Hewlett Packard Enterprise

TOOLS AND STRATEGIES FOR DEBUGGING ON HPE-CRAY SYSTEMS

Kostas Makrides
Performance Engineer

20-May-2021

COPYRIGHT AND TRADEMARK ACKNOWLEDGEMENTS

FORWARD LOOKING STATEMENTS

This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries ("Hewlett Packard Enterprise") may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties' control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray's business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise's businesses; the competitive pressures faced by Hewlett Packard Enterprise's businesses; risks associated with executing Hewlett Packard Enterprise's strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise's other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.

AGENDA

- Introductory Materials
  - OpenMP in the Cray Programming Environment
  - Example module setup
- Debugging applications with CRAY_ACC_DEBUG
  - Example
- Compiler Listings
  - Detailed walkthrough of an example
- Cray Performance Tools and Apprentice2
  - Sampling
  - Tracing
  - GPU Tracing
  - Apprentice2
- Introduction to GDB4HPC
- Questions

INTRODUCTORY MATERIALS

INTRODUCTORY MATERIALS

OpenMP in the Cray Programming Environment

- Our Fortran (`ftn`) and C/C++ (`cc,CC`) compilers are fully compliant with the openMP 4.5 Standard
  - Add `-homp` for Fortran or `-fopenmp` for C/C++ to both the compile and link lines to build/link with openMP support
  - There is no additional flag needed for OpenMP offload, however you will need to load the appropriate module for the target GPU
    - `craype-accel-amd-gfx906` : for the MI60 GPUs
    - `craype-accel-amd-gfx908` : for the MI100 GPUs
    - `craype-accel-nvidia70` : for the V100 GPUs
- We have implemented many OpenMP 5.0 features in CCE-11 (a number from the 5.1 standard)
  - See `man intro_openmp` for a complete list of OpenMP 5.0 features that are currently supported
- There are a few ways that you can use the environment to change the behavior of the runtime
  - **CRAY_ACC_DEBUG**: is a great way to get information about our offload directives
  - `CRAY_ACC_REUSE_MEM_LIMIT`: Option to control how much memory the openMP runtime will hold on to
- For our C/C++ friends
  - Can be a replacement for upstream clang (`-fno-cray` removes HPE/Cray added enhancements)

INTRODUCTORY MATERIALS

Example environment on Spock

- Environment setup on Spock with MI100 GPU offload support

```
module load craype-accel-amd-gfx908 rocm/4.1.0
Currently Loaded Modules:
 1) cce/11.0.4             5) craype-network-ofi                       9) cray-mpich/8.1.4       13) DefApps/default
 2) craype/2.7.6           6) cray-dsmml/0.1.4                        10) cray-libsci/21.04.1.1  14) PrgEnv-cray/8.0.0
 3) craype-x86-rome        7) perftools-base/21.02.0                  11) cray-pmi/6.0.10       15) craype-accel-amd-gfx908
 4) libfabric/1.11.0.3.74  8) xpmem/2.2.40-2.1_2.7__g3cf3325.shasta  12) cray-pmi-lib/6.0.10    16) rocm/4.1.0
```

- Environment setup on Redwood with MI100 GPU offload support

```
Currently Loaded Modulefiles:
1) cce/12.0.0.9005            5) shared                     9) perftools-base/21.05.0    13) craype-accel-amd-gfx908
2) craype/2.7.8.1             6) cuda11.2/toolkit/11.2.0   10) slurm/slurm/19.05.7
3) craype-x86-naples          7) cray-mvapich2/2.3.5       11) PrgEnv-cray/8.1.0
4) craype-network-infiniband  8) cray-libsci/20.03.1       12) rocm/4.1.1
```

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

- We provide a mechanism that emits messages for all offloading operations
  - Most offloading operations can trace back to the source with line numbers
  - You can thus pinpoint where in the source data is flowing
- Setting CRAY_ACC_DEBUG to 1, 2, or 3 controls the level of verbosity from the OpenMP runtime
  - **`CRAY_ACC_DEBUG=2`** is designed to be rather user friendly and is recommended for users
  - `CRAY_ACC_DEBUG=3` is very verbose and uses idioms that may not be straightforward at first sight
    - I often default to this value when utilizing this tool
  - `CRAY_ACC_DEBUG=1` is the least verbose of the three
    - I don't typically use this level
- All this functionality is for "free"
  - You do not need to re-compile your application
  - There are no special flags to add to either compile nor link steps
- In practice, this is the first option that I set when things break
  - I often can avoid using the more heavyweight debuggers by using this option (and maybe some print statements)

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

CRAY_ACC_DEBUG=1 output

- The output shows transfers to the host and to the accelerator
  - Source line numbers are shown
    - I've condensed the file names here for visual purposes. Complete paths can show up in the output
  - Does not show the variable being transferred.
  - The amount of data being transferred to the device and the host are shown for each transfer.
- Kernel execution is also shown in the output.

```
CRAY_ACC_DEBUG=1 srun -N 1 -n 1 -p amdMI100 ./omp_map_derived_types
ACC: Transfer 1 items (to acc 120 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 3 items (to acc 80000 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 1 items (to acc 128 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 3 items (to acc 80000 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 2 items (to acc 80000 bytes, to host 0 bytes) from main.f:35
ACC: Transfer 5 items (to acc 0 bytes, to host 0 bytes) from main.f:37
ACC: Execute kernel derived_type_openmp_$ck_L37_1 async(auto) from main.f:37
ACC: Wait async(auto) from main.f:37
ACC: Transfer 4 items (to acc 0 bytes, to host 0 bytes) from main.f:37
ACC: Transfer 5 items (to acc 0 bytes, to host 0 bytes) from main.f:37
ACC: Execute kernel derived_type_openmp_$ck_L37_1 async(auto) from main.f:37
ACC: Wait async(auto) from main.f:37
ACC: Transfer 4 items (to acc 0 bytes, to host 0 bytes) from main.f:37
ACC: Transfer 5 items (to acc 0 bytes, to host 0 bytes) from main.f:37
ACC: Execute kernel derived_type_openmp_$ck_L37_1 async(auto) from main.f:37
ACC: Wait async(auto) from main.f:37
ACC: Transfer 4 items (to acc 0 bytes, to host 0 bytes) from main.f:37
ACC: Transfer 5 items (to acc 0 bytes, to host 0 bytes) from main.f:37
```

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

```
 2 program derived_type_openmp
 3   use operation_def, only : multiplication
 4   use setup, only : setup_types, remove_types, op_ptr, op_ptr_b
 5
 6   implicit none
 7   real(kind=8), dimension(:,:), pointer :: v, dv
 8   integer :: a, i, iop
 9   character(len=32) :: arg
10
11   ! Size of arrays to allocate
12   a = 100
13   ! Operation value to use
14   iop = 2
15
16   ! Setup derived types
17     call setup_types(a,iop)
18
19   ! Allocate arrays to operate on
20   allocate( v(a,a), dv(a,a) )
21
22   ! Initialize v
23   v = 4
24
25   print *, 'v = ', v(1,1)
26   print *, 'array = ',op_ptr%array(1,1)
27   print *, 'type-bound array = ',op_ptr_b%array(1,1)
28
29 ! Call multiplication operator
30 print *, 'Calling operations....'
31
32 !==========================================!
33 !== Direct call to multiply ===============!
34 ! Perform operation on v to get dv
35     !$omp target data map(to:v) map(from:dv)
36   do i=1,10
37     call multiplication(op_ptr, v, dv,
38   end do
39     !$omp end target data
```

```
ACC: Transfer 1 items (to acc 120 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 3 items (to acc 80000 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 1 items (to acc 128 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 3 items (to acc 80000 bytes, to host 0 bytes) from main.f:17
```

```
ACC: Transfer 2 items (to acc 80000 bytes, to host 0 bytes) from main.f:35
```

```
ACC: Transfer 5 items (to acc 0 bytes, to host 0 bytes) from main.f:37
ACC: Execute kernel derived_type_openmp_$ck_L37_1 async(auto) from main.f:37
ACC: Wait async(auto) from main.f:37
```

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

```
2  program derived_type_openmp
3   use operation_def, only : multiplication
4   use setup, only : setup_types, remove_types, op_ptr, op_ptr_b
5
6   implicit none
7   real(kind=8), dimension(:,:), pointer :: v, dv
8   integer :: a, i, iop
9   character(len=32) :: arg
10
11  ! Size of arrays to allocate
12  a = 100
13  ! Operation value to use
14  iop = 2
15
16  ! Setup derived types
17    call setup_types(a,iop)
18
19  ! Allocate arrays to operate on
20  allocate( v(a,a), dv(a,a) )
21
22  ! Initialize v
23  v = 4
24
25  print *, 'v = ', v(1,1)
26  print *, 'array = ',op_ptr%array(1,1)
27  print *, 'type-bound array = ',op_ptr_b%array(1,1)
28
29 ! Call multiplication operator
30 print *, 'Calling operations....'
31
32 !=========================================!
33 !== Direct call to multiply ===============!
34 ! Perform operation on v to get dv
35   !$omp target data map(to:v) map(from:dv)
36  do i=1,10
37   call multiplication(op_ptr, v, dv,
38  end do
39   !$omp end target data
```

```
ACC: Transfer 1 items (to acc 120 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 3 items (to acc 80000 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 1 items (to acc 128 bytes, to host 0 bytes) from main.f:17
ACC: Transfer 3 items (to acc 80000 bytes, to host 0 bytes) from main.f:17
```

```
12  subroutine setup_types(a,i)
13    implicit none
14    integer :: i, a
15    ! Setup multiplication operation
16    call operation_b%setup(a,a,i)
17    call operation%setup(a,a,i)
18
19    ! Set pointer
20    op_ptr_b => operation_b
21    op_ptr => operation
22
23    ! Map derived type data to GPU
24    !$omp target enter data map(to:op_ptr)
25    !$omp target enter data map(to:op_ptr%array)
26    !$omp target enter data map(to:op_ptr_b)
27    !$omp target enter data map(to:op_ptr_b%array)
28
29  end subroutine setup_types
```

```
ACC: Trans ... 35
```

```
ACC: Trans
ACC: Execu
ACC: Wait ... f:37
```

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

CRAY_ACC_DEBUG=2 output

- The output shows transfers to the accelerator and to the host
  - Line numbers in the source are also listed in this view
  - A message is printed for data that is already present on the device
- Variables on the map directives are also listed in this output
  - Arrays with unknown shape information at compile time are shown with question marks in the output
- It's important to note the "pointer attach" information for derived types with pointer components
- This view gives us a decent view of the memory transfers occurring in our application.

```
CRAY_ACC_DEBUG=2 srun -N 1 -n 1 -p amdMI100 ./omp_map_derived_types
ACC: Version 4.0 of HIP already initialized, runtime version 3212
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from main.f:17
ACC:       allocate, copy to acc 'op_ptr' (120 bytes)
ACC: End transfer (to acc 120 bytes, to host 0 bytes)
ACC: Start transfer 3 items from main.f:17
ACC:       allocate, copy to acc 'op_ptr%array(?:?,?:?)' (80000 bytes)
ACC:       present 'op_ptr' (120 bytes)
ACC:       attach pointer 'op_ptr%array' (96 bytes)
ACC: End transfer (to acc 80000 bytes, to host 0 bytes)
ACC: Start transfer 1 items from main.f:17
ACC:       allocate, copy to acc 'op_ptr_b' (128 bytes)
ACC: End transfer (to acc 128 bytes, to host 0 bytes)
ACC: Start transfer 3 items from main.f:17
ACC:       allocate, copy to acc 'op_ptr_b%base_type%array(?:?,?:?)'
(80000 bytes)
ACC:       present 'op_ptr_b' (128 bytes)
ACC:       attach pointer 'op_ptr_b%base_type%array' (96 bytes)
ACC: End transfer (to acc 80000 bytes, to host 0 bytes)
```

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

CRAY_ACC_DEBUG=2 output

- The output shows transfers to the the memory transfers occurring in our application.

```
subroutine setup_types(a,i)
  implicit none
  integer :: i, a
  ! Setup multiplication operation
  call operation_b%setup(a,a,i)
  call operation%setup(a,a,i)

  ! Set pointer
  op_ptr_b => operation_b
  op_ptr => operation

  ! Map derived type data to GPU
  !$omp target enter data map(to:op_ptr)
  !$omp target enter data map(to:op_ptr%array)
  !$omp target enter data map(to:op_ptr_b)
  !$omp target enter data map(to:op_ptr_b%array)

end subroutine setup_types
```

```
CRAY_ACC_DEBUG=2 srun -N 1 -n 1 -p amdMI100 ./omp_map_derived_types
 HIP already initialized, runtime version 3212

ext
 1 items from main.f:17
, copy to acc 'op_ptr' (120 bytes)
to acc 120 bytes, to host 0 bytes)
 3 items from main.f:17
, copy to acc 'op_ptr%array(?:?,?:?)' (80000 bytes)
'op_ptr' (120 bytes)
ointer 'op_ptr%array' (96 bytes)
to acc 80000 bytes, to host 0 bytes)
 1 items from main.f:17
, copy to acc 'op_ptr_b' (128 bytes)
to acc 128 bytes, to host 0 bytes)
 3 items from main.f:17
, copy to acc 'op_ptr_b%base_type%array(?:?,?:?)'

'op_ptr_b' (128 bytes)
ointer 'op_ptr_b%base_type%array' (96 bytes)
ACC: End transfer (to acc 80000 bytes, to host 0 bytes)
```

DEBUGGING APPLICATIONS WITH CRAY_ACC_DEBUG

CRAY_ACC_DEBUG=3 output

```
ACC: Start transfer 3 items from main.f:17
ACC:   flags: NEED_POST_PHASE
ACC:
ACC:   Transfer Phase
ACC:   Trans 1
ACC:       Simple transfer of 'op_ptr%array(?:?,?:?)' (96 bytes)
ACC:           host ptr 7fffffff71c0
ACC:           acc  ptr 0
ACC:           flags: DOPE_VECTOR DV_ONLY_DATA ALLOCATE
COPY_HOST_TO_ACC ACQ_PRESENT REG_PRESENT
ACC:           Transferring dope vector
ACC:             dim:1 lowbound:1 extent:100 stride_mult:1
ACC:             dim:2 lowbound:1 extent:100 stride_mult:100
ACC:             DV size=80000 (scale:8, elem_size:8)
ACC:             total mem size=80000 (dv:0 obj:80000)
ACC:           memory not found in present table
ACC:           allocate (80000 bytes)
ACC:             get new reusable memory, added entry
ACC:           new allocated ptr (154e5640a000)
ACC:           add to present table index 1: host 4dedc0 to
4f2640, acc 154e5640a000
ACC:           copy host to acc (4dedc0 to 154e5640a000)
ACC:             internal copy host to acc (host 4dedc0 to acc
154e5640a000) size = 80000
ACC:           new acc ptr 154e5640a000
ACC:
ACC:   Trans 2
ACC:       Simple transfer of 'op_ptr' (120 bytes)
ACC:           host ptr 410cc0
ACC:           acc  ptr 0
ACC:           flags: ALLOCATE ACQ_PRESENT REG_PRESENT
ACC:           host region 410cc0 to 410d38 found in present table
index 0 (ref count 2)
ACC:           memory found in present table (154e56409000, base
154e56409000)
ACC:           new acc ptr 154e56409000
ACC:
ACC:   Trans 3
ACC:   Post Transfer Phase
ACC:   Trans 1
ACC:   Trans 2
ACC:   Trans 3
ACC:       Simple transfer of 'op_ptr%array' (96 bytes)
ACC:           host ptr 410cc0
ACC:           acc  ptr 0
ACC:           flags: REG_PRESENT OMP_PTR_ATTACH
ACC:           host region 4dedc0 to 4dedc1 found in present table
index 1 (ref count 1)
ACC:             attach pointer host 0x410cc0 (pointee 0x4dedc0)
to device 154e56409000 (pointee 154e5640a000) for 'op_ptr%array'
from main.f:17
ACC:             internal copy host to acc (host 154e5b000ba0 to
acc 154e56409000) size = 96
ACC:
ACC: End transfer (to acc 80000 bytes, to host 0 bytes)
```

COMPILER LISTINGS

- You can get compiler listings by compiling with
  Fortran: `-hlist=a`
  C/C++: `-fsave-loopmark`
  - This generates a `.lst` file with the listing
- The top of the listing file gives you a legend for the symbols in-between the line numbers and source.
  - I : inlined
  - p: partial
  - r: unrolled
  - V: vectorized
  - G: Accelerated
  - F: Flattened
  - M: Multithreaded
  - C: Collapsed
- Lines with a "+" indicate that there are additional comments further down the listing file.

```
  13.                          ! Operation value to use
  14.                          iop = 2
  15.
  16.                          ! Setup derived types
  17.  + Ip                    call setup_types(a,iop)
  18.
  19.                          ! Allocate arrays to operate on
  20.                          allocate( v(a,a), dv(a,a) )
  21.
  22.                          ! Initialize v
  23.    VCr2------------<>      v = 4
  24.
 …
  35.  +                      !$omp target data map(to:v) map(from:dv)
  36.  + F----------------<   do i=1,10
  37.  + F MpmGCFr8 I----<>    call multiplication(op_ptr, v, dv, a)
ftn-7256 ftn: WARNING DERIVED_TYPE_OPENMP, File = main.f, Line = 37
  An OpenMP parallel construct in a target region is limited to a single thread.
  38.    F---------------->   end do
  39.                         !$omp end target data
  40.
…
  45.                          ! Perform operation on v to get dv
  46.  +                      !$omp target data map(to:v) map(from:dv)
  47.  + F----------------<    do i=1,10
  48.  + F MmgGCFr8 I----<>      call op_ptr_b%multiply(v, dv, a)
ftn-7256 ftn: WARNING DERIVED_TYPE_OPENMP, File = main.f, Line = 48
  An OpenMP parallel construct in a target region is limited to a single thread.
  49.    F---------------->    end do
  50.                         !$omp end target data
```

COMPILER LISTINGS

- You c… with Fortra… C/C++…
  - This generates a `.lst` file with the listing
- The top of the listing file gives you a legend for the symbols in-between the line numbers and source.
  - I : inlined
  - p: partial
  - r: unrolled
  - V: vectorized
  - G: Accelerated
  - F: Flattened
  - M: Multithreaded
  - C: Collapsed
- Lines with a "+" indicate that there are additional comments further down the listing file.

```
13.                                  ! Operation value to use
14.                                  iop = 2
15.
16.                                  ! Setup derived types
17.  + Ip                            call setup_types(a,iop)
18.
```

```
ftn-3001 ftn: IPA DERIVED_TYPE_OPENMP, File = main.f, Line = 17, Column = 20
  Routine "setup_types"(/home/users/cmakrides/presentations/fortran_tools/FGPU/openmp/target_map/derived_types/setup.f:12) was
  textually inlined because argument 1 is a constant.  NOT INLINED:  setup_values : setup_values.
```

```
23.    VCr2------------<>            v = 4
24.
…
35.  +                              !$omp target data map(to:v) map(from:dv)
36.  + F----------------<           do i=1,10
37.  + F MpmGCFr8 I----<>            call multiplication(op_ptr, v, dv, a)
ftn-7256 ftn: WARNING DERIVED_TYPE_OPENMP, File = main.f, Line = 37
```

```
ftn-6405 ftn: ACCEL DERIVED_TYPE_OPENMP, File = main.f, Line = 37
  A region starting at line 37 and ending at line 37 was placed on
the accelerator.
```

```
…
45.                                  ! Perform operation on v to get dv
46.  +                              !$omp target data map(to:v) map(from:dv)
47.  + F----------------<            do i=1,10
48.  + F MmgGCFr8 I----<>              call op_ptr_b%multiply(v, dv, a)
ftn-7256 ftn: WARNING DERIVED_TYPE_OPENMP, File = main.f, Line = 48
```

```
ftn-6005 ftn: SCALAR DERIVED_TYPE_OPENMP, File = main.f, Line = 48
  A loop starting at line 48 was unrolled 8 times.
```

APPLICATION USE CASE

Mapping Derived Types with Pointer components

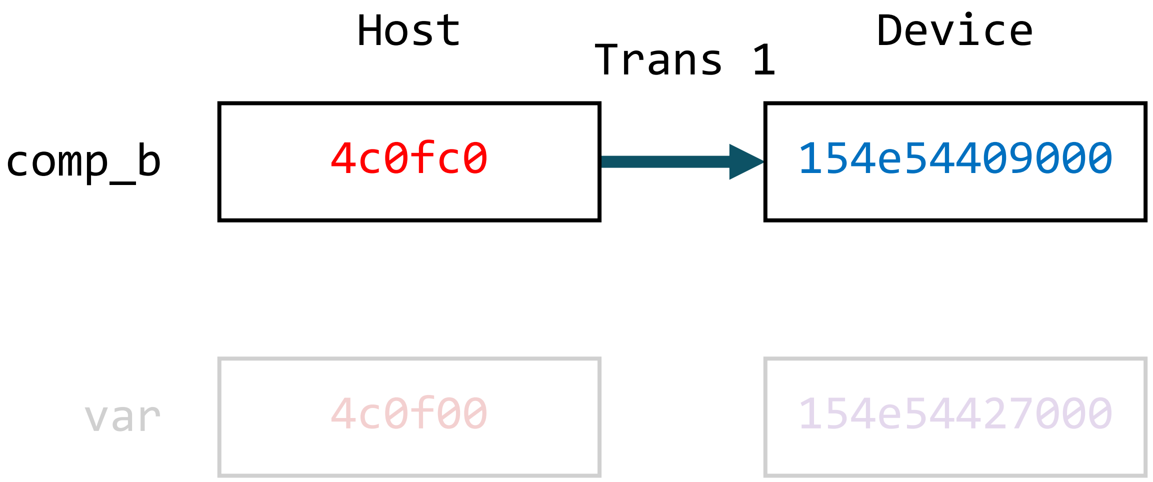
- Between the compiler listings and the output from CRAY_ACC_DEBUG many errors can be successfully understood
- This is a reproducer from another application of a data offloading procedure
  - We've identified using the compiler listings on the real application that the analogous "map_array" subroutine call was not being inlined and have added a directive to explicitly do so here.
- I will use compiler listings and **`CRAY_ACC_DEBUG=3`** to clarify the data offloading operation examining two cases
  - "inlined" case: line 28 is replaced line 13
    `!$omp target enter data map(to:var%comp_b)`
  - "Non-inlined" case: the program as it is on the right

```
1  module test_map
2    type type_a
3       integer, pointer, contiguous :: comp_b(:)
4    end type type_a
5
6  contains
7
8    subroutine map_array(h_ptr)
9      implicit none
10
11     integer, pointer :: h_ptr(:)
12
13     !$omp target enter data map(to:h_ptr)
14   end subroutine map_array
15 end module test_map
16
17 program test_mapper
18 !DIR$ NOINLINE
19   use test_map
20   implicit none
21   integer, parameter :: n=30000
22   integer :: i
23
24   type(type_a), allocatable:: var
25   allocate(var)
26   allocate(var%comp_b(n))
27
28   call map_array(var%comp_b)
29
30   !$omp target teams distribute simd
31   do i=1,n
32      var%comp_b(i) = i
33   end do
34
35 end program test_mapper
```

APPLICATION USE CASE

Mapping Derived Types with Pointer components, inlined

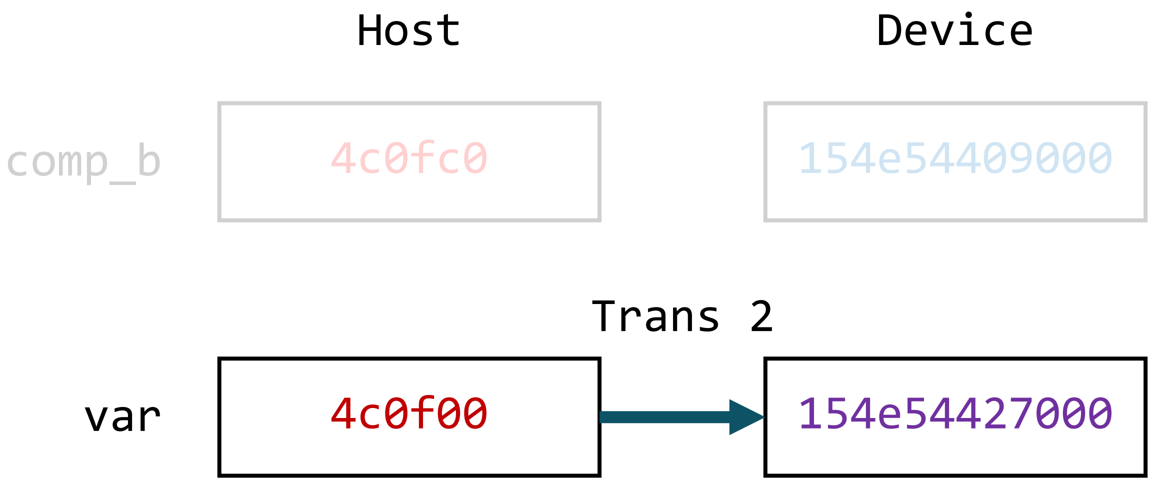
```
ACC: Start transfer 3 items from origMain.F90:28
ACC:   flags: NEED_POST_PHASE
ACC:   Transfer Phase
ACC:   Trans 1
ACC:       Simple transfer of 'var%comp_b(:)' (72 bytes)
ACC:            host ptr 4c0f00
ACC:            acc  ptr 0
ACC:            flags: DOPE_VECTOR DV_ONLY_DATA ALLOCATE
COPY_HOST_TO_ACC ACQ_PRESENT REG_PRESENT
ACC:            Transferring dope vector
ACC:              DV size=120000 (dim:1 extent:30000
stride_mult:1 scale:4 elem_size:4)
ACC:              total mem size=120000 (dv:0 obj:120000)
ACC:            memory not found in present table
ACC:            allocate (120000 bytes)
ACC:              get new reusable memory, added entry
ACC:            new allocated ptr (154e54409000)
ACC:            add to present table index 0: host 4c0fc0 to
4de480, acc 154e54409000
ACC:            copy host to acc (4c0fc0 to 154e54409000)
ACC:                internal copy host to acc (host 4c0fc0 to
acc 154e54409000) size = 120000
ACC:            new acc ptr 154e54409000
```



APPLICATION USE CASE

Mapping Derived Types with Pointer components, inlined

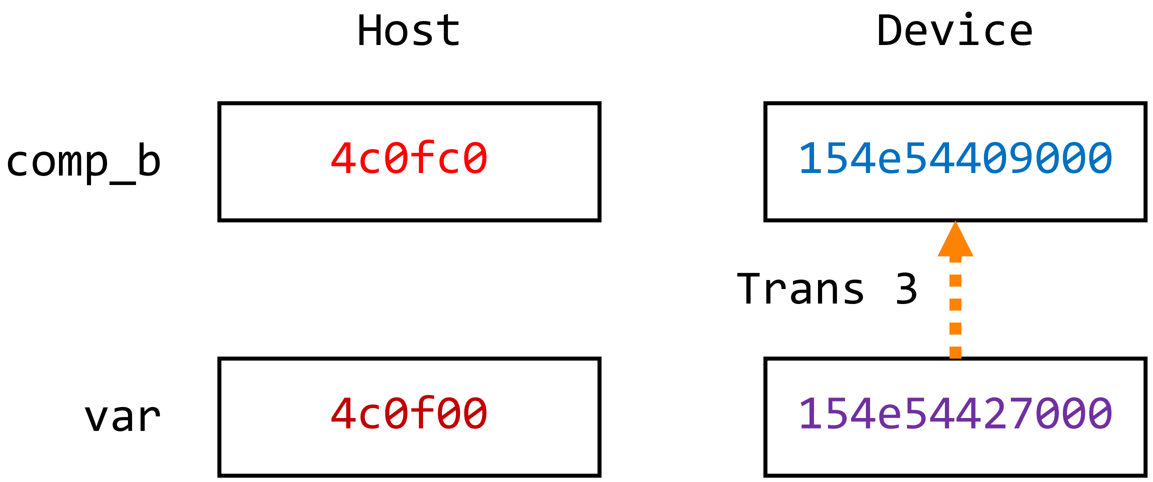
```
ACC:   Trans 2
ACC:       Simple transfer of 'var' (72 bytes)
ACC:            host ptr 4c0f00
ACC:            acc  ptr 0
ACC:            flags: ALLOCATE ACQ_PRESENT
REG_PRESENT
ACC:            memory not found in present table
ACC:            allocate (72 bytes)
ACC:              get new reusable memory, added entry
ACC:            new allocated ptr (154e54427000)
ACC:            add to present table index 1: host
4c0f00 to 4c0f48, acc 154e54427000
ACC:            new acc ptr 154e54427000
```



APPLICATION USE CASE

Mapping Derived Types with Pointer components, inlined

```
ACC:    Trans 3
ACC:    Post Transfer Phase
ACC:    Trans 1
ACC:    Trans 2
ACC:    Trans 3
ACC:        Simple transfer of 'var%comp_b' (72 bytes)
ACC:             host ptr 4c0f00
ACC:             acc  ptr 0
ACC:             flags: REG_PRESENT OMP_PTR_ATTACH
ACC:             host region 4c0fc0 to 4c0fc1 found in
present table index 0 (ref count 1)
ACC:               attach pointer host 0x4c0f00 (pointee
0x4c0fc0) to device 154e54427000 (pointee 154e54409000)
for 'var%comp_b' from origMain.F90:28
ACC:                 internal copy host to acc (host d38b70
to acc 154e54427000) size = 72
ACC:
ACC: End transfer (to acc 120000 bytes, to host 0 bytes)
ACC:
```

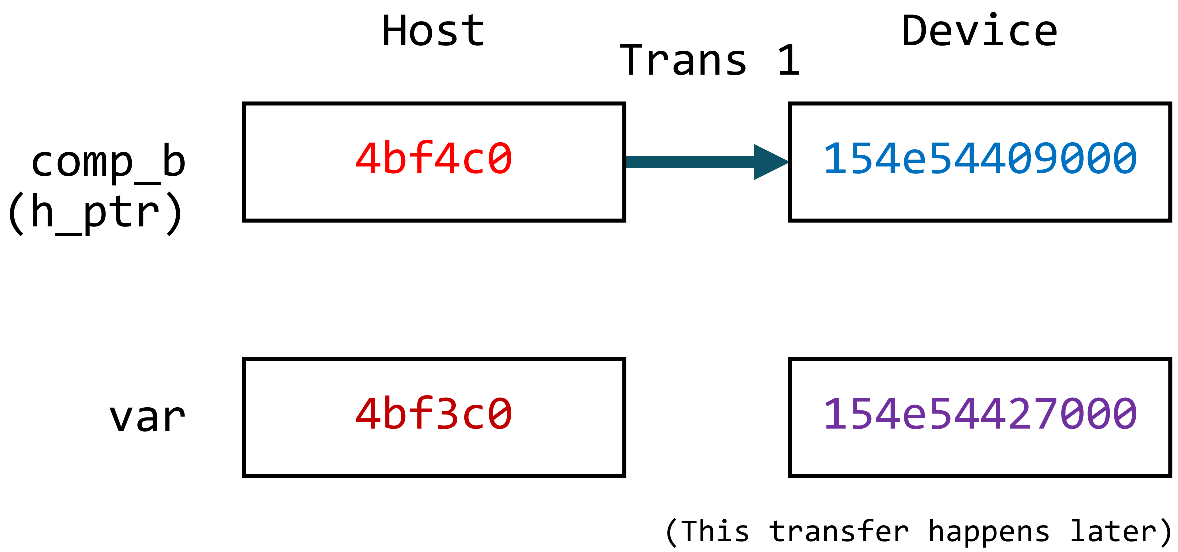


APPLICATION USE CASE

Mapping Derived Types with Pointer components, **not inlined**

```
ACC: Start transfer 1 items from origMain.F90:13
ACC:   flags:
ACC:
ACC:   Trans 1
ACC:       Simple transfer of 'h_ptr(:)' (72 bytes)
ACC:            host ptr 4bf3c0
ACC:            acc  ptr 0
ACC:            flags: DOPE_VECTOR DV_ONLY_DATA ALLOCATE
COPY_HOST_TO_ACC ACQ_PRESENT REG_PRESENT
ACC:            Transferring dope vector
ACC:              DV size=120000 (dim:1 extent:30000
stride_mult:1 scale:4 elem_size:4)
ACC:              total mem size=120000 (dv:0 obj:120000)
ACC:            memory not found in present table
ACC:            allocate (120000 bytes)
ACC:              get new reusable memory, added entry
ACC:            new allocated ptr (154e56a09000)
ACC:            add to present table index 0: host 4bf4c0 to 4dc980, acc 154e56a09000
ACC:            copy host to acc (4bf4c0 to 154e56a09000)
ACC:                internal copy host to acc (host 4bf4c0 to acc 154e56a09000) size = 120000
ACC:            new acc ptr 154e56a09000
ACC: End transfer (to acc 120000 bytes, to host 0 bytes)
ACC: Start transfer 1 items from origMain.F90:30
```

```
 8    subroutine map_array(h_ptr)
 9      implicit none
10
11      integer, pointer :: h_ptr(:)
12
13      !$omp target enter data map(to:h_ptr)
14    end subroutine map_array
15  end module test_map
```



CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Introduction

- Cray Performance Analysis Tool (CrayPat) is a performance analysis tool on Cray systems
  - Perftools-lite – simplified easy to use version of CrayPat.
    - Load the appropriate module and you should be good to go. I will not be covering this here
  - Perftools– Fully controlled version of the tool
    - Need to load the `perftools` module
    - You need to build an instrumented version of your executable using `pat_build`
    - run `pat_report` to generate reports
- Apprentice2 is a GUI for data visualization
  - It takes data collected from perftools
  - You can install it locally from: `${CRAY_PERFTOOLS_PREFIX}/share/desktop_installers`
- See `man intro_craypat`
  - Note: the perftools-base module needs to be loaded to see this man page
- Conceptually,
  nvprof ~ perftools
  nvvp ~ Apprentice2
  - They're not exact drop-in replacements for each other and there are features that don't translate (For example MPI reports)

CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Profiling using Perftools

- You will need to have the following modules loaded: `module load perftools-base/21.04.0 perftools`
  - Note that the Perftools version will get updated periodically. The upcoming 21.05 version will have substantial improvements for perftools

1. Build your application normally with the appropriate modules
   If you use cc, CC, or ftn to compile and link you usually don't need to do anything different.
   For other compilers, you will need to determine the appropriate options. The `pat_opts` utility can help derive the correct options.
2. Instrument the applications using:
   `pat_build [options] ProgramName -o instProgramName`
   Default output program name is `<ProgramName>+pat`
   1. Sampling: `pat_build ProgramName`
      Default runs a sampling experiment called Automatic Program Analysis (APA)
   2. Tracing: `pat_build –u –g MPI ProgramName` (can also substitute `-u` for `-w`)
      This will do a tracing experiment tracing user functions/subroutines and MPI API calls
3. Run the instrumented program as you would normally run your program
   This will create a directory that ends in either a 't' for tracing or an 's' for sampling
4. Generate reports from the experiment directory
   `pat_report [options] instProgramName+12345-18t`
5. Optional: After generating a single report, you can then use Apprentice2 to visualize the data
   There is better Apprentice2 support in Perftools-21.05

The `pat_help` utility that can assist with specifying the options you desire.
We have something similar to nvtx/roctx regions

`int PAT_region_begin (int id, const char *label), int PAT_region_end (int id)`

CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Profiling using CrayPat: Sampling

- I've instrumented an application for a sampling experiment (host only section)
  `pat_build -O apa -o driver+patAPA driver`
- To get a report
  `pat_report driver+patAPA+123413-18s`
- Specifics of this experiment can be found towards the end of the report
  - `Sampling interval:  10000 microsecs`
  - It also includes the command used to instrument the program
- The sampling interval by default is 10000 µs
  - This can be set by the environment variable `PAT_RT_SAMPLING_INTERVAL`
  - It's best not to make this very small since that might add biases
  - Experiment with what works best!

```
Samp% | Samp | Imb. |  Imb. | Group
      |      | Samp | Samp% |  Function
      |      |      |       |   Thread=HIDE

100.0% | 90.0 |   -- |    -- | Total
|-----------------------------------------------------------
|  85.6% | 77.0 |   -- |    -- | ETC
||----------------------------------------------------------
||  34.4% | 31.0 |   -- |    -- | do_futex_wait.constprop.1
||  34.4% | 31.0 |   -- |    -- | do_futex_wait
||  13.3% | 12.0 |   -- |    -- | __pat_memset
||   2.2% |  2.0 |   -- |    -- | __pthread_timedjoin_ex
||   1.1% |  1.0 |   -- |    -- | __cray_memset_ROME
||==========================================================
|  10.0% |  9.0 |   -- |    -- | RT
||----------------------------------------------------------
||   8.9% |  8.0 |   -- |    -- | sched_yield
||   1.1% |  1.0 |   -- |    -- | nanosleep
||==========================================================
|   2.2% |  2.0 |   -- |    -- | USER
||----------------------------------------------------------
||   1.1% |  1.0 |   -- |    -- | compare
||   1.1% |  1.0 |   -- |    -- | Array<>::index<>
||==========================================================
```

CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Profiling using CrayPat: Tracing

- I've instrumented an application for a tracing experiment (host only section)

  `pat_build -f -u -g mpi,omp driver`

  - The options after the `-g` flag specifies trace groups you wish to examine
- The MPI and OMP trace groups specify that MPI and openMP API functions are traced
  - A complete list can be found in the `pat_build` man pages
  - Specifics of each trace group can be found in `${CRAYPAT_ROOT}/share/traces`

```
Table 1:  Profile by Function Group and Function

  Time% |     Time | Imb. |  Imb. | Calls | Group
        |          | Time | Time% |       |  Function
        |          |      |       |       |   Thread=HIDE

 100.0% | 0.829906 |   -- |    -- | 601.0 | Total
|------------------------------------------------------------------
|  87.9% | 0.729575 |   -- |    -- |  57.0 | USER
||-----------------------------------------------------------------
||  58.3% | 0.483584 |   -- |    -- |   1.0 | main
||  27.4% | 0.227148 |   -- |    -- |  24.0 | faces::share
||   1.6% | 0.013412 |   -- |    -- |   2.0 | Mugs::share.LOOP@li.140
||=================================================================
|  11.7% | 0.097026 |   -- |    -- |  19.0 | OMP
||-----------------------------------------------------------------
||  11.7% | 0.096953 |   -- |    -- |   5.0 | omp_get_num_devices
|==================================================================
```

CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Profiling using CrayPat: Call tree

Using:

pat_report –O ct+src <expDirectory>

```
 100.0% | 0.829906 |     -- | Total
|-----------------------------------------------------
|  58.5% | 0.485607 |    1.0 | main:main.cpp:line.117
|  27.9% | 0.231415 |     -- | main:main.cpp:line.176
|  27.4% | 0.227086 |    8.0 |  faces::share:faces.cpp:line.591
|  11.7% | 0.096953 |     -- | main:main.cpp:line.149
|  11.7% | 0.096953 |    5.0 |  omp_get_num_devices
|   1.9% | 0.015633 |     -- | main:main.cpp:line.161
|        |          |        |  Mugs::share:Mugs.cpp:line.138
3   1.9% | 0.015585 |     -- |   CLANG$$kernel_trampoline_cray
4        |          |        |    Mugs::share:Mugs.cpp:line.143
5   1.9% | 0.015578 |     -- |     Mugs::share.REGION@li.143:Mugs.cpp:line.138
6   1.6% | 0.013412 |    2.0 |      Mugs::share.LOOP@li.140:Mugs.cpp:line.140
|=====================================================
```

Using:

pat_report –O ct<expDirectory>

```
 100.0% | 0.829906 |     -- | Total
|-----------------------------------------------
| 100.0% | 0.829906 |    1.0 | main
||----------------------------------------------
||  58.3% | 0.483584 |    1.0 | main(exclusive)
||  27.8% | 0.230748 |   24.0 | faces::share
||  11.7% | 0.096953 |    5.0 | omp_get_num_devices
||   1.9% | 0.015633 |     -- | Mugs::share
3|   1.9% | 0.015585 |     -- |  CLANG$$kernel_trampoline_cray
4|        |          |        |   Mugs::share
5|   1.9% | 0.015585 |    2.0 |    Mugs::share.REGION@li.143
6|   1.6% | 0.013412 |    2.0 |     Mugs::share.LOOP@li.140
|===============================================
```

CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Profiling using CrayPat: GPUs

- You can instrument your executable to include HIP API calls
  `pat_build -u -g hip,mpi faces`
- In a future perftools release we directly trace openMP-offload directives
  - This will appear as another section, similarly to how HIP, MPI, and USER sections are shown in the report on the right.
- Our pat build command had both HIP and MPI being traced
- The default imbalance function is

$$1 - \frac{\bar{X}_{\mathrm{Ex.M}}}{X_{\mathrm{M}}}$$

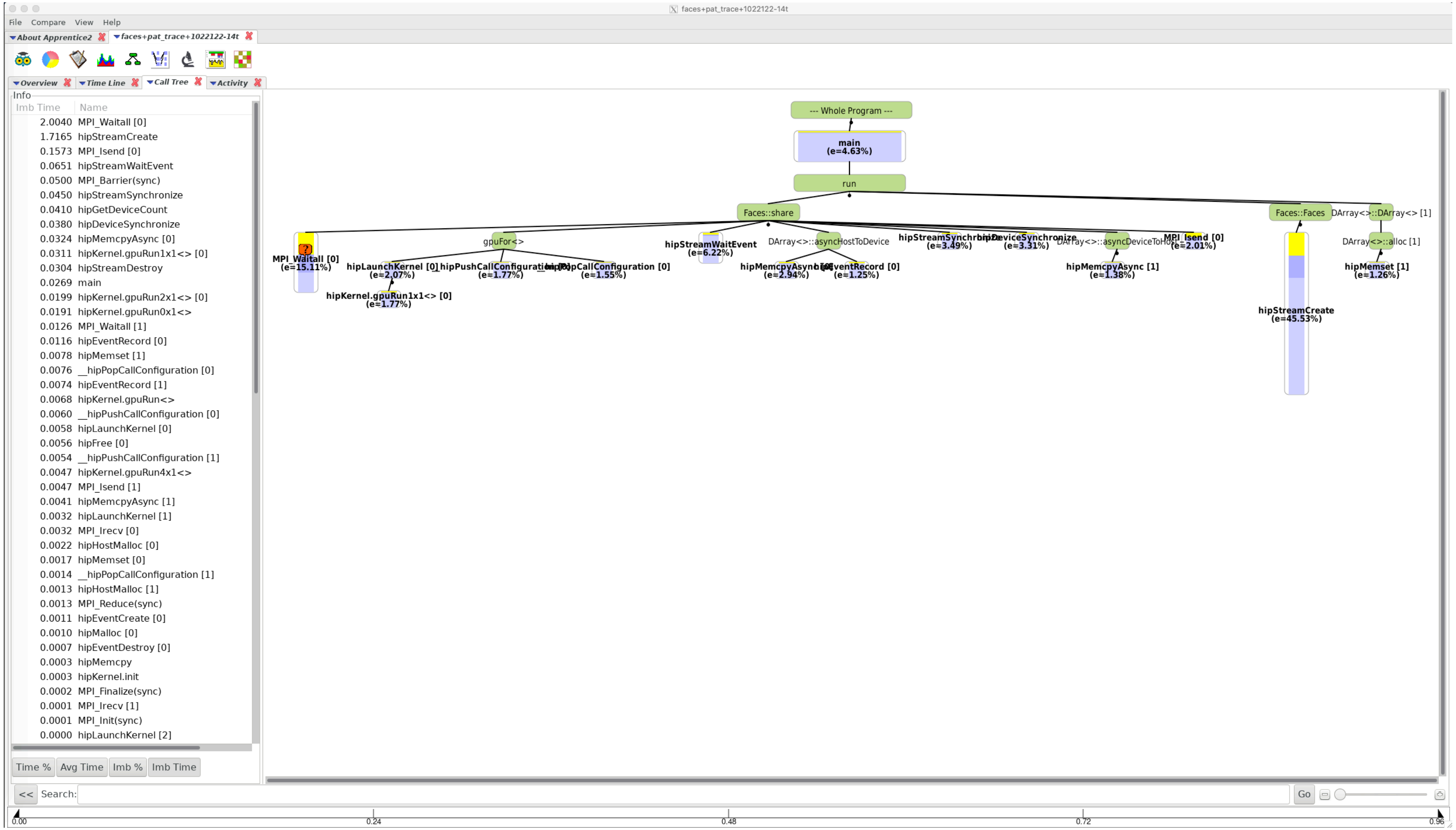
  - Where X_{M} is the maximum value of the quantity X across all ranks
  - $\bar{X}_{\mathrm{Ex.M}}$ is the average of the remaining set excluding the maximum value
  - A 100% imbalanced function occurs when X_{M} is much larger than all other X_{i}

```
 100.0% | 10.884670 |        -- |     -- | 2,037,541.0 | Total
|-----------------------------------------------------------------------------
|  85.1% |  9.257728 |        -- |     -- | 1,461,749.0 | HIP
||----------------------------------------------------------------------------
||  26.9% |  2.930075 | 0.042946 |  1.9% |       340.0 | hipStreamCreate
||  21.9% |  2.385538 | 0.005494 |  0.3% |    50,000.0 | hipStreamSynchronize
||   8.3% |  0.903107 | 0.038917 |  5.5% |    10,000.0 | hipDeviceSynchronize
||   5.0% |  0.542292 | 0.007931 |  1.9% |   400,200.0 | hipLaunchKernel
||   4.7% |  0.515733 | 0.018394 |  4.6% |    90,000.0 | hipKernel.gpuRun1x1<>
||   3.7% |  0.398302 | 0.011655 |  3.8% |   400,200.0 | __hipPushCallConfiguration
||   3.6% |  0.392681 | 0.045757 | 13.9% |    50,000.0 | hipKernel.gpuRun2x1<>
||   3.4% |  0.372933 | 0.013457 |  4.6% |   400,200.0 | __hipPopCallConfiguration
||   2.8% |  0.299503 | 0.000316 |  0.1% |       143.0 | hipMemset
||   1.7% |  0.186282 | 0.005251 |  3.7% |    30,000.0 | hipKernel.gpuRun0x1<>
||   1.2% |  0.134061 | 0.029983 | 24.4% |       170.0 | hipStreamDestroy
||   1.1% |  0.115870 | 0.002522 |  2.8% |    20,000.0 | hipKernel.gpuRun<>
||============================================================================
|   8.1% |  0.879836 |        -- |     -- |   565,648.0 | MPI
||----------------------------------------------------------------------------
||   5.0% |  0.548648 | 0.291412 | 46.3% |    40,200.0 | MPI_Waitall
||   2.7% |  0.293085 | 0.023391 |  9.9% |   262,600.0 | MPI_Isend
||============================================================================
|   6.8% |  0.741264 |        -- |     -- |    10,123.0 | USER
||----------------------------------------------------------------------------
||   5.1% |  0.558683 | 0.038988 |  8.7% |       100.0 | Mugs::share
||   1.4% |  0.151455 | 0.002882 |  2.5% |    10,000.0 | Faces::share
|=============================================================================
```

CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Apprentice2

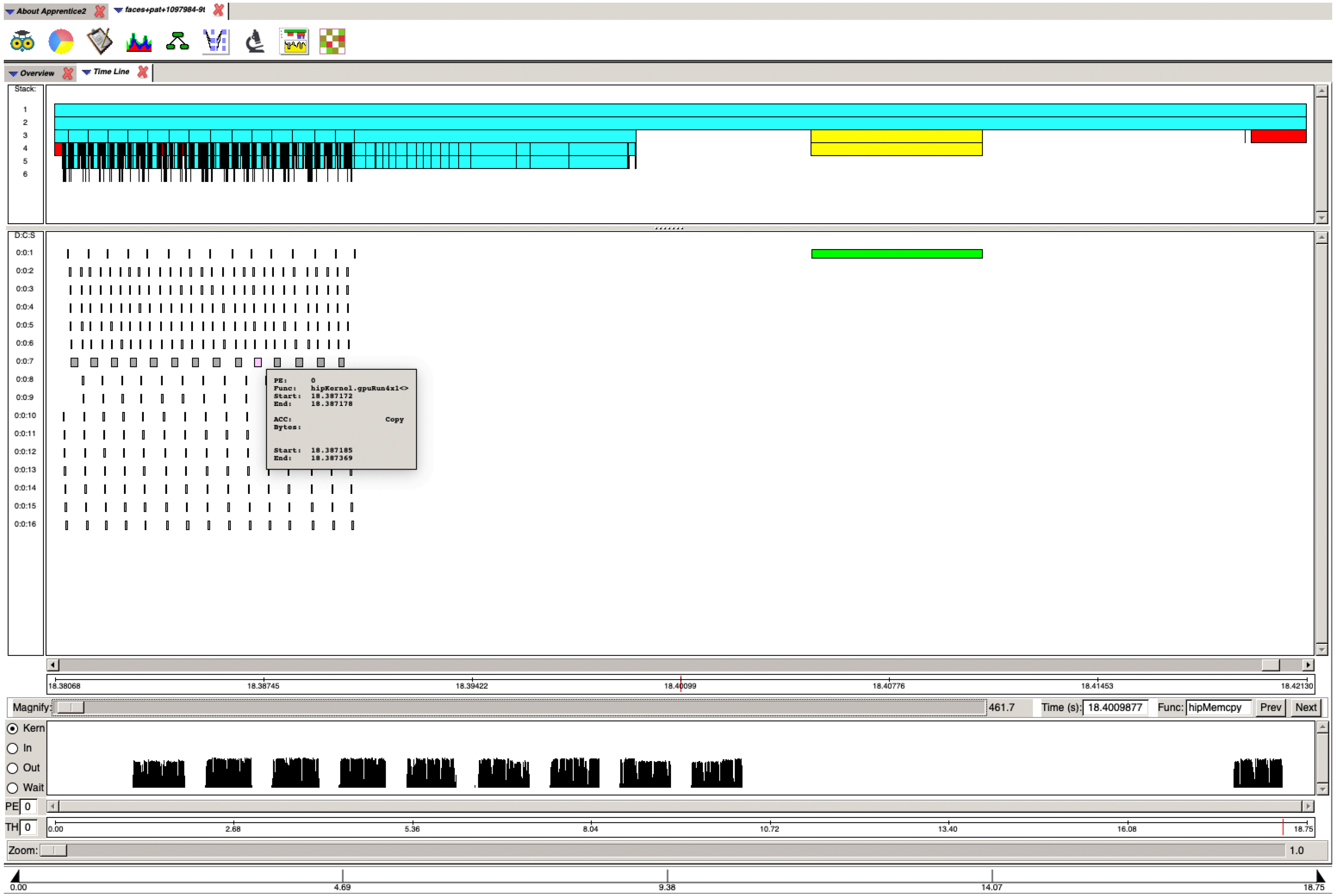
- We have a GUI that visualize data taken by perftools
  - Apprentice takes information from the experiment directories `app2 <dirName>`
  - `export PAT_RT_SUMMARY=0`
- Here we show the call tree for the faces mini-app
- You can download the experiment directory and run the application locally
- We also have options to run apprentice2 remotely as well
  - This is a bit complicated right now for Spock



CRAY PERFORMANCE ANALYSIS TOOL AND APPRENTICE2

Apprentice2

- This timeline view is similar to that of nvvp
- CPU information (and call stack) is listed above
- The main viewer has the GPU information
- Green bars are memory operations
- Grey bars are kernels
- The bottom has “activity”



GDB4HPC

GDB4HPC

- The familiar GDB debugger is a common tool to diagnose errors
  - One limiting factor is that common HPC technologies such as MPI are typically not well supported
- HPE has developed a debugger that has been designed to assist in debugging HPC applications
  - The syntax is similar to that of regular GDB
    - In fact, gdb4hpc is built on top of the GDB
- This is a very large and sophisticated program with many features
  - In this short introduction to gdb4hpc I will show some features that might be useful for getting started
- 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 command 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.
- Quick Note: 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.
  - If debugging GPU accelerated applications, it's recommended to add `-ggdb`

GDB4HPC

Quick start with launching an application

```
>$ dbg all> launch $a{2} --gpu --gdb=/opt/rocm-4.1.1/bin/rocgdb --launcher-args="--ntasks-per-node=1 --mpi=pmi2 -p amdMI100" -a "-n 10 10 10 -P 2 1 1" ./runProgram
```

- `launch $a{2}` : This launches a job with 2 MPI ranks. I've named this instance "a", but you can choose any name you would like.
- `--gpu` : This option is needed to signal to gdb4hpc that you want to use a GPU debugger
- `--gdb=/opt/rocm-4.1.1/bin/rocgdb` : This specifies the debugger that you intend to use. We currently need to use this now, but may change in the future to key off other active modules.
- `--launcher-args="…"` : These are options that are being passed to slurm to on how to run the job. The option –ntasks-per-node=1 is a requirement for debugging on AMD GPUs.
- `-a "…"` : This specifies the command line arguments that your program is expecting.
- *-i <<inputFile>>* : file to pass to you program as stdin (not used in the above example)

After launching a debugging session, you can start to add breakpoints and follow through the code as you would in gdb

GDB4HPC

Adding Breakpoints

```
dbg all> launch $a{2} --gpu --gdb=/opt/rocm-4.1.1/bin/rocgdb --
launcher-args="--ntasks-per-node=1 --mpi=pmi2 -p amdMI100" -a "-
n 10 10 10 -P 2 1 1" ./amg
Starting application, please wait...
Creating MRNet communication network...
Waiting for debug servers to attach to MRNet communications
network...
Timeout in 400 seconds. Please wait for the attach to complete.
Number of dbgsrvs connected: [1];  Timeout Counter: [0]
Number of dbgsrvs connected: [1];  Timeout Counter: [1]
Number of dbgsrvs connected: [2];  Timeout Counter: [0]
Finalizing setup...
Launch complete.
a{0..1}: Initial breakpoint, in
dbg all> break par_csr_communication.c:430
dbg all> continue
<$a>: MPI VERSION    : CRAY MPICH version 8.1.2.1 (ANL base
3.4a2)
<$a>: MPI BUILD INFO : Tue Feb 16 15:56 2021 (git hash 966808e)
(CH4)
<$a>: Running with these driver parameters:
…
```

```
a{0..1}: Breakpoint 2, hypre_ParCSRCommHandleCreate_v2 at
par_csr_communication.c:430
dbg all> list
a{0..1}: 430   printf("###KM in par_csr_communication.c | my_id=%d |
send_data - %p , recv_data - %p, num_requests - %d,
job=%d",my_id,send_data,recv_data,num_requests, job);
a{0..1}: 431   fflush(stdout);
a{0..1}: 432 #endif
a{0..1}: 433
a{0..1}: 434   j = 0;
a{0..1}: 435   switch (job)
a{0..1}: 436   {
a{0..1}: 437       case  1:
a{0..1}: 438       {
a{0..1}: 439         HYPRE_Complex *d_send_data = (HYPRE_Complex *)
send_data;
dbg all> info locals
a{0..1}: Name:job                    Type:HYPRE_Int
a{0..1}: Name:comm_pkg               Type:hypre_ParCSRCommPkg *
a{0..1}: Name:send_memory_location   Type:HYPRE_MemoryLocation
a{0..1}: Name:send_data_in           Type:void *
a{0..1}: Name:recv_memory_location   Type:HYPRE_MemoryLocation
a{0..1}: Name:recv_data_in           Type:void *
a{0..1}: Name:num_sends              Type:HYPRE_Int
a{0..1}: Name:comm                   Type:MPI_Comm
a{0..1}: Name:recv_data              Type:void *
a{0..1}: Name:my_id                  Type:HYPRE_Int
a{0..1}: Name:j                      Type:HYPRE_Int
```

GDB4HPC

Switching Ranks

Focus on rank 1

The MPI ranks have diverged (slightly) from one another in the code base.

Focus on rank 0

You can use regular-like expressions to focus a number of ranks. `Focus $all` can revert to the global-focus view that we start with.

```
a{1}: Program received signal SIGABRT.
a{1}: In raise at :0
…
dbg all> focus $a{1}
dbg a_temp> bt
a{1}: #12 main at /home/users/cmakrides/benchmarks/amg_april20201/AMG_new/amg.c:414
a{1}: #11 HYPRE_PCGSetup at HYPRE_pcg.c:34
a{1}: #10 hypre_PCGSetup at pcg.c:228
a{1}: #9  hypre_BoomerAMGSetup at par_amg_setup.c:1220
a{1}: #8  hypre_BoomerAMGCreate2ndS at par_strength.c:2931
a{1}: #7  hypre_BoomerAMGCreate2ndSDevice
a{1}: #6  hypre_MatvecCommPkgCreate at par_csr_communication.c:938
a{1}: #5  hypre_ParCSRCommPkgCreateApart at new_commpkg.c:585
a{1}: #4  hypre_ParCSRCommPkgCreateApart_core at new_commpkg.c:278
a{1}: #3  hypre_DataExchangeList at exchange_data.c:247
a{1}: #2  hypre_MPI_Irecv at mpistubs.c:1044
a{1}: #1  abort
a{1}: #0  raise
dbg a_temp> focus $a{0}
dbg a_temp> bt
a{0}: #9  main at /home/users/cmakrides/benchmarks/amg_april20201/AMG_new/amg.c:414
a{0}: #8  HYPRE_PCGSetup at HYPRE_pcg.c:34
a{0}: #7  hypre_PCGSetup at pcg.c:228
a{0}: #6  hypre_BoomerAMGSetup at par_amg_setup.c:1220
a{0}: #5  hypre_BoomerAMGCreate2ndS at par_strength.c:2931
a{0}: #4  hypre_BoomerAMGCreate2ndSDevice
a{0}: #3  hypre_MatvecCommPkgCreate at par_csr_communication.c:938
a{0}: #2  hypre_ParCSRCommPkgCreateApart at new_commpkg.c:585
a{0}: #1  hypre_ParCSRCommPkgCreateApart_core at new_commpkg.c:278
a{0}: #0  hypre_DataExchangeList at exchange_data.c:247
```

GDB4HPC

Final Tips

- The `source` command can sequentially enter in a number of commands given by a file.
- This is a good way to gather a list of break points and load them in.
- I find this is a good way to return to a point in debugging after doing a number of modifications

- You can shutdown a debugging instance by
  - Killing the individual instance: `kill $a`
  - Quitting gdb4hpc : `quit`

```
>$ cat gdb_input
launch $a{2} --gpu --gdb=/opt/rocm-4.1.1/bin/rocgdb --launcher-args="--ntasks-per-node=1 --mpi=pmi2 -p amdMI100" -a "-n 10 10 10 -P 2 1 1" ./amg
continue
break par_csr_communication.c:430
break exchange_data.c:247
>$ gdb4hpc
dbg a_temp> source gdb_input
Starting application, please wait...
Creating MRNet communication network...
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: [2];  Timeout Counter: [0]
Finalizing setup...
Launch complete.
a{0..1}: Initial breakpoint, in
a{0}: Breakpoint 1: file par_csr_communication.c, line 430.
a{0}: Breakpoint 2: file exchange_data.c, line 247.
…
dbg a_temp> kill $a
Shutting down debugger and killing application for 'a'.
```

OTHER DEBUGGING OPTIONS

OTHER DEBUGGING OPTIONS

More debugging tools

- rocgdb
  - Heavyweight debugger like gdb
- rocprof
  - Very similar to something like nvprof
  - Works with hip codes
  - Can export json files and open them up in chrome tracing (or another third-party visualization tool)
- Good ol' `write/printf`
  - Currently works for C/C++ in openMP target regions
  - Write/printf in target regions is not available yet in Fortran openMP offload ;_;

- AMD_LOG_LEVEL environment variable on AMD GPUs: https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-porting-guide.html#more-tips

Not exactly a tool for debugging

- Reveal – Can help the addition of adding openMP directives

THANK YOU

QUESTIONS?

Kostas Makrides
makrides@hpe.com