AMD Tools Overview

René van Oostrum, Scott Moe, Damon McDougall, Paul Bauman

Outline

- Tools for porting C or C++ CUDA codes to HIP
- Calling HIP from Fortran
- Profiling tools

AMD

Enter HIPify

- AMD provides 'Hipify' tools to automatically convert most CUDA code
  - Hipify-perl
  - Hipify-clang

- Good resource to help with porting: https://github.com/ROCm-Developer-Tools/HIP/blob/master/docs/markdown/hip_porting_guide.md

- In practice, large portions of many HPC codes have been automatically Hipified:
  - ~90% of CUDA code in CORAL-2 HACC
  - ~80% of CUDA code in CORAL-2 PENNANT
  - ~80% of CUDA code in CORAL-2 QMCPack
  - ~95% of CUDA code in CORAL-2 Laghos

The remaining code requires programmer intervention

Hipify tools

- Hipify-perl:
  - Easy to use –point at a directory and it will attempt to hipify CUDA code
  - Very simple string replacement technique: may make incorrect translations

    `sed -e ‘s/cuda/hip/g’` (e.g., cudaMemcpy becomes hipMemcpy)
  - Recommended for quick scans of projects

- Hipify-clang:
  - Requires CLANG compiler
  - More robust translation of the code
  - Uses clang to parse files and perform semantic translation
  - Can generate warnings and assistance for code for additional user analysis
  - High quality translation, particularly for cases where the user is familiar with the make system

Hipify-clang

- https://github.com/ROCm-Developer-Tools/HIP/tree/master/hipify-clang
- Build from source
- 'Hipification' requires same headers that would be needed to compile it with clang:

```
./hipify-clang foo.cu -I /usr/local/cuda-8.0/samples/common/inc
```

- Understands how to translate many CUDA libraries (cuBLAS, cuFFT, cuSPARSE, etc.)
- Will get useful warning messages about unknown conversions

```
[10:59:29][pabauman@fry:~/work/qmcpack/build/hipify]$ /home/pabauman/work/hip-testing/hipify-clang-install/bin/hi
pify-clang /home/pabauman/work/qmcpack/src/src/QMCWaveFunctions/EinsplineSetCuda.cpp -o-dir=. -examine -I/usr/inc
lude/libxml2 -I/usr/include/hdf5/serial -I/home/pabauman/work/qmcpack/src/src -I/home/pabauman/work/qmcpack/build
/src -I/home/pabauman/work/qmcpack/build/include -I/home/pabauman/work/qmcpack/src/external_codes/mpi_wrapper -I/
home/pabauman/work/qmcpack/src/external_codes/boost_multi -I/home/pabauman/work/qmcpack/src/external_codes/catch
-I/usr/lib/x86_64-linux-gnu/openmpi/include
/tmp/EinsplineSetCuda.cpp-9b0c60.hip:135:5: warning: CUDA identifier is unsupported in HIP.
    cudaMemPrefetchAsync(pos, 3 * N * sizeof(float), curr_gpu, spline_streams[devicenr]);
    ^
/tmp/EinsplineSetCuda.cpp-9b0c60.hip:183:5: warning: CUDA identifier is unsupported in HIP.
    cudaMemPrefetchAsync(pos, 3 * N * sizeof(float), curr_gpu, spline_streams[devicenr]);
    ^
/tmp/EinsplineSetCuda.cpp-9b0c60.hip:226:5: warning: CUDA identifier is unsupported in HIP.
    cudaMemPrefetchAsync(pos, 3 * N * sizeof(float), curr_gpu, spline_streams[devicenr]);
    ^
```

Thing to be aware of

- Be aware of the things hipifying can't handle:
  - Inline ptx assembly
    - Can either use inline GCN ISA, or convert it to HIP
  - Hipify-perl can't handle library calls, hipify-clang can handle library calls
  - Hard-coded warp size of 32
- Then focus on performance

Fortran + CUDA C/C++ -> Fortran + HIP C/C++

- The only difference here is that the CUDA C/C++ code is linked with some Fortran routines
- Assumption is these Fortran routines do not contain CUDA Fortran
- This behaves like you would expect:
  - hipify the CUDA
  - Compile your HIP C/C++ with hipcc
  - Compile your Fortran code
  - Link with hipcc
- Example scenario: your HIP C/C++ code makes calls to Fortran functions (e.g., LAPACK functions) on the host

CUDA Fortran -> Fortran + HIP C/C++

- There is no HIP equivalent to CUDA Fortran
- But HIP functions are callable from C, using `extern C`, so they can be called directly from Fortran
- The strategy here is:
  - **Manually port** CUDA Fortran code to HIP kernels in C++
  - Wrap the kernel launch in a C function
  - Call the C function from Fortran through Fortran 2003 C binding
- This strategy should be usable by Fortran users since it is standard conforming Fortran
- **This is not currently officially supported**
  - It is just here to show you what you can do now

AMD

Alternatives to using AMD GPUs from Fortran

- **WILL** provide OpenMP® 5.0 support for Fortran
- What are the options for writing HIP kernels with Fortran host code?

HIP with Fortran Strategy

- Idea is to use `interface` clause to `bind` to underlying C functions
  - CRITICAL to remember Fortran is always pass-by-reference
  - This has ramifications when declaring the function/subroutine in the interface clause
  - If you get it wrong: undefined behavior

- Will also make use of Fortran enumerator
  - C enums used frequently in HIP functions, helps preserve code readability and portability

AMD

Fortran Enumerators

- Unfortunately, cannot name enumerators and link directly to existing C enums
  - Must reproduce consistently with underlying C enum
  - Rules follow usual C rules for enums
    - Will increment by one for each entry in the list or can specify value directly
- Strategy in this talk is to put them in a separate Fortran module
  - Can `use` as needed in program and other modules

- Let’s go through an example

AMD

Fortran Enumerators

```
enum, bind(c)
   enumerator :: hipMemcpyHostToHost = 0, hipMemcpyHostToDevice = 1
   enumerator :: hipMemcpyDeviceToHost = 2, hipMemcpyDeviceToDevice = 3
   enumerator :: hipMemcpyDefault = 4
end enum
```

Fortran Enumerators

```fortran
enum, bind(c)
   enumerator :: hipSuccess = 0, hipErrorOutOfMemory = 2
   enumerator :: hipErrorNotInitialized = 3, hipErrorDeinitialized = 4
   enumerator :: hipErrorProfilerDisabled = 5, hipErrorProfilerNotInitialized = 6
   enumerator :: hipErrorProfilerAlreadyStarted = 7, hipErrorProfilerAlreadyStopped = 8
   enumerator :: hipErrorInvalidImage = 200
   enumerator :: hipErrorInvalidContext = 201, hipErrorContextAlreadyCurrent = 202
   enumerator :: hipErrorMapFailed = 205, hipErrorUnmapFailed = 206
   enumerator :: hipErrorArrayIsMapped = 207, hipErrorAlreadyMapped = 208
   enumerator :: hipErrorNoBinaryForGpu = 209, hipErrorAlreadyAcquired = 210
   enumerator :: hipErrorNotMapped = 211, hipErrorNotMappedAsArray = 212
   enumerator :: hipErrorNotMappedAsPointer = 213, hipErrorECCNotCorrectable = 214
   enumerator :: hipErrorUnsupportedLimit = 215, hipErrorContextAlreadyInUse = 216
   enumerator :: hipErrorPeerAccessUnsupported = 217, hipErrorInvalidKernelFile = 218
   enumerator :: hipErrorInvalidGraphicsContext = 219, hipErrorInvalidSource = 300
   enumerator :: hipErrorFileNotFound = 301, hipErrorSharedObjectSymbolNotFound = 302
   enumerator :: hipErrorSharedObjectInitFailed = 303, hipErrorOperatingSystem = 304
   enumerator :: hipErrorSetOnActiveProcess = 305, hipErrorInvalidHandle = 400
   enumerator :: hipErrorNotFound = 500, hipErrorIllegalAddress = 700
   enumerator :: hipErrorInvalidSymbol = 701, hipErrorMissingConfiguration = 1001
   enumerator :: hipErrorMemoryAllocation = 1002, hipErrorInitializationError = 1003
   enumerator :: hipErrorLaunchFailure = 1004, hipErrorPriorLaunchFailure = 1005
   enumerator :: hipErrorLaunchTimeOut = 1006, hipErrorLaunchOutOfResources = 1007
   enumerator :: hipErrorInvalidDeviceFunction = 1008, hipErrorInvalidConfiguration = 1009
   enumerator :: hipErrorInvalidDevice = 1010, hipErrorInvalidValue = 1011
   enumerator :: hipErrorInvalidDevicePointer = 1017, hipErrorInvalidMemcpyDirection = 1021
   enumerator :: hipErrorUnknown = 1030, hipErrorInvalidResourceHandle = 1033
   enumerator :: hipErrorNotReady = 1034, hipErrorNoDevice = 1038
   enumerator :: hipErrorPeerAccessAlreadyEnabled = 1050, hipErrorPeerAccessNotEnabled = 1051
   enumerator :: hipErrorRuntimeMemory = 1052, hipErrorRuntimeOther = 1053
   enumerator :: hipErrorHostMemoryAlreadyRegistered = 1061
   enumerator :: hipErrorHostMemoryNotRegistered = 1062, hipErrorMapBufferObjectFailed = 1071
   enumerator :: hipErrorTbd
end enum
```

HIP Function Interfaces

- Now put inside module, interface block

```fortran
module hip

  ! If we use hipenums here, then the user doesn't have to,
  ! they can just 'use hip'
  use hip_enums

  interface

     function hipDeviceSynchronize() bind(c, name="hipDeviceSynchronize")
       use iso_c_binding
       use hip_enums
       implicit none
       integer(kind(hipSuccess)) :: hipDeviceSynchronize
     end function hipDeviceSynchronize

  end interface
end module hip
```


AMD

HIP Function Interfaces

- Consider hipMemcpy

```
 * hipMemcpyAtoD, hipMemcpyAtoH, hipMemcpyAtoHAsync, hipMemcpyDtoA, hipMemcpyDtoD,
 * hipMemcpyDtoDAsync, hipMemcpyDtoH, hipMemcpyDtoHAsync, hipMemcpyHtoA, hipMemcpyHtoAAs
 * hipMemcpyHtoDAsync, hipMemFree, hipMemFreeHost, hipMemGetAddressRange, hipMemGetInfo,
 * hipMemHostAlloc, hipMemHostGetDevicePointer
 */
hipError_t hipMemcpy(void* dst, const void* src, size_t sizeBytes, hipMemcpyKind kind);
```

- Passing in an enum type, a void* and a size_t
- Function returns an enum
  - Could write as a Fortran subroutine, but lose error checking

HIP Function Interfaces

- Consider hipMemcpy

```
function hipMemcpy(dst,src,sizeBytes,cpykind) bind(c,name="hipMemcpy")
  use iso_c_binding
  use hip_enums
  implicit none
  integer(kind(hipSuccess)) :: hipMemcpy
  type(c_ptr),value :: dst
  type(c_ptr),value :: src
  integer(c_size_t), value :: sizeBytes

  ! We want to make sure we get the right integer for the enum
  integer(kind(hipMemcpyHostToHost)), value :: cpykind

end function hipMemcpy
```

- bind(c,name="function_name"), function_name must match C name
- Returns an integer(kind(hipSuccess))
  - Using our hip_enums module to define the kind of integer ensures portability with C enum type
- type(c_ptr) is Fortran 2003 interface for pointers
- Notice the `value` keyword which tells the compiler the pointer should be passed by value
- Can use `c_size_t` from iso_c_binding module to make sure we're portable with the integer size

Putting It All Together

- Convenience function for return code checking

```
subroutine hipCheck(hipError_t)
  use hip_enums

  implicit none

  integer(kind(hipSuccess)) :: hipError_t

  if(hipError_t /= hipSuccess)then
     write(*,*) "HIP ERROR: Error code = ", hipError_t
     call exit(hipError_t)
  end if

end subroutine hipCheck
```

- Note there is a `hipGetErrorString` function
  - Requires wrangling strings between C and Fortran

Putting It All Together

- Can allocate host memory as usual
- Device pointers are c_ptr

```
! Allocate host memory
allocate(a(N))
allocate(b(N))
allocate(out(N))

! Initialize host arrays
a(:) = 1.0
b(:) = 2.0
```

```
type(c_ptr) :: da = c_null_ptr
type(c_ptr) :: db = c_null_ptr
type(c_ptr) :: dout = c_null_ptr
```

- Allocate device memory and copy to device
- Note need to provide pointer location since we're passing pointer

```
! Allocate array space on the device
call hipCheck(hipMalloc(da,Nbytes))
call hipCheck(hipMalloc(db,Nbytes))
call hipCheck(hipMalloc(dout,Nbytes))

! Transfer data from host to device memory
call hipCheck(hipMemcpy(da, c_loc(a), Nbytes, hipMemcpyHostToDevice))
call hipCheck(hipMemcpy(db, c_loc(b), Nbytes, hipMemcpyHostToDevice))
```


AMD

Putting It All Together

- Kernel launch must be in C++
- Provide separate interface

```
call launch(dout,da,db,N)
```

```
interface
   subroutine launch(out,a,b,N) bind(c)
     use iso_c_binding
     implicit none
     type(c_ptr) :: a, b, out
     integer, value :: N
   end subroutine
end interface
```

Putting It All Together

- Kernel launch must be in C++
- Provide separate interface

```
#include <hip/hip_runtime.h>
#include <cstdio>

__global__ void vector_add(double *out, double *a, double *b, int n)
{
  size_t index = blockIdx.x * blockDim.x + threadIdx.x;
  size_t stride = blockDim.x * gridDim.x;

  for (size_t i = index; i < n; i += stride)
    out[i] = a[i] + b[i];
}


extern "C"
{
  void launch(double **dout, double **da, double **db, int N)
  {
    printf("launching kernel\n");
    hipLaunchKernelGGL((vector_add), dim3(320), dim3(256), 0, 0, *dout, *da, *db, N);
  }
}
```

Putting It All Together

- Synchronize and copy back
- Again, have to provide pointer location for host arrays

```
call hipCheck(hipDeviceSynchronize())

! Transfer data back to host memory
call hipCheck(hipMemcpy(c_loc(out), dout, Nbytes, hipMemcpyDeviceToHost))
```

- Free device and host memory

```
call hipCheck(hipFree(da))
call hipCheck(hipFree(db))
call hipCheck(hipFree(dout))

! Deallocate host memory
deallocate(a)
deallocate(b)
deallocate(out)
```


AMD

Putting It All Together

- Now build and run

```
gfortran -I../modules -c main.f03
hipcc -c hip_implementation.cpp
hipcc -lgfortran main.o hip_implementation.o ../modules/common.o -o fort_vector_add
```

  - gfortran on Fortran source
  - hipcc on C++ source with HIP kernels
    - If no HIP, could use regular g++
    - Note: hipcc is not really a compiler, just a wrapper around our compiler hcc
  - Need to use hipcc to link
    - Need to include -lgfortran

Closing Thoughts

- No formal support for CUDA Fortran (“HIP Fortran”)
- HIP is C-callable (extern “C”)
  - Can use F2003 C-binding to get a lot of the way there

- Limitations
  - Kernel code + launch must be in C++
  - Lose HIP portability layer to CUDA
    - HIP layer to CUDA on Nvidia hardware uses `static inline` functions
    - No symbols for Fortran interface to link against

AMD

AMD GPU Profiling: Currently in Flux

- What tools should I use?
- We are developing and supporting rocprofiler and roctracer
- The rocprofiler and roctracer libraries contain the central components allowing the collection of application traces and counters
  - **NOTE:** These libraries are currently under development
- The rocprofiler library comes with a command line tool, rocprof, to collect traces and counters.
  - The output of rocprof can be visualized using the chrome browser with chrome tracing

AMD GPU Profiling: rocprofiler and roctracer

- rocprofiler: A HW-specific low-level performance analysis interface allowing the collection of GPU hardware counters for compute applications written in GPU for OpenCL™ and ROCm/HSA/HIP
  - Documentation: https://github.com/ROCm-Developer-Tools/rocprofiler#user-content-profiling-utility-usage
  - Installation
    - From repo: **`sudo apt install rocprofiler-dev`**
    - From source: https://github.com/ROCm-Developer-Tools/rocprofiler#user-content-to-build-with-the-current-installed-rocm
  - Executable: **`/opt/rocm/bin/rocprof`**
- roctracer: A library containing tools for registering a generic runtime's API callbacks and asynchronous activity. When used with rocprofiler it allows the collection of GPU traces
  - Installation
    - From repo: **`sudo apt install roctracer-dev`**
    - From source: https://github.com/ROCm-Developer-Tools/roctracer#user-content-to-build-and-run-test
- roctracer works with rocprofiler.
  - To install both: **`sudo apt install rocprofiler-dev roctracer-dev`**
- **Note:** rocprofiler and roctracer can be used directly with the scripts discussed above, but they are meant to be used with higher level tools (such as Tau).

rocprofiler: Getting started + useful flags

- To get help:
  - `$ /opt/rocm/bin/rocprof -h`
- Useful housekeeping flags:
  - --timestamp <on|off> : turn on/off gpu kernel timestamps
  - --basenames <on|off>: turn on/off truncating gpu kernel names
  - -o <output csv file>: Direct counter information to a particular file name
  - -d <data directory>: Send profiling data to a particular directory
  - -t <temporary directory>: Change the directory where data files typically created in /tmp are placed. This allows you to save these temporary files.
- Flags directing rocprofiler activity:
  - -i input<.txt|.xml> - specify an input file (note the output files will now be named input.*)
  - --hsa-trace - to trace GPU Kernels and host HSA events (more later).
  - --hip-trace - to trace HIP API calls

rocprofiler: Collecting hardware counters

- rocprofiler can collect a number of hardware counters and derived counters
  - `$ /opt/rocm/bin/rocprof --list-basic`
  - `$ /opt/rocm/bin/rocprof --list-derived`
- Specify counters in a counter file. For example:
  - `$ /opt/rocm/bin/rocprof -i rocprof_counters.txt <app with args>`
  - `$ cat rocprof_counters.txt`

```
pmc : Wavefronts VALUInsts VFetchInsts VWriteInsts VALUUtilization VALUBusy WriteSize
pmc : SALUInsts SFetchInsts LDSInsts GDSInsts SALUBusy FetchSize
pmc : L2CacheHit MemUnitBusy MemUnitStalled WriteUnitStalled ALUStalledByLDS LDSBankConflict
pmc : TCC_HIT_sum TCC_MISS_sum TCC_EA_RDREQ_32B_sum TCC_EA_RDREQ_sum
pmc : TCC_EA_WRREQ_sum TCC_EA_WRREQ_64B_sum TCC_WRREQ_STALL_max
```

  - A limited number of counters can be collected during a specific pass of code.
    - Each line in the counter file will be collected in one pass
    - You will receive an error suggesting alternative counter ordering if you have too many counters on one line
  - A .csv file will be created by this command containing all of the requested counters

rocprofiler: Commonly Used Counters

- VALUUtilization: The percentage of ALUs active in a wave. Low VALUUtilization is likely due to high divergence or a poorly sized grid
- VALUBusy: The percentage of GPUTime vector ALU instructions are processed. Can be thought of as something like compute utilization.
- FETCH_SIZE: The total kilobytes fetched from DRAM
- L2CacheHit : The percentage of fetch, write, atomic, and other instructions that hit the data in L2 cache
- MemUnitBusy : The percentage of GPUTime the memory unit is active. The result includes the stall time
- MemUnitStalled : The percentage of GPUTime the memory unit is stalled

performance counters—things to look out for

- GPU Hardware counters are global
  - Kernel dispatches are serialized to ensure that only one dispatch is ever in flight
  - It is recommended that no other applications are running that use the GPU when collecting performance counters.

Other Tips

- Use "`--basenames on`" which will report only kernel names, leaving off kernel arguments.
- How do you time a kernel's duration?
  - `$ /opt/rocm/bin/rocprof --timestamps on -i rocprof_counters.txt <app with args>`
  - This produces four times: DispatchNs, BeginNs, EndNs, and CompleteNs
  - Closest thing to a kernel duration: EndNs - BeginNs
  - If you run with "`--stats`" the resultant results file will automatically include a column that calculates kernel duration
    - Note: the duration is aggregated over repeated calls to the same kernel

AMD

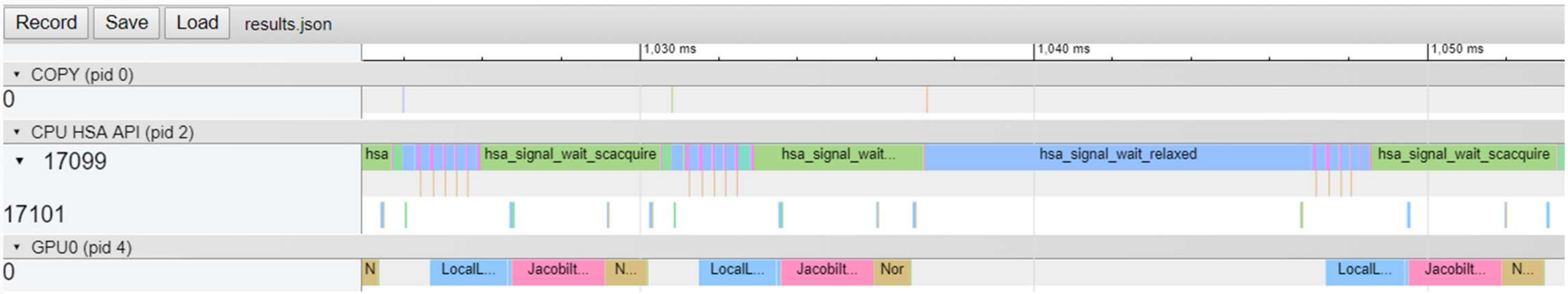
rocprofiler + roctracer: Collecting application traces

- rocprofiler can collect traces
  - “--hip-trace - to trace HIP, generates API execution stats and JSON file chrome-tracing compatible”
    - Basically collects HIP API calls.
  - “--hsa-trace - to trace HSA, generates API execution stats and JSON file chrome-tracing compatible”
    - CPU side:
      - Traces all HSA APIs called by the application
      - Collects CPU side timing data for these calls
    - GPU side:
      - Traces kernels dispatched to the GPU.
      - Traces data transfers (host to device and/or device to device).
      - Collects GPU side timing

AMD

rocprofiler + roctracer: Collecting application traces

- rocprofiler can collect traces
  - **`$ /opt/rocm/bin/rocprof --hsa-trace <app with arguments>`**
  - This will output a .json file that can be visualized using the chrome browser
  - Go to chrome://tracing and then load in the .json file.
    - The trace will display copies, hsa signals, and kernel calls.
    - It can handle multiple traces



rocprofiler + roctracer: Multiple MPI Ranks

- rocprofiler can collect counters and traces for multiple MPI ranks as long as it is told how to output data for each rank.
- Say you want to profile an application usually called like this:
  - `mpiexec –np <n> ./Jacobi_hip –g <x> <y>`
  - To obtain trace and counter information for each rank you should create a shell script (we can call it wrapper.sh) that calls rocprof:
  - ```
    #!/bin/bash
    rocprof -i counters_${OMPI_COMM_WORLD_RANK}.txt  --hsa-trace ./Jacobi_hip -g $1 $2
    ```
  - Then invoke the script by executing:

    `mpiexec -np <n> wrapper.sh <x> <y>`
- This will produce separate traces for each rank.
  - Note: roctracer doesn’t yet provide a way to combine the produced traces
- If all your ranks are on the same node, you can combine traces by involving rocprof like so:
  - `rocprof <rocprof options> mpiexec <mpi options> application <application arguments>`
    ```

Obtaining Occupancy Information

- Problem: rocprofiler can't be used to find VGPR or SGPR information currently.
- The extractkernel command summarizes information about the generated AMDGCN in a .bundle file. **vectoradd_hip.exe-000.bundle**:

vectoradd_hip.exe-000.bundle excerpt:

```
- Name:            width
        Size:            4
        Align:           4
        ValueKind:       ByValue
        ValueType:       I32
      - Name:            height
        Size:            4
        Align:           4
        ValueKind:       ByValue
        ValueType:       I32
    CodeProps:
      KernargSegmentSize: 32
      GroupSegmentFixedSize: 0
      PrivateSegmentFixedSize: 0
      KernargSegmentAlign: 8
      WavefrontSize:    64
      NumSGPRs:         14
      NumVGPRs:         7
      MaxFlatWorkGroupSize: 256
```


AMD

Interpreting the ISA

- Problem: Sometimes you need to see what the compiler has done in a key region of your kernel.
  - What if your kernel is huge?
- It is possible to produce an .isa file annotated with source code line numbers.
  - Pass the "`-gline-tables-only`" flag to hipcc
  - Set the environment variable KMDUMPISA=1
  - `$ /opt/rocm/hcc/bin/llvm-objdump -mcpu=gfx900 -source -line-numbers dump-gfx900.isabin > linenumbers.isa`

linenumbers.isa example excerpt:

```
00000000000002a0 BB0_2:
; /home/smoe/git/HIP-Examples/gpu-burn/BurnKernel.cpp:64
;            local_c += alpha * A[ idx_y + k * K] * B[ idx_x * K + k];
        global_load_dword v7, v[5:6], off                   // 0000000002A0: DC508000 077F0005
        global_load_dword v8, v[2:3], off                   // 0000000002A8: DC508000 087F0002
; /home/smoe/git/HIP-Examples/gpu-burn/BurnKernel.cpp:63
;          for(int k = 0; k < K; k++) {
        v_add_co_u32_e32 v5, vcc, 4, v5                     // 0000000002B0: 320A0A84
        v_addc_co_u32_e32 v6, vcc, 0, v6, vcc               // 0000000002B4: 380C0C80
        s_add_i32 s2, s2, -1                                // 0000000002B8: 8102C102
        v_mov_b32_e32 v9, s1                                // 0000000002BC: 7E120201
        v_add_co_u32_e32 v2, vcc, s0, v2                    // 0000000002C0: 32040400
        s_cmp_lg_u32 s2, 0                                  // 0000000002C4: BF078002
        v_addc_co_u32_e32 v3, vcc, v3, v9, vcc              // 0000000002C8: 38061303
```


AMD

AMD

QUESTIONS?

Disclaimer

DISCLAIMER
The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

AMD Compilers

- aocc
  - C/C++/Fortran compilers with optimizations for AMD CPUs
- hipcc
  - Script to wrap around nvcc or call AMD's internal HIP compiler
  - Needed to compile HIP device code, HIP API functions compatible with normal C++ compilers
- aomp
  - AMD OpenMP 5.0 Compiler
  - Compiles C/C++ code with OpenMP "target" pragmas
  - Links with libomptarget to produce a binary that can offload work to the GPU

All compilers are based on clang, compilers will be upstreamed to clang where possible.

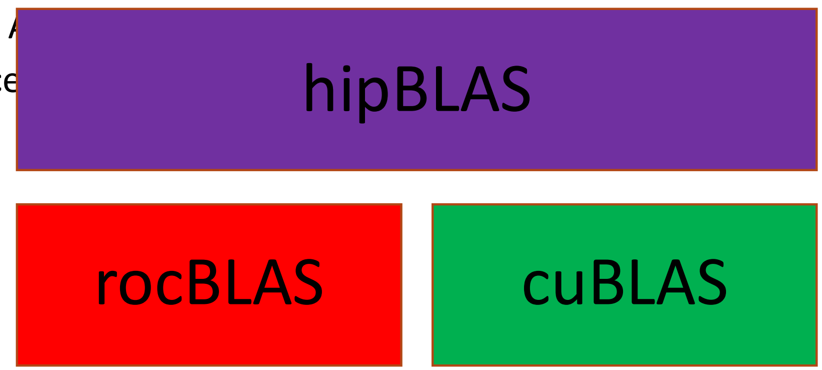
AMD

ROCm

- HIP is part of a larger software distribution called the Radeon Open Compute, or ROCm, Package
- Install instructions and documentation can be found here: https://githu
- The ROCm package provide libraries and programming tools for develop
  - rocminfo
  - rocm-smi
  - rocprof

ROCm GPU Libraries

- ROCm provides several GPU math libraries
  - Typically two versions:
    - roc* -> AMD GPU library, usually written in HIP
    - hip* -> Thin interface between roc* and Nvidia cu* library
  - When developing an application meant to target both CUDA and
  - When developing an application meant to target only AMD device



Some Links to Key Libraries

- BLAS
  - rocBLAS (https://github.com/ROCmSoftwarePlatform/rocBLAS)
  - hipBLAS (https://github.com/ROCmSoftwarePlatform/hipBLAS)
- FFTs
  - rocFFT (https://github.com/ROCmSoftwarePlatform/rocFFT)
- Random number generation
  - rocRAND (https://github.com/ROCmSoftwarePlatform/rocRAND)
  - hipRAND (https://github.com/ROCmSoftwarePlatform/hipRAND)
- Sparse linear algebra
  - rocSPARSE (https://github.com/ROCmSoftwarePlatform/rocSPARSE)
  - hipSPARSE (https://github.com/ROCmSoftwarePlatform/hipSPARSE)
- Iterative solvers
  - rocALUTION (https://github.com/ROCmSoftwarePlatform/rocALUTION)
- Parallel primitives
  - rocPRIM (https://github.com/ROCmSoftwarePlatform/rocPRIM)
  - hipCUB (https://github.com/ROCmSoftwarePlatform/hipCUB)

More links to key libraries

Machine Learning libraries and Frameworks

- Tensorflow: https://github.com/ROCmSoftwarePlatform/tensorflow-upstream
- Pytorch: https://github.com/ROCmSoftwarePlatform/pytorch
- MIOpen (similar to cuDNN): https://github.com/ROCmSoftwarePlatform/MIOpen
- Tensile: https://github.com/ROCmSoftwarePlatform/Tensile
- RCCL (ROCm analogue of NCCL): https://github.com/ROCmSoftwarePlatform/rccl

AMD

Extra Slides:
Current profiling tools, unsupported on Frontier:
CodeXL and rcprof

AMD GPU Profiling: Currently in Flux

- What tools should I use?
- Names you may have seen in our old documentation:
  - rocm-profiler
  - rocprofiler
  - roctracer
  - RCP
  - rocprof
  - rcprof
  - CodeXL
- Going forward we will be developing and supporting rocprofiler and roctracer
- The Radeon™ Compute Profiler (RCP) is another command line tool for collecting traces and counters. The binary is to run RCP is rcprof. The output from RCP can be visualized using CodeXL
- Going forward we will be developing and supporting rocprofiler and roctracer, not rcprof and CodeXL

AMD

RCP and CodeXL

- RCP: A command line tool for collecting hardware counters and application traces. The binary for RCP is rcprof.
  - Documentation: https://radeon-compute-profiler-rcp.readthedocs.io/en/latest/
  - Installation
    - From repo: **`sudo apt install rocm-profiler cxlactivitylogger`**
    - From source: https://github.com/GPUOpen-Tools/RCP
    - Executable: **`rcprof`**
- CodeXL: A GUI application for visualizing the output of RCP.
  - Documentation: https://github.com/GPUOpen-Tools/CodeXL
  - Installation:
    - From repo: **`sudo apt install codexl`**
    - From source:https://github.com/GPUOpen-Tools/CodeXL/releases
- Activity Logger: A library that allows users to instrument code with annotations that can be displayed in CodeXL.
  - Documentation: https://github.com/GPUOpen-Tools/common-src-AMDTActivityLogger

rcprof: application trace mode

- Getting usage info
  - `rcprof`

- Getting ROCm/HSA/HIP application traces:
  - `rcprof --hsatrace <application with arguments>`
  - `rcprof -A <application with arguments>`

- CPU-side trace
  - Traces all HSA APIs called by the application (function name, return value, argument values)
  - Collects CPU-side timing data for all API calls

- GPU-side trace
  - Traces all kernels dispatched to the GPU
  - Traces all data transfers between devices (host<->device, device<->device)
  - Collects GPU-side timing data for both of the above

AMD

rcprof: application trace mode

- Default output file is `~/apitrace.atp`
  - Can be overridden with `--outputfile filename or -o filename`

- Default working directory is ~
  - Can be overridden with `--workingdirectory dir` or `-w dir`

- Summary info (add `--tracesummary` or `–T` to above command lines)
  - Generates HTML Summary pages providing
    - API Usage Warnings/Errors
    - Summary of all APIs called (# of times, total elapsed time)
    - Kernel Dispatch Summary and Top 10 Kernel Dispatches
    - Top 10 Data Transfers

- ActivityLogger instrumentation
  - The ActivityLogger is a library that allows you to instrument your code with annotations that can appear in the CodeXL timeline viewer
  - It's a good way to "fill the gaps" in the timeline
  - It's can also be a good way to correlate user code to HSA-specific events in the timeline

rcprof: performance counter mode

- ROCm/HSA
  - `rcprof --hsapmc <application with arguments>`
  - `rcprof -C <application with arguments>`

- Kernel dispatch statistics
  - Kernel Name, Global Grid Size, Work Group Size, LDS, VGPR and SGPR usage
- Default output file is ~/Session1.csv
  - Can override output file name and location using `--outputfile filename`
  - or `-o filename`
- Single-pass performance counters
  - List Available Counters:
  - `rcprof --list (-l) or rcprof --listdetailed (-L)`
  - Available Counters: Wavefronts, VALUInsts, SALUInsts, VFetchInsts, SFetchInsts, VWriteInsts, FlatVMemInsts, LDSInsts, FlatLDSInsts, GDSInsts, VALUUtilization, VALUBusy, SALUBusy, FetchSize, WriteSize, L2CacheHit, MemUnitBusy, MemUnitStalled, WriteUnitStalled, LDSBankConflict

rcprof: specifying performance counters and output files

- Profile using default counter set
  - `rcprof --hsapmc (-C) <application with arguments>`
  - The profiler will enable as many counters as possible that will fit into a single pass (varies by hardware generation)
- Specify counters using `--counterfile` or `-c`
  - `rcprof -C --counterfile counterfile.txt <application with args>`
  - The argument to `--counterfile` is a file name. That file should contain one counter name per line.
- Check number of passes required
  - `rcprof --counterfile counterfile.txt --numberofpass`
- Generate counter files
  - `rcprof --list --outputfile counterfile.txt`

- Generate single-pass counter files
  - `rcprof --list --outputfile counterfile.txt --maxpassperfile 1`
  - Generates "counterfile_pass1.txt", "counterfile_pass2.txt", etc.
  - A set of single-pass counter files is generated in /opt/rocm/profiler/counterfiles when installing the profiler using the Debian package. If generation fails, there will be a text files containing information on how to generate these manually.

rcprof: limiting profiling data

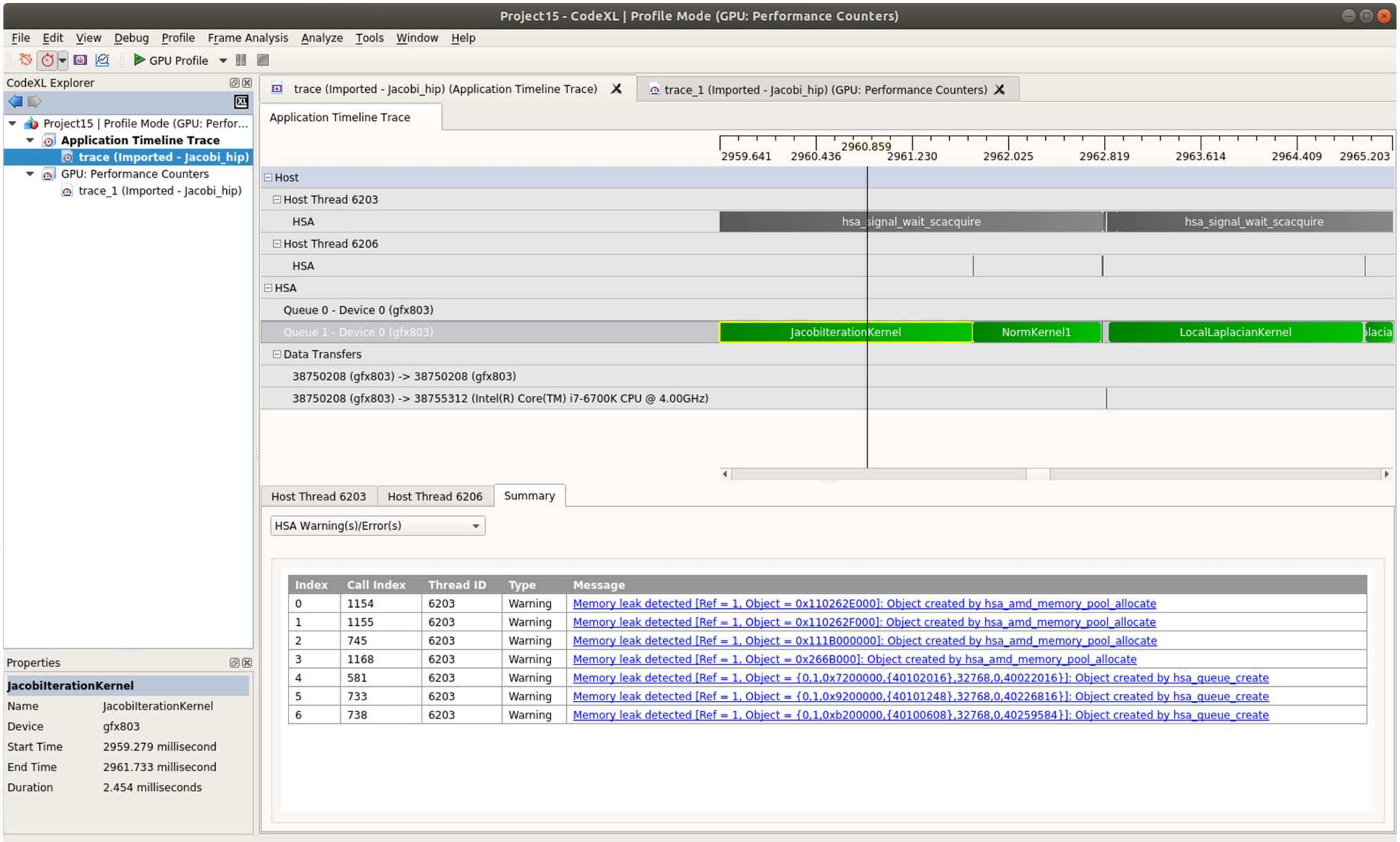
- General
  - Limiting profile duration
    - `rcprof --startdelay X`
      - Runs the application, but doesn't start collecting profile data until the specified start delay (in ms) passes
    - `rcprof --profileduration X`
      - Runs the application and stops collecting profile data after the specified duration (in ms) passes
    - `rcprof --startdisabled`
      - Run the application but doesn't collect any profile data.
      - Can be used in conjunction with instrumenting an application with the ActivityLogger
  - ActivityLogger instrumentation
    - The ActivityLogger library can be used to instrument an application to control which parts of an application generate profile data.
    - `amdtStopProfiling`, `amdtResumeProfiling`

CodeXL: viewing profiling data

- Steps to import a profiler session file

1. Start CodeXL
2. Create a new project using `File>New Project` menu item
   - Accept all default settings and options
3. Switch to Profile Mode using `Profile>Switch to Profile Mode` menu item
4. Switch to GPU profiling using `Profile>GPU: Performance Counters` or `Profile>Application Timeline Trace` (either of the two works)
5. In the CodeXL Explorer window, right click and select `Import Session` menu item
6. Navigate to the location of the .atp file or .csv file that you want to import
   - The file will be imported and the data will be displayed.

AMD


Project15 - CodeXL | Profile Mode (GPU: Performance Counters)
File Edit View Debug Profile Frame Analysis Analyze Tools Window Help
GPU Profile
CodeXL Explorer
Project15 | Profile Mode (GPU: Perfor...
Application Timeline Trace
trace (Imported - Jacobi_hip)
GPU: Performance Counters
trace_1 (Imported - Jacobi_hip)
trace (Imported - Jacobi_hip) (Application Timeline Trace)
trace_1 (Imported - Jacobi_hip) (GPU: Performance Counters)
Application Timeline Trace
2959.641 2960.436 2960.859 2961.230 2962.025 2962.819 2963.614 2964.409 2965.203
Host
Host Thread 6203
HSA
hsa_signal_wait_scacquire
hsa_signal_wait_scacquire
Host Thread 6206
HSA
HSA
Queue 0 - Device 0 (gfx803)
Queue 1 - Device 0 (gfx803)
JacobiIterationKernel
NormKernel1
LocalLaplacianKernel
Data Transfers
38750208 (gfx803) -> 38750208 (gfx803)
38750208 (gfx803) -> 38755312 (Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz)
Host Thread 6203
Host Thread 6206
Summary
HSA Warning(s)/Error(s)
Index | Call Index | Thread ID | Type | Message
0 | 1154 | 6203 | Warning | Memory leak detected [Ref = 1, Object = 0x110262E000]: Object created by hsa_amd_memory_pool_allocate
1 | 1155 | 6203 | Warning | Memory leak detected [Ref = 1, Object = 0x110262F000]: Object created by hsa_amd_memory_pool_allocate
2 | 745 | 6203 | Warning | Memory leak detected [Ref = 1, Object = 0x111B000000]: Object created by hsa_amd_memory_pool_allocate
3 | 1168 | 6203 | Warning | Memory leak detected [Ref = 1, Object = 0x266B000]: Object created by hsa_amd_memory_pool_allocate
4 | 581 | 6203 | Warning | Memory leak detected [Ref = 1, Object = {0,1,0x7200000,{40102016},32768,0,40022016}]: Object created by hsa_queue_create
5 | 733 | 6203 | Warning | Memory leak detected [Ref = 1, Object = {0,1,0x9200000,{40101248},32768,0,40226816}]: Object created by hsa_queue_create
6 | 738 | 6203 | Warning | Memory leak detected [Ref = 1, Object = {0,1,0xb200000,{40100608},32768,0,40259584}]: Object created by hsa_queue_create
Properties
JacobiIterationKernel
Name JacobiIterationKernel
Device gfx803
Start Time 2959.279 millisecond
End Time 2961.733 millisecond
Duration 2.454 milliseconds

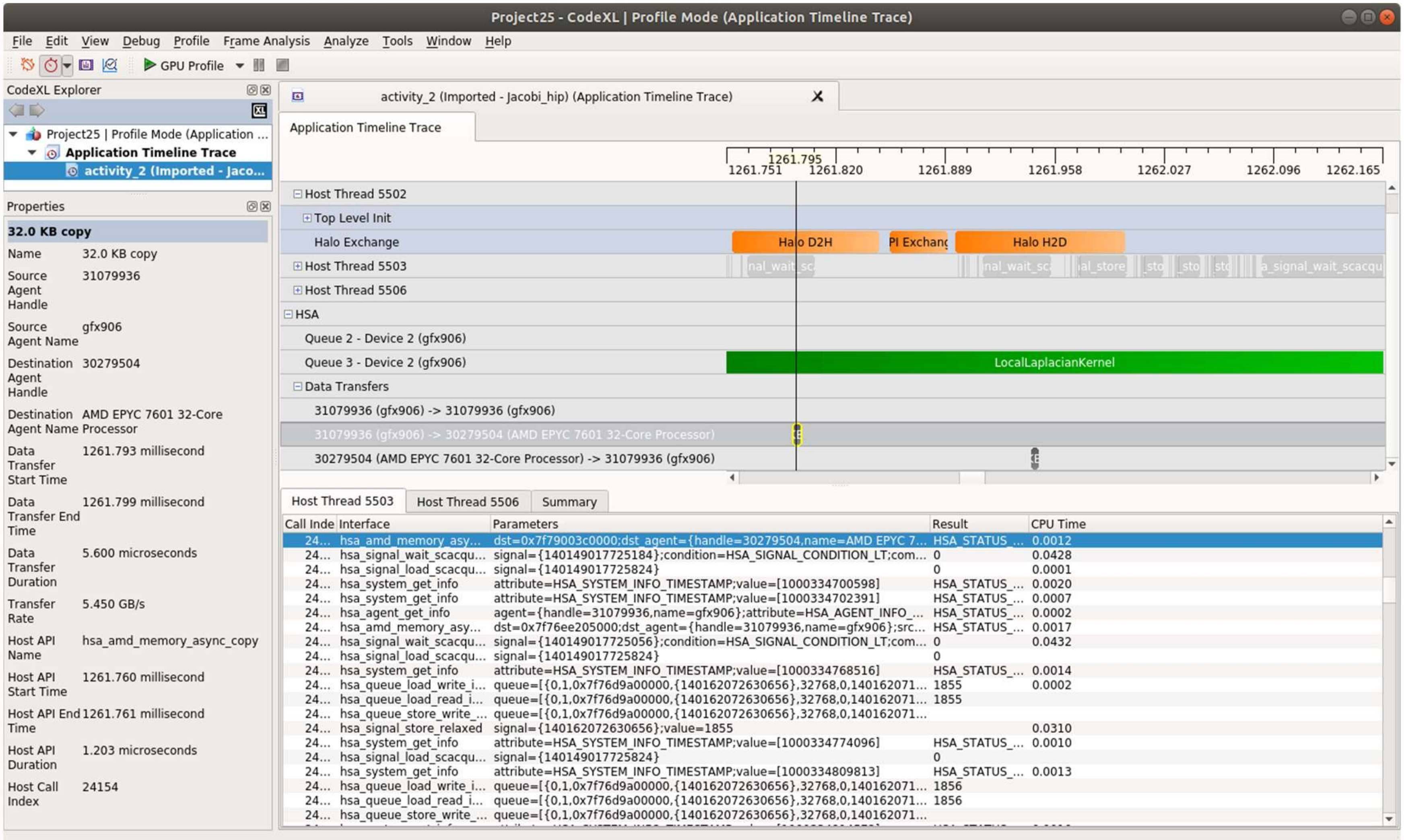
AMD

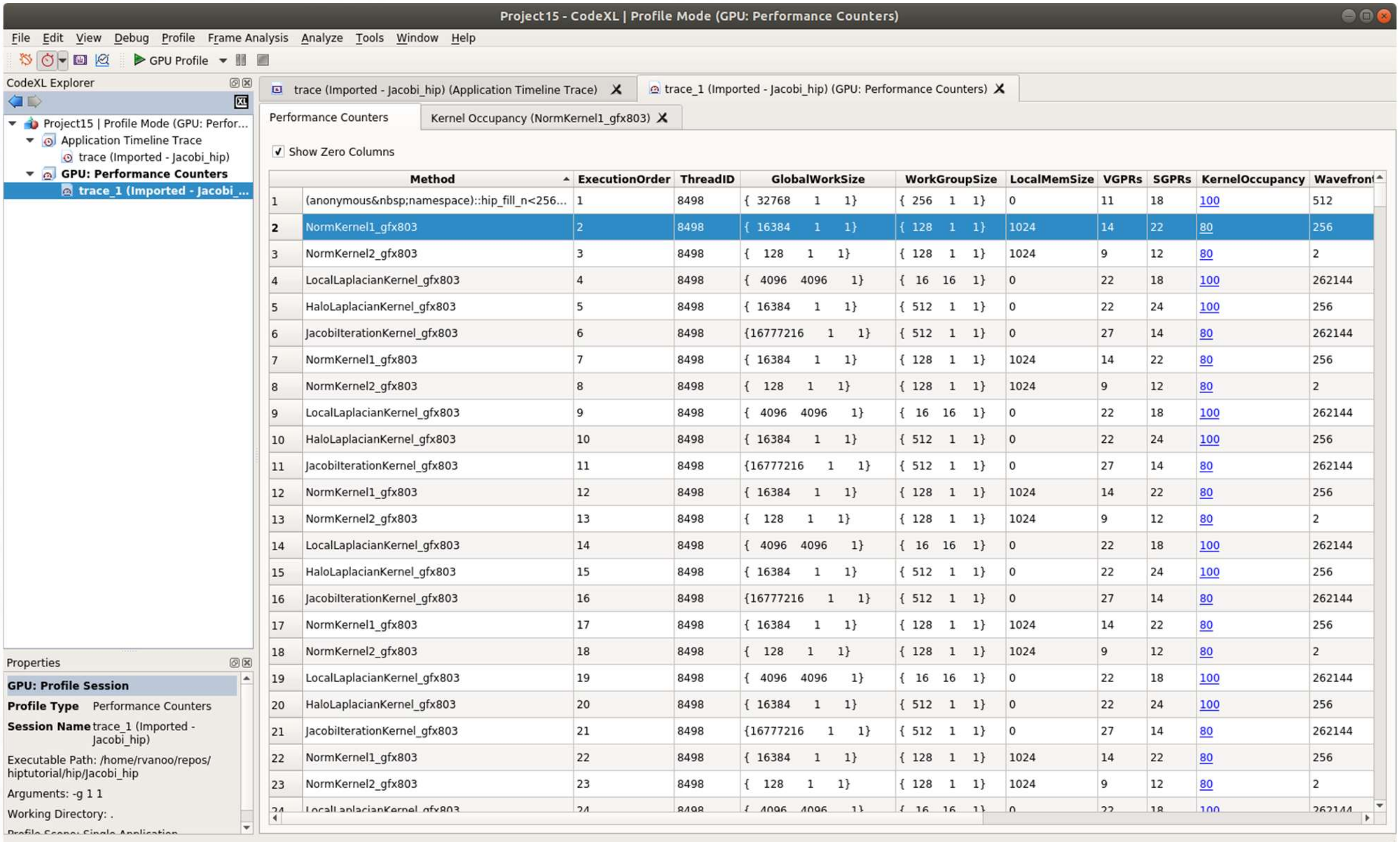
Annotating the timeline with the ActivityLogger

```
void HaloExchange(grid_t& grid, mesh_t& mesh,
                  hipStream_t stream, dfloat* d_U) {

  //copy each side to the haloBuffer
  amdtBeginMarker("Halo D2H", "Halo Exchange", "");
  if (grid.Neighbor[SIDE_DOWN]>-1)
    SafeHipCall(hipMemcpy2DAsync(mesh.sendBuffer + mesh.sideOffset[SIDE_DOWN],
                                 mesh.Nx*sizeof(dfloat),
                                 d_U,
                                 mesh.Nx*sizeof(dfloat),
                                 mesh.Nx*sizeof(dfloat), 1,
                                 hipMemcpyDeviceToHost, stream));
  // more code, omitted
  // wait for the data to arrive on host
  hipStreamSynchronize(stream);
  amdtEndMarker();

  //post recvs & sends
  amdtBeginMarker("MPI Exchange", "Halo Exchange", "");
  for (int s=0;s<NSIDES;s++) {
    if (grid.Neighbor[s]>-1) {
      MPI_Irecv(mesh.recvBuffer + mesh.sideOffset[s], mesh.sideLength[s], MPI_DFLOAT,
                grid.Neighbor[s], 0, grid.comm, mesh.requests+2*s);
      MPI_Isend(mesh.sendBuffer + mesh.sideOffset[s], mesh.sideLength[s], MPI_DFLOAT,
                grid.Neighbor[s], 0, grid.comm, mesh.requests+2*s+1);
    }
  }
 // more code, omitted
```


Project25 - CodeXL | Profile Mode (Application Timeline Trace)
File Edit View Debug Profile Frame Analysis Analyze Tools Window Help
GPU Profile
CodeXL Explorer
Project25 | Profile Mode (Application ...
Application Timeline Trace
activity_2 (Imported - Jaco...
Properties
32.0 KB copy
Name 32.0 KB copy
Source Agent Handle 31079936
Source Agent Name gfx906
Destination Agent Handle 30279504
Destination Agent Name AMD EPYC 7601 32-Core Processor
Data Transfer Start Time 1261.793 millisecond
Data Transfer End Time 1261.799 millisecond
Data Transfer Duration 5.600 microseconds
Transfer Rate 5.450 GB/s
Host API Name hsa_amd_memory_async_copy
Host API Start Time 1261.760 millisecond
Host API End Time 1261.761 millisecond
Host API Duration 1.203 microseconds
Host Call Index 24154
activity_2 (Imported - Jacobi_hip) (Application Timeline Trace)
Application Timeline Trace
1261.795
1261.751 1261.820 1261.889 1261.958 1262.027 1262.096 1262.165
Host Thread 5502
Top Level Init
Halo Exchange
Halo D2H
PI Exchang
Halo H2D
Host Thread 5503
Host Thread 5506
HSA
Queue 2 - Device 2 (gfx906)
Queue 3 - Device 2 (gfx906)
LocalLaplacianKernel
Data Transfers
31079936 (gfx906) -> 31079936 (gfx906)
31079936 (gfx906) -> 30279504 (AMD EPYC 7601 32-Core Processor)
30279504 (AMD EPYC 7601 32-Core Processor) -> 31079936 (gfx906)
Host Thread 5503
Host Thread 5506
Summary
Call Inde Interface Parameters Result CPU Time
24... hsa_amd_memory_asy... dst=0x7f79003c0000;dst_agent={handle=30279504,name=AMD EPYC 7... HSA_STATUS_... 0.0012
24... hsa_signal_wait_scacqu... signal={140149017725184};condition=HSA_SIGNAL_CONDITION_LT;com... 0 0.0428
24... hsa_signal_load_scacqu... signal={140149017725824} 0 0.0001
24... hsa_system_get_info attribute=HSA_SYSTEM_INFO_TIMESTAMP;value=[1000334700598] HSA_STATUS_... 0.0020
24... hsa_system_get_info attribute=HSA_SYSTEM_INFO_TIMESTAMP;value=[1000334702391] HSA_STATUS_... 0.0007
24... hsa_agent_get_info agent={handle=31079936,name=gfx906};attribute=HSA_AGENT_INFO_... HSA_STATUS_... 0.0002
24... hsa_amd_memory_asy... dst=0x7f76ee205000;dst_agent={handle=31079936,name=gfx906};src... HSA_STATUS_... 0.0017
24... hsa_signal_wait_scacqu... signal={140149017725056};condition=HSA_SIGNAL_CONDITION_LT;com... 0 0.0432
24... hsa_signal_load_scacqu... signal={140149017725824} 0
24... hsa_system_get_info attribute=HSA_SYSTEM_INFO_TIMESTAMP;value=[1000334768516] HSA_STATUS_... 0.0014
24... hsa_queue_load_write_i... queue=[{0,1,0x7f76d9a00000,{140162072630656},32768,0,140162071... 1855 0.0002
24... hsa_queue_load_read_i... queue=[{0,1,0x7f76d9a00000,{140162072630656},32768,0,140162071... 1855
24... hsa_queue_store_write_... queue=[{0,1,0x7f76d9a00000,{140162072630656},32768,0,140162071...
24... hsa_signal_store_relaxed signal={140162072630656};value=1855 0.0310
24... hsa_system_get_info attribute=HSA_SYSTEM_INFO_TIMESTAMP;value=[1000334774096] HSA_STATUS_... 0.0010
24... hsa_signal_load_scacqu... signal={140149017725824} 0
24... hsa_system_get_info attribute=HSA_SYSTEM_INFO_TIMESTAMP;value=[1000334809813] HSA_STATUS_... 0.0013
24... hsa_queue_load_write_i... queue=[{0,1,0x7f76d9a00000,{140162072630656},32768,0,140162071... 1856
24... hsa_queue_load_read_i... queue=[{0,1,0x7f76d9a00000,{140162072630656},32768,0,140162071... 1856
24... hsa_queue_store_write_... queue=[{0,1,0x7f76d9a00000,{140162072630656},32768,0,140162071...



	Method	ExecutionOrder	ThreadID	GlobalWorkSize	WorkGroupSize	LocalMemSize	VGPRs	SGPRs	KernelOccupancy	Wavefront
1	(anonymous namespace)::hip_fill_n<256...	1	8498	{ 32768 1 1}	{ 256 1 1}	0	11	18	100	512
2	NormKernel1_gfx803	2	8498	{ 16384 1 1}	{ 128 1 1}	1024	14	22	80	256
3	NormKernel2_gfx803	3	8498	{ 128 1 1}	{ 128 1 1}	1024	9	12	80	2
4	LocalLaplacianKernel_gfx803	4	8498	{ 4096 4096 1}	{ 16 16 1}	0	22	18	100	262144
5	HaloLaplacianKernel_gfx803	5	8498	{ 16384 1 1}	{ 512 1 1}	0	22	24	100	256
6	JacobiIterationKernel_gfx803	6	8498	{16777216 1 1}	{ 512 1 1}	0	27	14	80	262144
7	NormKernel1_gfx803	7	8498	{ 16384 1 1}	{ 128 1 1}	1024	14	22	80	256
8	NormKernel2_gfx803	8	8498	{ 128 1 1}	{ 128 1 1}	1024	9	12	80	2
9	LocalLaplacianKernel_gfx803	9	8498	{ 4096 4096 1}	{ 16 16 1}	0	22	18	100	262144
10	HaloLaplacianKernel_gfx803	10	8498	{ 16384 1 1}	{ 512 1 1}	0	22	24	100	256
11	JacobiIterationKernel_gfx803	11	8498	{16777216 1 1}	{ 512 1 1}	0	27	14	80	262144
12	NormKernel1_gfx803	12	8498	{ 16384 1 1}	{ 128 1 1}	1024	14	22	80	256
13	NormKernel2_gfx803	13	8498	{ 128 1 1}	{ 128 1 1}	1024	9	12	80	2
14	LocalLaplacianKernel_gfx803	14	8498	{ 4096 4096 1}	{ 16 16 1}	0	22	18	100	262144
15	HaloLaplacianKernel_gfx803	15	8498	{ 16384 1 1}	{ 512 1 1}	0	22	24	100	256
16	JacobiIterationKernel_gfx803	16	8498	{16777216 1 1}	{ 512 1 1}	0	27	14	80	262144
17	NormKernel1_gfx803	17	8498	{ 16384 1 1}	{ 128 1 1}	1024	14	22	80	256
18	NormKernel2_gfx803	18	8498	{ 128 1 1}	{ 128 1 1}	1024	9	12	80	2
19	LocalLaplacianKernel_gfx803	19	8498	{ 4096 4096 1}	{ 16 16 1}	0	22	18	100	262144
20	HaloLaplacianKernel_gfx803	20	8498	{ 16384 1 1}	{ 512 1 1}	0	22	24	100	256
21	JacobiIterationKernel_gfx803	21	8498	{16777216 1 1}	{ 512 1 1}	0	27	14	80	262144
22	NormKernel1_gfx803	22	8498	{ 16384 1 1}	{ 128 1 1}	1024	14	22	80	256
23	NormKernel2_gfx803	23	8498	{ 128 1 1}	{ 128 1 1}	1024	9	12	80	2
24	LocalLaplacianKernel_gfx803	24	8498	{ 4096 4096 1}	{ 16 16 1}	0	22	18	100	262144