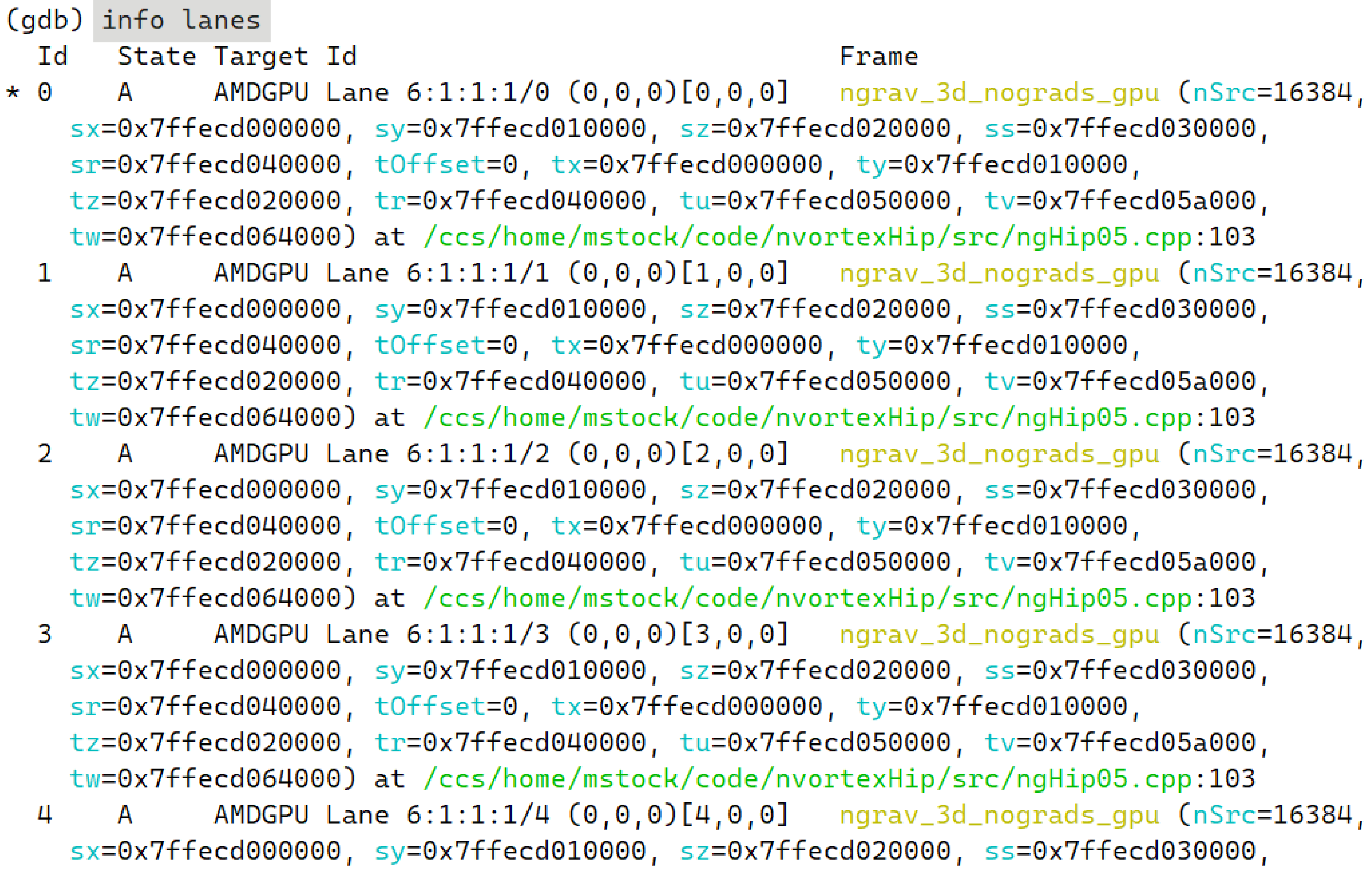
```
(gdb) info queues
  Id   Target Id                Type         Read   Write  Size     Address
* 1    AMDGPU Queue 6:1 (QID 1) HSA (Multi)  4      6      262144   0x00007fffcf640000
  2    AMDGPU Queue 6:2 (QID 0) HSA (Multi)  4      4      4096     0x00007fffed818000
  3    AMDGPU Queue 6:3 (QID 2) DMA                        1048576  0x00007ff7b5a00000
(gdb) info dispatches
  Id   Target Id                      Grid         Workgroup Fence   Kernel Function
* 1    AMDGPU Dispatch 6:1:1 (PKID 4) [10240,64,1] [256,1,1] B|As|Rs ngrav_3d_nograds_gpu(int, floa
t const*, float const*, float const*, float const*, float const*, int, float const*, float const*,
float const*, float const*, float*, float*, float*)
(gdb) info agents
  Id State Target Id                  Architecture Device Name Cores Threads Location
  1  A     AMDGPU Agent (GPUID 58294) gfx90a       aldebaran   440   3520    ce:00.0
  2  A     AMDGPU Agent (GPUID 60925) gfx90a       aldebaran   440   3520    c6:00.0
  3  A     AMDGPU Agent (GPUID 34574) gfx90a       aldebaran   440   3520    de:00.0
  4  A     AMDGPU Agent (GPUID 39007) gfx90a       aldebaran   440   3520    d1:00.0
  5  A     AMDGPU Agent (GPUID 35483) gfx90a       aldebaran   440   3520    c9:00.0
* 6  A     AMDGPU Agent (GPUID 62663) gfx90a       aldebaran   440   3520    c1:00.0
  7  A     AMDGPU Agent (GPUID 44563) gfx90a       aldebaran   440   3520    d9:00.0
  8  A     AMDGPU Agent (GPUID 61770) gfx90a       aldebaran   440   3520    d6:00.0
(gdb)
```

rocgdb - Viewing threads



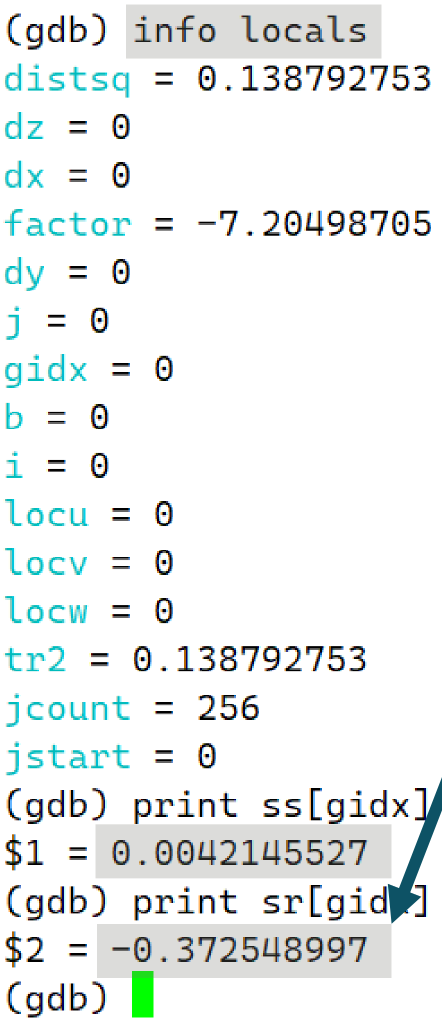
```
(gdb) info threads
  Id   Target Id                                         Frame
  1    Thread 0x7fffed8e3d40 (LWP 6110) "ngHip05.bin"   0x00007fffe2b58e45 in ?? ()
   from /opt/rocm-5.3.0/hsa/lib/libhsa-runtime64.so.1
  2    Thread 0x7fffe1b92700 (LWP 6117) "ngHip05.bin"   0x00007fffeaf2dc47 in ioctl ()
   from /lib64/libc.so.6
  5    Thread 0x7ffecf5cf700 (LWP 6120) "ngHip05.bin"   0x00007fffeaf2dc47 in ioctl ()
   from /lib64/libc.so.6
* 6    AMDGPU Wave 6:1:1:1 (0,0,0)/0 "ngHip05.bin"      ngrav_3d_nograds_gpu (
    nSrc=16384, sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000,
    ss=0x7ffecd030000, sr=0x7ffecd040000, tOffset=0, tx=0x7ffecd000000,
    ty=0x7ffecd010000, tz=0x7ffecd020000, tr=0x7ffecd040000, tu=0x7ffecd050000,
    tv=0x7ffecd05a000, tw=0x7ffecd064000)
    at /ccs/home/mstock/code/nvortexHip/src/ngHip05.cpp:103
  7    AMDGPU Wave 6:1:1:2 (0,0,0)/1 "ngHip05.bin"      ngrav_3d_nograds_gpu (
    nSrc=16384, sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000,
    ss=0x7ffecd030000, sr=0x7ffecd040000, tOffset=0, tx=0x7ffecd000000,
    ty=0x7ffecd010000, tz=0x7ffecd020000, tr=0x7ffecd040000, tu=0x7ffecd050000,
    tv=0x7ffecd05a000, tw=0x7ffecd064000)
    at /ccs/home/mstock/code/nvortexHip/src/ngHip05.cpp:103
  8    AMDGPU Wave 6:1:1:3 (0,0,0)/2 "ngHip05.bin"      ngrav_3d_nograds_gpu (
    nSrc=16384, sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000,
    ss=0x7ffecd030000, sr=0x7ffecd040000, tOffset=0, tx=0x7ffecd000000,
```

rocgdb - Viewing lanes



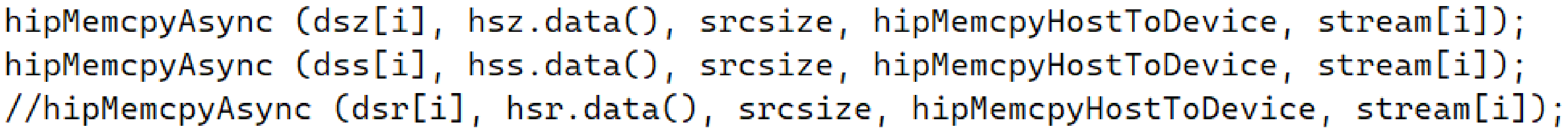
```
(gdb) info lanes
  Id   State Target Id                              Frame
* 0    A     AMDGPU Lane 6:1:1:1/0 (0,0,0)[0,0,0]   ngrav_3d_nograds_gpu (nSrc=16384,
    sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000, ss=0x7ffecd030000,
    sr=0x7ffecd040000, tOffset=0, tx=0x7ffecd000000, ty=0x7ffecd010000,
    tz=0x7ffecd020000, tr=0x7ffecd040000, tu=0x7ffecd050000, tv=0x7ffecd05a000,
    tw=0x7ffecd064000) at /ccs/home/mstock/code/nvortexHip/src/ngHip05.cpp:103
  1    A     AMDGPU Lane 6:1:1:1/1 (0,0,0)[1,0,0]   ngrav_3d_nograds_gpu (nSrc=16384,
    sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000, ss=0x7ffecd030000,
    sr=0x7ffecd040000, tOffset=0, tx=0x7ffecd000000, ty=0x7ffecd010000,
    tz=0x7ffecd020000, tr=0x7ffecd040000, tu=0x7ffecd050000, tv=0x7ffecd05a000,
    tw=0x7ffecd064000) at /ccs/home/mstock/code/nvortexHip/src/ngHip05.cpp:103
  2    A     AMDGPU Lane 6:1:1:1/2 (0,0,0)[2,0,0]   ngrav_3d_nograds_gpu (nSrc=16384,
    sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000, ss=0x7ffecd030000,
    sr=0x7ffecd040000, tOffset=0, tx=0x7ffecd000000, ty=0x7ffecd010000,
    tz=0x7ffecd020000, tr=0x7ffecd040000, tu=0x7ffecd050000, tv=0x7ffecd05a000,
    tw=0x7ffecd064000) at /ccs/home/mstock/code/nvortexHip/src/ngHip05.cpp:103
  3    A     AMDGPU Lane 6:1:1:1/3 (0,0,0)[3,0,0]   ngrav_3d_nograds_gpu (nSrc=16384,
    sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000, ss=0x7ffecd030000,
    sr=0x7ffecd040000, tOffset=0, tx=0x7ffecd000000, ty=0x7ffecd010000,
    tz=0x7ffecd020000, tr=0x7ffecd040000, tu=0x7ffecd050000, tv=0x7ffecd05a000,
    tw=0x7ffecd064000) at /ccs/home/mstock/code/nvortexHip/src/ngHip05.cpp:103
  4    A     AMDGPU Lane 6:1:1:1/4 (0,0,0)[4,0,0]   ngrav_3d_nograds_gpu (nSrc=16384,
    sx=0x7ffecd000000, sy=0x7ffecd010000, sz=0x7ffecd020000, ss=0x7ffecd030000,
```

rocgdb – Lane-local variables



```
(gdb) info locals
distsq = 0.138792753
dz = 0
dx = 0
factor = -7.20498705
dy = 0
j = 0
gidx = 0
b = 0
i = 0
locu = 0
locv = 0
locw = 0
tr2 = 0.138792753
jcount = 256
jstart = 0
(gdb) print ss[gidx]
$1 = 0.0042145527
(gdb) print sr[gidx]
$2 = -0.372548997
(gdb) 
```

Why is my particle radius negative?

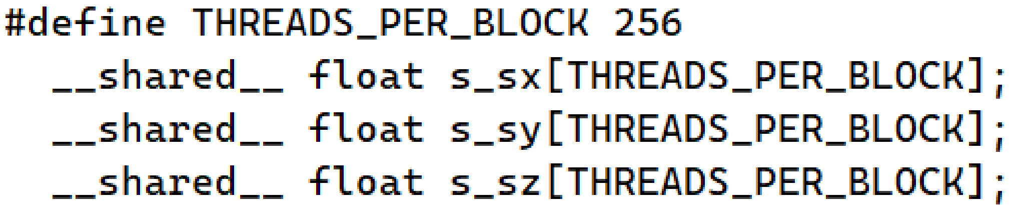


```
hipMemcpyAsync (dsz[i], hsz.data(), srcsize, hipMemcpyHostToDevice, stream[i]);
hipMemcpyAsync (dss[i], hss.data(), srcsize, hipMemcpyHostToDevice, stream[i]);
//hipMemcpyAsync (dsr[i], hsr.data(), srcsize, hipMemcpyHostToDevice, stream[i]);
```

Maybe because I didn’t copy that array to the device!

rocgdb – Shared memory (LDS)

This is the LDS declaration in our kernel code:



```
#define THREADS_PER_BLOCK 256
  __shared__ float s_sx[THREADS_PER_BLOCK];
  __shared__ float s_sy[THREADS_PER_BLOCK];
  __shared__ float s_sz[THREADS_PER_BLOCK];
```

LDS starts at `local` byte 0

Each array here is 1024 bytes, so

`s_sx` starts at 0x0

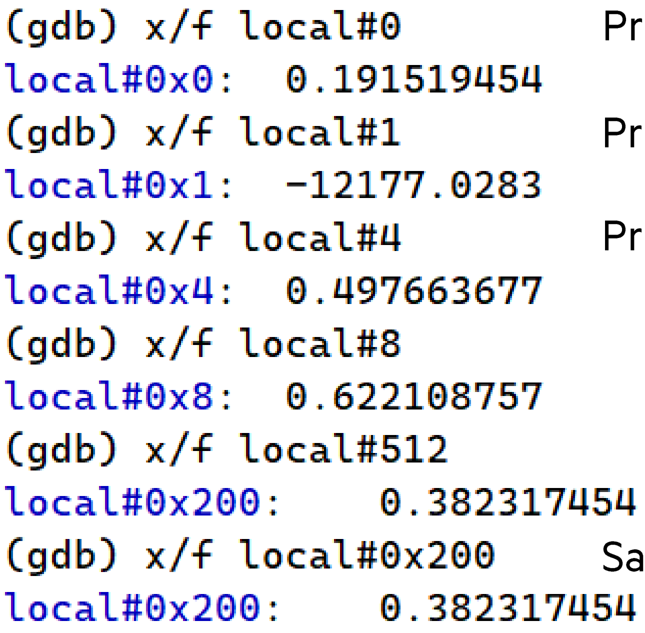
`s_sy` starts at 1024, or 0x400

You can't `print` values from LDS in gdb yet

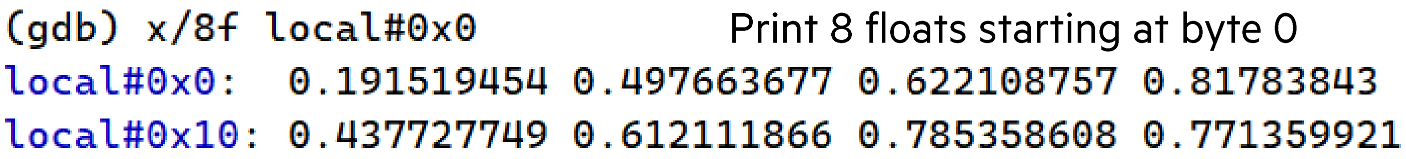
Use `x` to view an address as an integer

Use `x/f` to view as a 4-byte float

Use `x/gf` to view as a double-precision float



```
(gdb) x/f local#0           Print float starting at byte 0
local#0x0:  0.191519454
(gdb) x/f local#1           Print float starting at byte 1 – bad!
local#0x1:  -12177.0283
(gdb) x/f local#4           Print float starting at byte 4
local#0x4:  0.497663677
(gdb) x/f local#8
local#0x8:  0.622108757
(gdb) x/f local#512
local#0x200:    0.382317454
(gdb) x/f local#0x200       Same address as above, but in hex
local#0x200:    0.382317454
```



```
(gdb) x/8f local#0x0                    Print 8 floats starting at byte 0
local#0x0:  0.191519454 0.497663677 0.622108757 0.81783843
local#0x10: 0.437727749 0.612111866 0.785358608 0.771359921
```

gdb4hpc

- A parallel harness and aggregator around gdb / rocgdb / cuda-gdb
- Load the module to have it in your path and the man pages available
  - `module load gdb4hpc`
  - `man gdb4hpc`
- gdb4hpc will allocate and srun for you, but you need to unload xalt first
  - `module unload xalt`
- Find help inside gdb4hpc by utilizing the `help` command

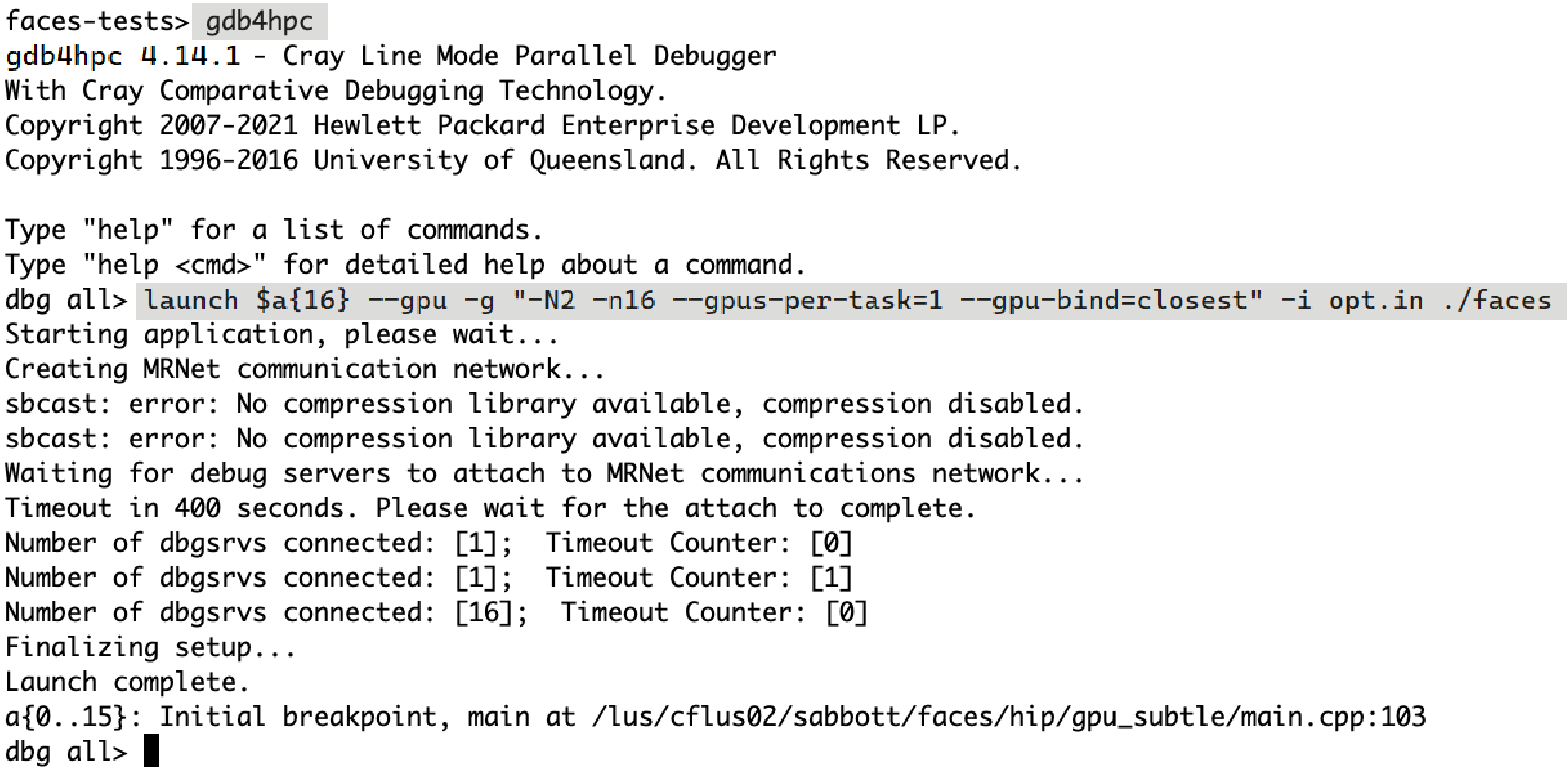
`help`	list all the commands
`help [command]`	print detailed help about a particular command
`help info threads`	display information on the info threads command.

- You can still debug your application at non-zero optimization levels although you might not be getting all the information that you desire when debugging
- gdb4hpc supports launching and attaching, and side-by-side debugging

gdb4hpc – Anatomy of a launch

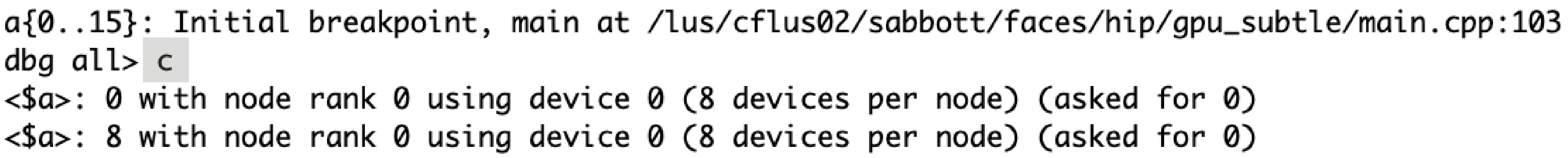
`launch $a{16}`	Launch process set “a” with 8 ranks
`--gpu`	We want to use a GPU debugger
`--env="MPICH_GPU_SUPPORT_ENABLED=1"`	gdb4hpc will use your environment, but set any additional values here
`-g "-N2 -n16 --gpu-bind=closest"`	Pass job launcher arguments
`-a "512 512 512"`	Your app’s arguments
`-i opt.in`	An input file to hand to stdin
`./faces`	The binary to debug

gdb4hpc – Launching your app

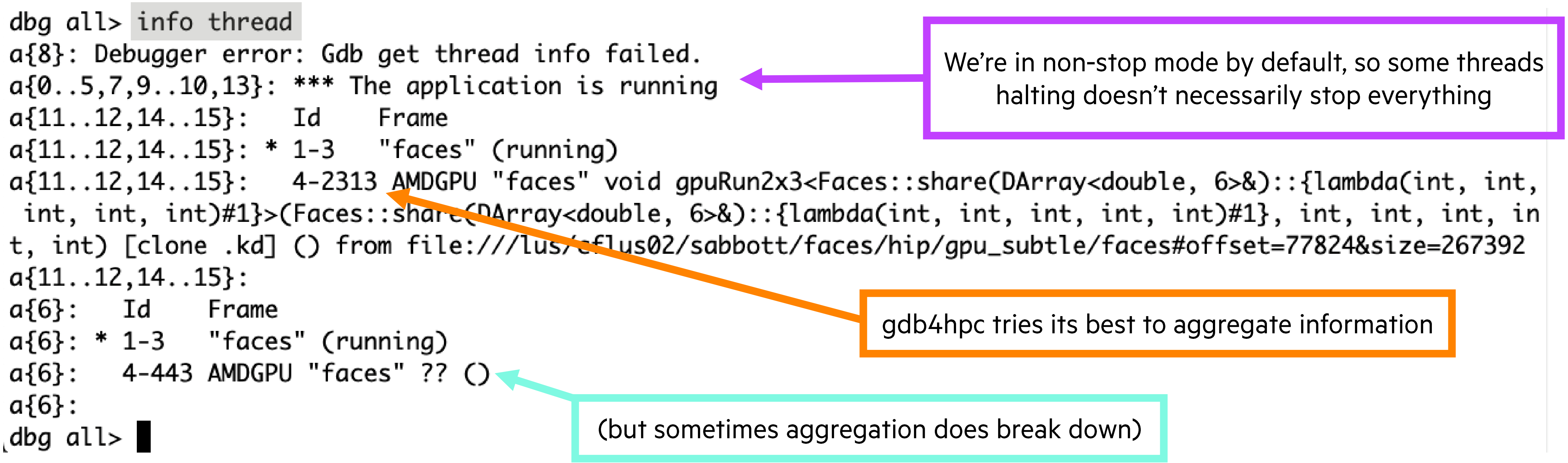


```
faces-tests> gdb4hpc
gdb4hpc 4.14.1 - Cray Line Mode Parallel Debugger
With Cray Comparative Debugging Technology.
Copyright 2007-2021 Hewlett Packard Enterprise Development LP.
Copyright 1996-2016 University of Queensland. All Rights Reserved.

Type "help" for a list of commands.
Type "help <cmd>" for detailed help about a command.
dbg all> launch $a{16} --gpu -g "-N2 -n16 --gpus-per-task=1 --gpu-bind=closest" -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> █
```

gdb4hpc - Thread aggregation



```
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> 
```

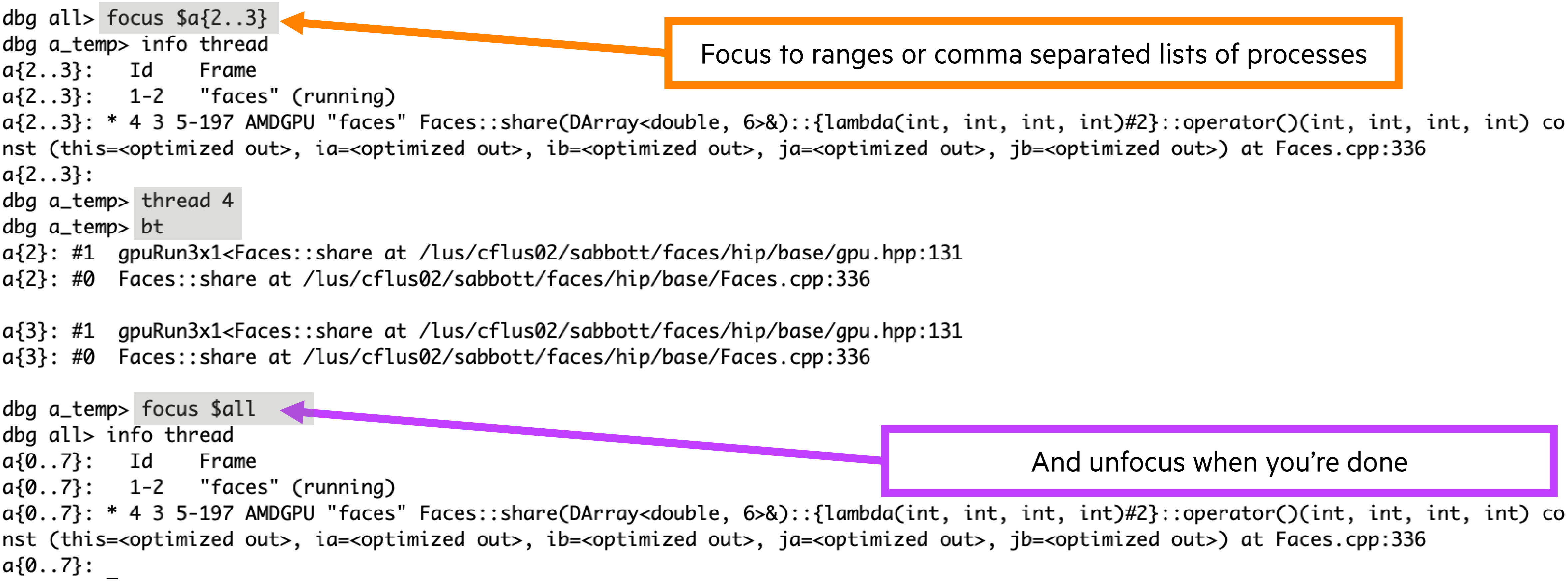
We're in non-stop mode by default, so some threads halting doesn't necessarily stop everything

gdb4hpc tries its best to aggregate information

(but sometimes aggregation does break down)

gdb4hpc – Focus on what matters

- The `focus` command lets you isolate specific processes/ranks

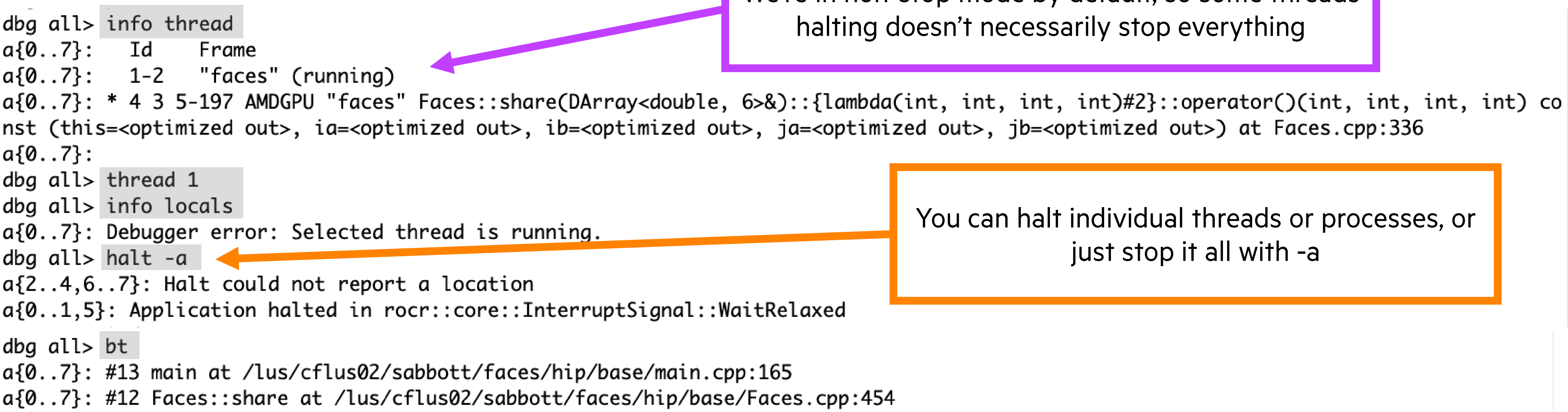


```
dbg all> focus $a{2..3}
dbg a_temp> info thread
a{2..3}:   Id    Frame
a{2..3}:   1-2   "faces" (running)
a{2..3}: * 4 3 5-197 AMDGPU "faces" Faces::share(DArray<double, 6>&)::{lambda(int, int, int, int)#2}::operator()(int, int, int, int) co
nst (this=<optimized out>, ia=<optimized out>, ib=<optimized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

Focus to ranges or comma separated lists of processes

And unfocus when you’re done

gdb4hpc – Halt it all

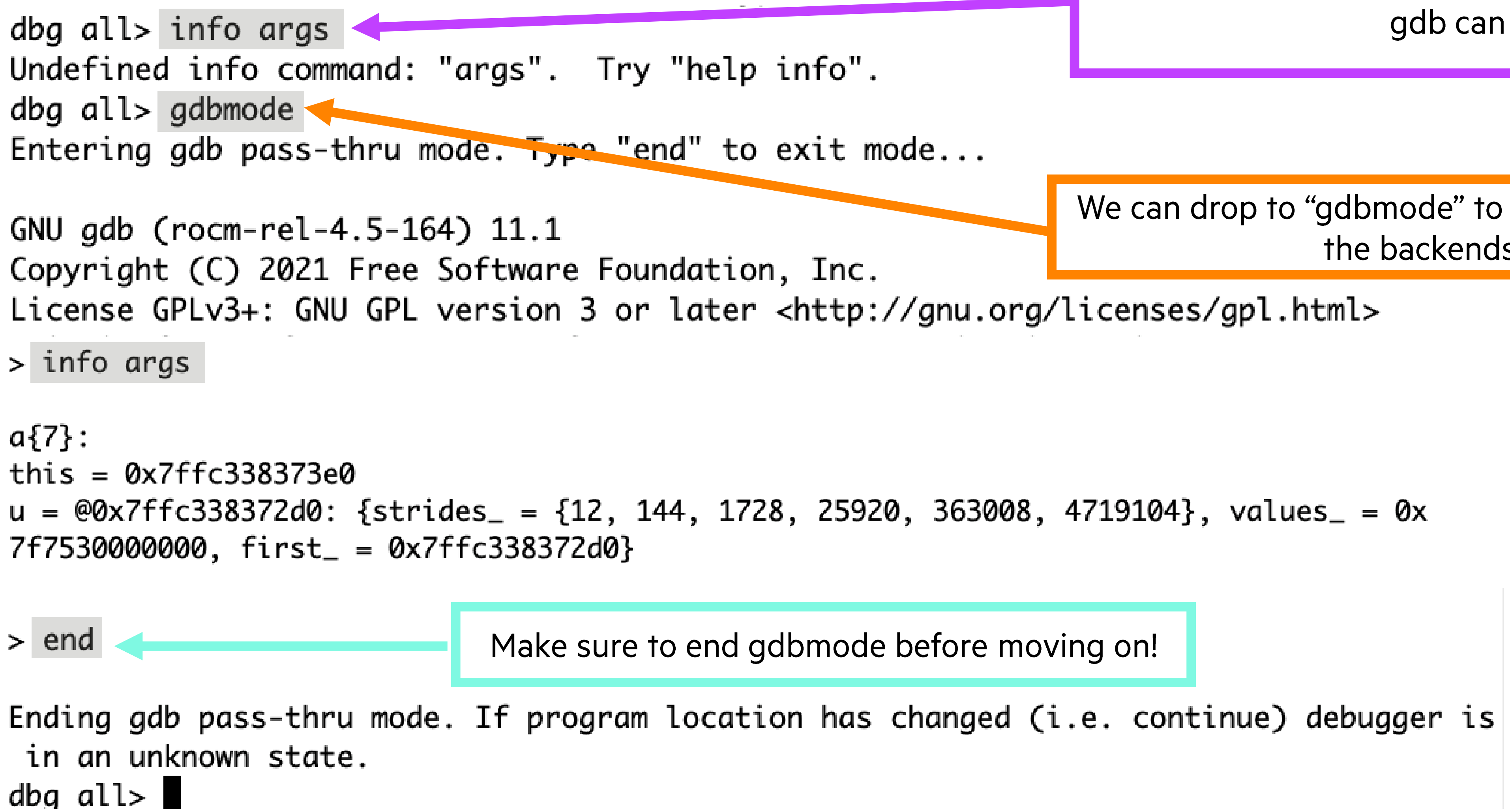


```
dbg all> info thread
a{0..7}:   Id     Frame
a{0..7}:   1-2    "faces" (running)
a{0..7}: * 4 3 5-197 AMDGPU "faces" Faces::share(DArray<double, 6>&)::{lambda(int, int, int, int)#2}::operator()(int, int, int, int) co
nst (this=<optimized out>, ia=<optimized out>, ib=<optimized out>, ja=<optimized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

gdb4hpc – Dig deeper with gdbmode



```
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> 
```

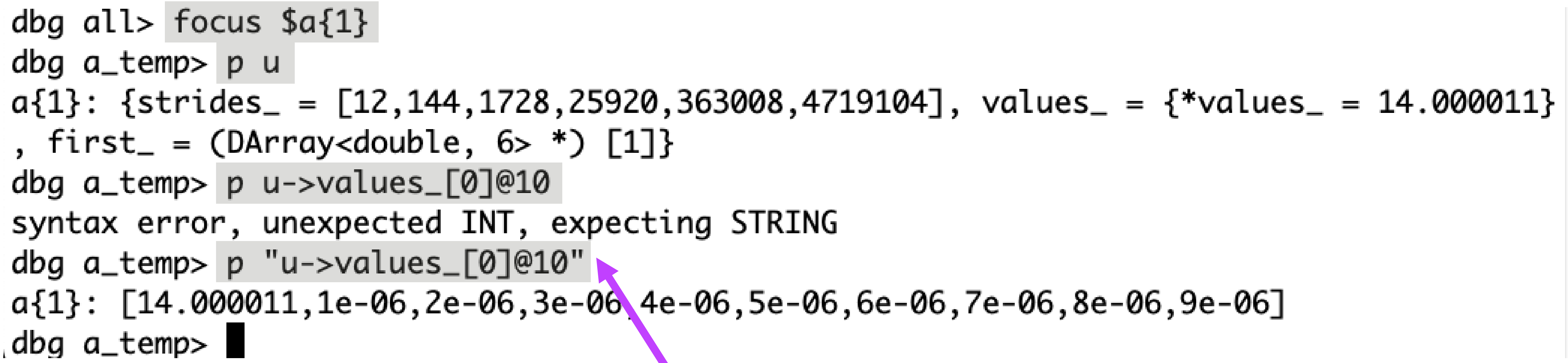
gdb4hpc doesn’t have commands for *everything* gdb can do

We can drop to “gdbmode” to get raw access to the backends

Make sure to end gdbmode before moving on!

gdb4hpc – mini-gdbmode for inline prints

- You can do mini-gdbmode inline for some things



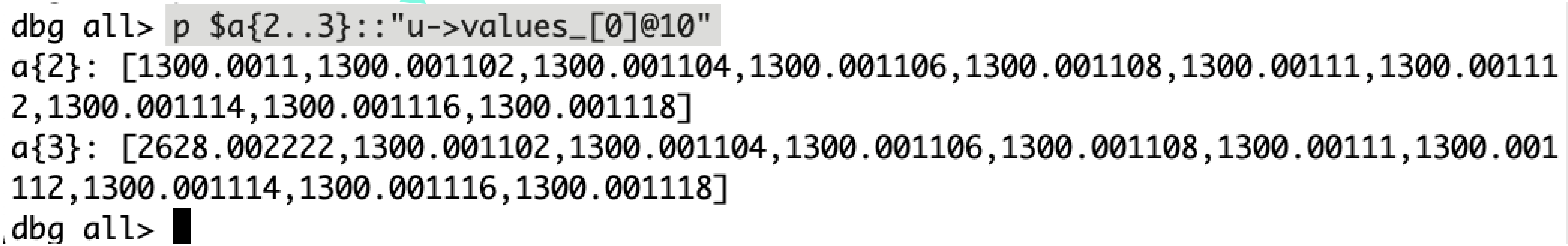
```
dbg all> focus $a{1}
dbg a_temp> p u
a{1}: {strides_ = [12,144,1728,25920,363008,4719104], values_ = {*values_ = 14.000011}
, first_ = (DArray<double, 6> *) [1]}
dbg a_temp> p u->values_[0]@10
syntax error, unexpected INT, expecting STRING
dbg a_temp> p "u->values_[0]@10"
a{1}: [14.000011,1e-06,2e-06,3e-06,4e-06,5e-06,6e-06,7e-06,8e-06,9e-06]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

Quotation marks evaluate the expression in GDB mode

gdb4hpc – Focus on a single print

- 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

- Understand what your bug *could* be before you go looking for it
  - A few well-designed tests may illuminate its location
- Understand what tools are at your disposal and what they can be used for
  - Keep this PDF around for reference
- Try to remember that every debugging session is a learning experience
  - If you knew what the bug was, you wouldn't need to debug
- There are tools we *didn't* talk about here
  - Address sanitizers
  - Thread sanitizers
  - Visualizers

University of Texas at Austin, CC0

Where to go for help

- Manual pages
  - Notably: intro_mpi, intro_openmp, CC
- OLCF
  - Frontier User Guide: https://docs.olcf.ornl.gov/systems/frontier_user_guide.html
  - Training archive (slides and videos): https://docs.olcf.ornl.gov/training/training_archive.html
  - *OLCF help email: help@olcf.ornl.gov*
- AMD
  - rocgdb PDF manuals and references: `$ROCM_PATH/share/doc/rocgdb/`
    or online: https://docs.amd.com/bundle/ROCDebugger-User-Guide-v5.4/page/Summary.html
  - Lab Notes: https://gpuopen.com/learn/amd-lab-notes/
- HPE Support Documents
  - https://support.hpe.com/connect/s/
  - "Fortran reference manual" "Cray Programming Environment User Guide"

Thank you

Mark Stock
mark.stock@hpe.com

Special thanks to:
Steve Abbot, Trey White, Kostas Makrides, Srinath Vadlamani (HPE), Tom Papatheodore (OLCF)