OAK RIDGE National Laboratory

Developing for Frontier using HIP on Spock

Balint Joo – OLCF

Oak Ridge Leadership Computing Facility
User Meeting 2021 (virtual)

Thursday, June 24, 2021

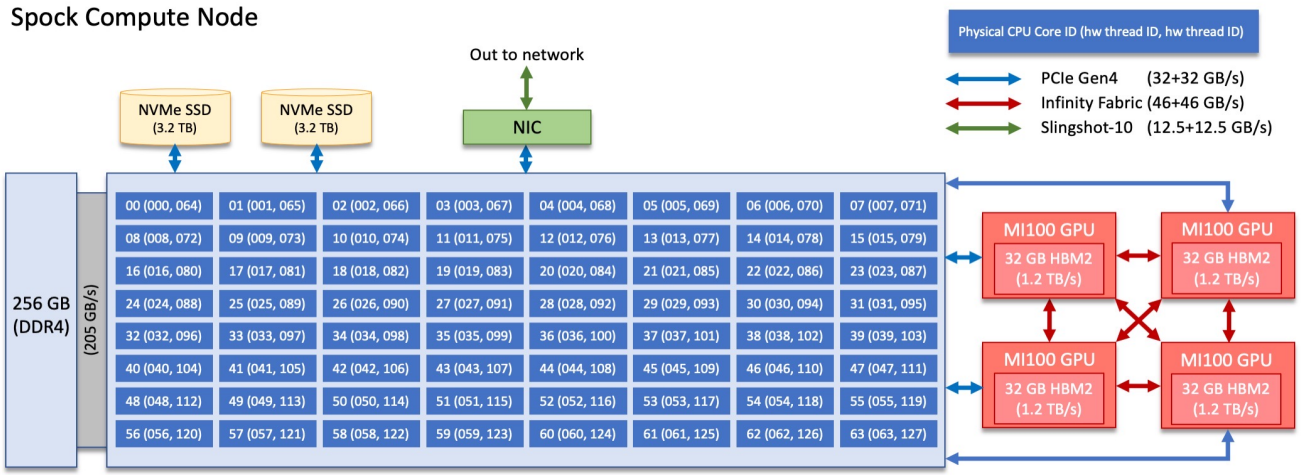
joob AT ornl.gov

Outline

- My Focus: HIP and C++ using AMD/ROCm compilers
  - I have not had much experience with the current HPE Wrappers yet
- The Spock System
  - brief overview
  - modules
  - simple tests/interactive running
- Tools:
  - CMake additions
  - Rocgdb
  - Rocprof

Spock: OLCF's Frontier Early Access System (EAS)

- 3 Cabinets w. 12 nodes/cabinet => 36 nodes, 144 GPUs total
- Each node has
  - 1x 64 core AMD EPYC 7662 "Rome" CPU (4 NUMA domains) + 256 GB DDR4 memory (205 GB/s)
  - 4x AMD Radeon Instinct MI100 GPUs (gfx908), 32 GB HBM2 @ (1.2 TB/sec)
    - GPU-GPU: All-to-all Infinity Fabric Interconnect, Host-GPU: PCie Gen4: 32+32 GB/sec
  - 2 x NVMe SSDs (3.2 TB each): 6800 MB/sec read, 4200 MB.sec write
- Slingshot 10 Interconnect: 12.5 + 12.5 GB/sec
- Spock Training Workshop materials: https://www.olcf.ornl.gov/spock-training
- Documentation: https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html



Original Spock Compute Node

Modules needed for ROCm/HIP development

- To use hipcc and other rocm tools: `module load rocm/4.2.0`
- To use a newer version of CMake (3.20): `module load cmake`
- To use a GPU aware MPI:
  - `module load craype-accel-amd-gfx908`
  - `export MPIR_CVAR_GPU_EAGER_DEVICE_MEM=0`
  - `export MPICH_GPU_SUPPORT_ENABLED=1`
  - `export GPU_TARGET=gfx908`
- To link against MPI
  - `MPI_CFLAGS="-I${MPICH_DIR}/include"`
  - `MPI_LDFLAGS="-L${MPICH_DIR}/lib -lmpi -L/opt/cray/pe/mpich/8.1.4/gtl/lib -lmpi_gtl_hsa”`
- Command line:
  - `hipcc ${MPI_CFLAGS} –o app app.cpp ${MPI_LDFLAGS}`
- Cmake builds: set CMake variables as (using –D on command line or in GUI)
  - `CMAKE_CXX_COMPILER=hipcc` and/or `CMAKE_C_COMPILER=hipcc`
  - `CMAKE_CXX_FLAGS=“${MPI_CFLAGS}”` and/or `CMAKE_C_FLAGS=“${MPI_CFLAGS}”`
  - `CMAKE_EXE_LINKER_FLAGS=“${MPI_LDFLAGS}”`
  - if using shared libs `CMAKE_SHARED_LINKER_FLAGS=”${MPI_LDFLAGS}”`

Modules needed for ROCm/HIP development

- To use hipcc and other rocm tools: `module load rocm/4.2.0`
- To use a newer version of CMake (3.20): `module load cmake`
- To use a GPU aware MPI:
  - `module load craype-accel-amd-gfx908`
  - `export MPIR_CVAR_GPU_EAGER_DEVICE_MEM=0`
  - `export MPICH_GPU_SUPPORT_ENABLED=1`
  - `export GPU_TARGET=gfx908`
- To link against MPI
  - `MPI_CFLAGS="-I${MPICH_DIR}/include"`
  - `MPI_LDFLAGS="-L${MPICH_DIR}/lib -lmpi -L/opt/cray/pe/mpich/8.1.4/gtl/lib -lmpi_gtl_hsa"`
- Command line:
  - `hipcc ${MPI_CFLAGS} –o app app.cpp ${MPI_LDFLAGS}`
- Cmake builds: set CMake variables as (using –D on command line or in GUI)
  - `CMAKE_CXX_COMPILER=hipcc` and/or `CMAKE_C_COMPILER=hipcc`
  - `CMAKE_CXX_FLAGS="${MPI_CFLAGS}"` and/or `CMAKE_C_FLAGS="${MPI_CFLAGS}"`
  - `CMAKE_EXE_LINKER_FLAGS="${MPI_LDFLAGS}"`
  - if using shared libs `CMAKE_SHARED_LINKER_FLAGS="${MPI_LDFLAGS}"`

Gotcha:
- hipcc automatically adds `–std=c++11` flag currently, even when compiling C code.

Suppress by:
- `adding –std=c99` (or any explicit standard level).
- defining CMake Variable `C_STANDARD=99`
- or just simply don't use hipcc to compile C-code.

Running Interactive Single node, Single Device jobs

- Frequently used when you may want to profile, check things, or profile and debug single device code
- Easiest without MPI for single device testing:

```
# Sample session
$ salloc -p ecp -A <Account> -t hh:mm:sss -N 1 --exclusive
$ srun --pty bash
$ rocminfo    # should see all the devices now
# Now run your test
$ ./my_app --gtest_filter=Tests:myFailingTest
#
# Now debug it
$ rocgdb --args ./my_app --gtest_filter=Tests:MyFailingTest
```

HIP and CMake

- HIP Language Level CMake features coming “real soon now”
- Until then one can use find_package()
- ROCm provides CMake helpers
- HIP CMake support is evolving so a bit of a moving target ATM. Expect changes!

```
# Get ROCm CMake Helpers onto your CMake Module Path
if (NOT DEFINED ROCM_PATH )
  if (NOT DEFINED ENV{ROCM_PATH} )
    set(ROCM_PATH "/opt/rocm" CACHE PATH "ROCm path")
  else()
    set(ROCM_PATH $ENV{ROCM_PATH} CACHE PATH "ROCm path")
  endif()
endif()
set(CMAKE_MODULE_PATH "${ROCM_PATH}/lib/cmake" ${CMAKE_MODULE_PATH})

# Set GPU Targets and Find all the HIP modules
set(GPU_TARGETS “gfx906;gfx908” CACHE STRING "The GPU TARGETs" )
find_package(HIP REQUIRED)
find_package(hipfft REQUIRED)
find_package(hiprand REQUIRED)
find_package(rocrand REQUIRED)
find_package(hipblas REQUIRED)
find_package(rocblas REQUIRED)
find_package(hipcub REQUIRED)
find_package(rocprim REQUIRED)

set( MY_HIP_SRCS my_hip_src1.cpp my_hip_src2.cpp my_hip_src3.cpp)

# Mark source files as HIP.  I guess in the future just a
# LANGUAGE HIP property will suffice. For now do it via compile flags
set_source_files_properties( ${MY_HIP_SRCS} PROPERTIES LANGUAGE CXX)
set_source_files_properties( ${MY_HIP_SRCS} PROPERTIES
                             COMPILE_FLAGS "-x hip”)

# Create a Library dependent on HIP
add_library( myLib ${MY_HIP_SRCS} )
target_include_directories(myLib PUBLIC ${ROCM_PATH}/hipfft/include)
target_link_libraries(myLib PUBLIC
    hip::hiprand roc::rocrand
    hip::hipfft
    roc::hipblas roc::rocblas
    hip::hipcub roc::rocprim_hip )
```

HIP and CMake

- HIP Language Level CMake features coming “real soon now”
- Until then one can use find_package()
- ROCm provides CMake helpers
- HIP CMake support is evolving so a bit of a moving target ATM. Expect changes!

```
# Get ROCm CMake Helpers onto your CMake Module Path
if (NOT DEFINED ROCM_PATH )
  if (NOT DEFINED ENV{ROCM_PATH} )
    set(ROCM_PATH "/opt/rocm" CACHE PATH "ROCm path")
  else()
    set(ROCM_PATH $ENV{ROCM_PATH} CACHE PATH "ROCm path")
  endif()
endif()
set(CMAKE_MODULE_PATH "${ROCM_PATH}/lib/cmake" ${CMAKE_MODULE_PATH})

# Set GPU Targets and Find all the HIP modules
set(GPU_TARGETS “gfx906;gfx908” CACHE STRING "The GPU TARGETs" )
find_package(HIP REQUIRED)
find_package(hipfft REQUIRED)
find_package(hiprand REQUIRED)
find_package(rocrand REQUIRED)
find_package(hipblas REQUIRED)
find_package(rocblas REQUIRED)
find_package(hipcub REQUIRED)
find_package(rocprim REQUIRED)

set( MY_HIP_SRCS my_hip_src1.cpp my_hip_src2.cpp my_hip_src3.cpp)

# Mark source files as HIP.  I guess in the future just a
# LANGUAGE HIP property will suffice. For now do it via compile flags
set_source_files_properties( ${MY_HIP_SRCS} PROPERTIES LANGUAGE CXX)
set_source_files_properties( ${MY_HIP_SRCS} PROPERTIES
                             COMPILE_FLAGS "-x hip”)

# Create a Library dependent on HIP
add_library( myLib ${MY_HIP_SRCS} )
target_include_directories(myLib PUBLIC ${ROCM_PATH}/hipfft/include)
target_link_libraries(myLib PUBLIC
    hip::hiprand roc::rocrand
    hip::hipfft
    roc::hipblas roc::rocblas
    hip::hipcub roc::rocprim_hip )
```

Gotcha:
- hipfft/rocfft combination is in transition
- just adding hip::hipfft as dependency ended up adding the wrong include directory for me, So I needed to add an explicit include directory here. This will be fixed.

ROCProf – The AMD Profiler and Tracer

- rocprof measures a variety of counters and can trace execution
- There are ‘basic counters’ and ‘derived counters’
  - rocprof --list-basic
  - rocprof --list-derived
- Useful to know your code limiters to guide what to measure
  - e.g. Lattice QCD Wilson Dslash (my all time fave kernel / Nemesis)
  - Memory Bandwidth bound (Flops/Byte $\in$ [~0.87 – ~2.7])
  - High register usage: minimally around 70 registers needed/kernel
    - Spilling is a possibility

Measuring Memory Bandwidth

- Derived Counters: FetchSize, WriteSize
- In single device interactive job invoke as:

```
pmc: FetchSize WriteSize
pmc: L2CacheHit
```

mem_counters.txt file

Input counters

Track Time

Output file

```
rocprof -i ./mem_counters.txt --timestamp on -o ./dslash_test.csv \
                                                                  \
          ./dslash_test --dim 16 16 16 16 --niter 10
```

Application command line

- 2 lines in input -> executable will run twice
- Profiling may affect performance

View CSV e.g. in Excel.

Shared Mem (LDS)=5632 VGPRs=64 Fetch=22.8 MiB Call Time=~370-390 µs

AutoSave OFF — dslash_test — Saved to my Mac

Home Insert Draw Page Layout Formulas Data Review View Acrobat Tell me — Share — Comments

Paste Cut Copy Format — Calibri (Body) 12 — Wrap Text — Merge & Center — General — Conditional Formatting — Format as Table — Normal Bad Good Neutral Calculation Check Cell — Insert Delete Format — AutoSum Fill Clear — Sort & Filter — Find & Select — Sensitivity — Create and Share Adobe PDF

Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format. — Save As...

S34

	A	B	H	I	J	K	L	M	Q	R	S	T	U	V	W	X	Y	Z
1	Index	KernelName	grd	wgr	lds	scr	vgpr	sgpr	FetchSize	WriteSize	L2CacheHit	DispatchNs	BeginNs	EndNs	CompleteNs	End-Begin (Ns)	Disp-Complete (Ns)	
2	0	_ZN4quda8Kernel1DINS_7reducer10init_countENS1_8init_ar	1152	384	8192	0	4	24	0	4	16	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	6400	14371282	
3	1	_ZN4quda8Kernel3DINS_10CopyGauge_ENS_12CopyGaugeA	262144	64	1536	528	32	48	57760	151552	16	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	235200	1082787	
8	6	_ZN4quda8Kernel3DINS_14GhostExtractorENS_15ExtractGho	33280	320	0	496	48	48	4918	17012	3	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	104160	433588	
18	16	_ZN4quda8Kernel2DINS_16CopyColorSpinor_ENS_18CopyCo	33152	448	0	0	64	24	6629	3072	44	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	65439	414302	
19	17	_ZN4quda11Reduction2DINS_4blas7Reduce_ENS_15ReduceK	69632	512	512	0	16	32	3090	1	3	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	41280	468292	
20	18	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23384	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	174080	541108	
21	19	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23387	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	52320	355923	
22	20	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23407	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	53760	356024	
23	21	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23402	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	56320	389947	
24	22	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23426	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	53280	351755	
25	23	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23400	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	53440	378756	
26	24	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23388	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	52640	367034	
27	25	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23412	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	53279	371512	
28	26	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23411	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	54240	356474	
29	27	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23433	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	53760	395748	
30	28	_ZN4quda8Kernel3DINS_14dslash_functorENS_18dslash_fun	32832	192	5632	276	64	112	23402	3072	39	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	56000	369308	
31	29	_ZN4quda8Kernel2DINS_16CopyColorSpinor_ENS_18CopyCo	32960	320	0	0	64	24	3106	6370	66	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	87679	518035	
32	30	_ZN4quda11Reduction2DINS_4blas7Reduce_ENS_15ReduceK	69632	512	512	0	16	32	3091	1	3	3.52547E+14	3.52547E+14	3.52547E+14	3.52547E+14	42880	432145	
33																		
34																		
35																		
36																		

dslash_test +

Ready — 126%

Scratch=276 SGPR=112 Write=3 MiB Kernel Time=53-56 µs

Comments

- Name Mangling: `llvm-cxxfilt` (supplied with ROCM) is your friend

```
[bjoo@login1.spock test]$ llvm-cxxfilt _ZN6Kokkos12Experimental4ImplL32hip_parallel_launch_local_memoryINS_
4Impl11ParallelForINS3_16ViewValueFunctorINS0_3HIPEjLb1EEENS_11RangePolicyIJS6_NS_9IndexTypeIlEEEEES6_
EELj1024ELj1EEEvPKT_

void Kokkos::Experimental::Impl::hip_parallel_launch_local_memory<Kokkos::Impl::ParallelFor<Kokkos::Impl::
ViewValueFunctor<Kokkos::Experimental::HIP, unsigned int, true>, Kokkos::RangePolicy<Kokkos::Experimental::HIP,
 Kokkos::IndexType<long> >, Kokkos::Experimental::HIP>, 1024u, 1u>(Kokkos::Impl::ParallelFor<Kokkos::Impl::
ViewValueFunctor<Kokkos::Experimental::HIP, unsigned int, true>, Kokkos::RangePolicy<Kokkos::Experimental::HIP,
 Kokkos::IndexType<long> >, Kokkos::Experimental::HIP> const*)
```

- CompleteNs – DispatchNs ~ call time
- EndNs – BeginNs – kernel run time << call time here -> latency !!
- Actual BW ~ 26 MiB/55 us ~ 461 GiB/s (End – Begin)
- Observed BW ~ 26MiB/380 us ~ 66.8 GiB/s (CompNs-Di
- Some ‘Scratch’ is used. Are we spilling registers?

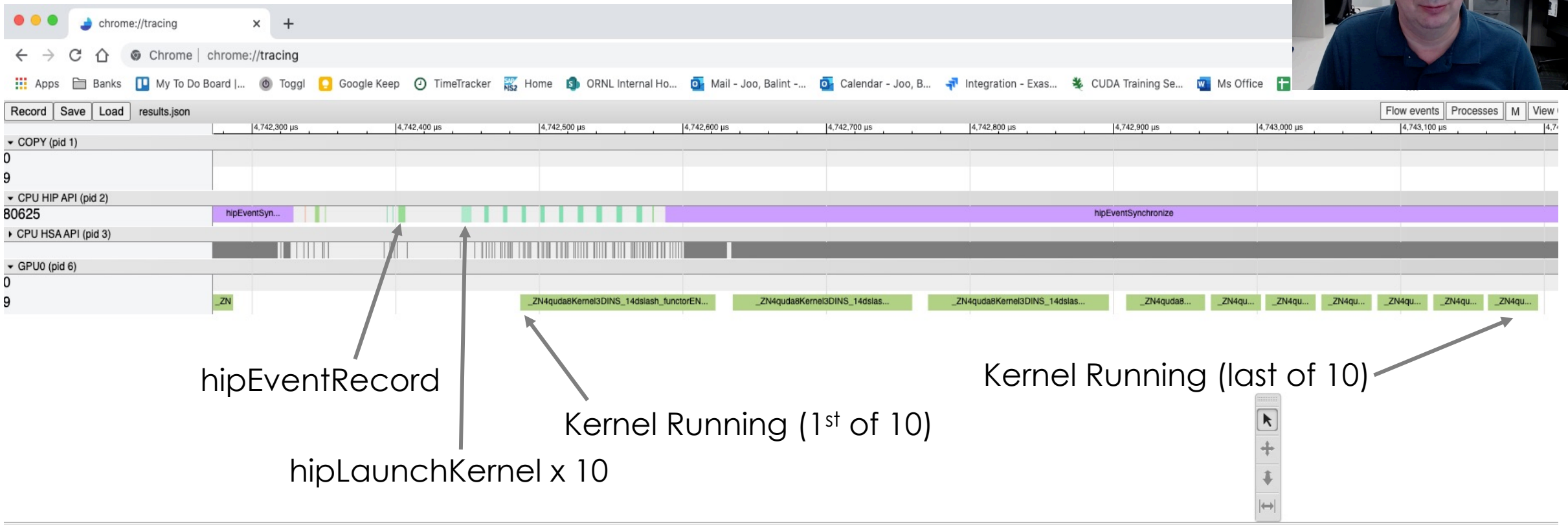
Rocprof and Tracing

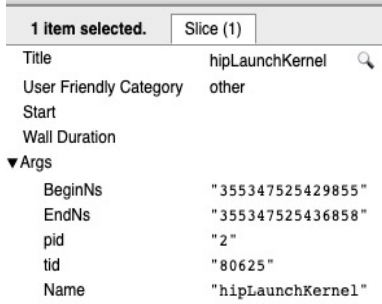
- To Trace HIP, HSA and GPU execution use

```
rocprof --sys-trace \
  ./dslash_test --dim 16 16 16 16 --niter 10
```

- Generates JSON file to use with ‘Chrome’ Trace viewer
- Default name: results.json
- You can view with a trace viewer.
  - Type ’chrome://tracing’ in your chrome URL location
  - Or use your favorite Chrome-Trace compatible tracer tool
  - Getting used to navigating the traces in Chrome can take some time.

Chrome Trace





Last Kernel: DurationNs => 35840 ns
⇒ Bandwidth ~ 709 GiB/s
⇒ different profiling methods have different overheads...

Marking code sections with Rocprof

- To help highlight/filter various ranges one can instrument code with the 'roctx' API
  - add to hipcc include path: `-I${ROCM_PATH}/roctracer/include`
  - link with: `-L${ROCM_PATH}/roctracer/lib -lroctracer64 -lroctx64`
  - `#include <roctracer_ext.h>`
    - `void roctracer_start()` - starts the tracer API
    - `void roctracer_stop()` - stop the tracer API
    - Can start roctracer with tracing stopped using: `–trace-start off`
  - `#include <roctx.h>`
    - `void roctxRangePush(const char *message)` – start a nested range
    - `void roctxRangePop()` - jump up from a nested range
- For more info about rocprof please see:
  - https://rocmdocs.amd.com/en/latest/ROCm_Tools/ROCm-

Generating ISA files

- Compile with
  `-g –ggdb –save-temps`
- This will save LLVM bytecode, GPU assembly and object files:
  - `test_kokkos_perf`
  - `test_kokkos_perf-hip-amdgcn-amd-amdhsa-gfx908.s` <- assembly
  - `test_kokkos_perf-hip-amdgcn-amd-amdhsa-gfx908.o` <- object
- Assembly can be immediately looked at
- Dump object files with llvm-objdump e.g.:
  - `llvm-objdump --source --line-numbers ./test_kokkos_perf-hip-amdgcn-amd-amdhsa-gfx908.o > ISA.dump`

Useful Info in Assembly files

- In the .s files look for function begin and end points:
  - .Lfunc_beginXXX – identify kernel
  - .Lfunc_end – useful into

```
        .text
                .globl
_ZN6Kokkos12Experimental4ImplL32hip_parallel_launch_local_memoryINS_4Impl11ParallelForIN2MG
13DslashFunctorINS_7complexIfEES8_S8_Li1ELi0EEENS_11RangePolicyIJNS0_3HIPENS_12LaunchBounds
ILj256ELj1EEEEEESB_EELj256ELj1EEEvPKT_ ; -- Begin function
_ZN6Kokkos12Experimental4ImplL32hip_parallel_launch_local_memoryINS_4Impl11ParallelForIN2MG
13DslashFunctorINS_7complexIfEES8_S8_Li1ELi0EEENS_11RangePolicyIJNS0_3HIPENS_12LaunchBounds
ILj256ELj1EEEEEESB_EELj256ELj1EEEvPKT_
                .p2align        8
                .type
_ZN6Kokkos12Experimental4ImplL32hip_parallel_launch_local_memoryINS_4Impl11ParallelForIN2MG
13DslashFunctorINS_7complexIfEES8_S8_Li1ELi0EEENS_11RangePolicyIJNS0_3HIPENS_12LaunchBounds
ILj256ELj1EEEEEESB_EELj256ELj1EEEvPKT_,@function
_ZN6Kokkos12Experimental4ImplL32hip_parallel_launch_local_memoryINS_4Impl11ParallelForIN2MG
13DslashFunctorINS_7complexIfEES8_S8_Li1ELi0EEENS_11RangePolicyIJNS0_3HIPENS_12LaunchBounds
ILj256ELj1EEEEEESB_EELj256ELj1EEEvPKT_: ;
@_ZN6Kokkos12Experimental4ImplL32hip_parallel_launch_local_memoryINS_4Impl11ParallelForIN2M
G13DslashFunctorINS_7complexIfEES8_S8_Li1ELi0EEENS_11RangePolicyIJNS0_3HIPENS_12LaunchBound
sILj256ELj1EEEEEESB_EELj256ELj1EEEvPKT_
        .Lfunc_begin12:
```

Mangled name: use llvm-cxxfilt to unmangle

entry point

Useful Info in Assembly files

- In the .s files look for function begin and end points:
  - .Lfunc_beginXXX – identify kernel
  - .Lfunc_end – useful into

```
.Lfunc_end12:
; -- End function
          ... — I REMOVED STUFF To save space....
        .section         .AMDGPU.csdata
; Kernel info:
; codeLenInByte = 10640
; NumSgprs: 13
; NumVgprs: 108
; NumAgprs: 0
; TotalNumVgprs: 108
; ScratchSize: 0
; MemoryBound: 0
; ...
```

Useful info about GPR's
NumAgprs + Scratch Space = 0 means no spills.

- .s files also give hints about spills. Search for "Folded Spill"

Summary

- We discussed
  - modules needed to get developing with hip on Spock
  - running single device, interactive jobs, for debugging & profiling
  - how to set up CMake for building for HIP/ROCm
  - how to generate profiles and traces using the QUDA ‘dslash_test’ as an example (memory b/w bound kernel run in a latency bound region)
  - how to generate ISA, and look for kernel information
- For more information, visit the spock training and documentation pages
  - https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html
  - https://www.olcf.ornl.gov/spock-training
- Live Long and Prosper !!!!

Acknowledgements and Thanks!

- These tidbits here are a disordered collection of information I have gathered from our Frontier Center of Excellence colleagues at AMD especially: Nick Curtis, Damon McDougall and Corbin Robeck
- Our profiling examples used the QUDA Code available from https://github.com/lattice/quda.git which is maintained by Kate Clark and the QUDA community.
- Our ISA example use Kokkos Dslash which uses Kokkos. Big shout out to the Kokkos Team! Locally at ORNL the HIP porting is the hard work of Damien Lebrun-Grandie, Bruno Trucsin, Daniel Arndt and colleagues working closely with Nick Curtis (https://github.com/kokkos)