HPCToolkit: New Support for GPU Measurement

John Mellor-Crummey and Yuning Xia
Rice University

Scalable Tools Workshop
August 12, 2024

NNSA National Nuclear Security Administration | U.S. DEPARTMENT OF ENERGY | Office of Science | STEP SOFTWARE TOOLS ECOSYSTEM PROJECT | RICE

HPCToolkit and Graphics Processing Units

Goal: Support performance analysis including PC sampling on AMD, Intel, and NVIDIA GPUs

Vendor	Coarse-grain measurement	Fine-grain measurement	Tracing	Binary analysis: loops, inlined code
NVIDIA	CUPTI	PC sampling	CUPTI	nvdisasm + Dyninst
AMD	Rocprofiler	PC sampling with Rocprofiler-sdk	Rocprofiler	Dyninst
Intel	OpenCL and Level 0	GTPin instrumentation + PC sampling	Level Zero callbacks	IGA + Dyninst

Outline

- **Experiences with HPCToolkit and Exawind on Frontier**
- **HPCToolkit and Rocprofiler-sdk**
- **HPCToolkit's new support for PC sampling on Intel GPUs**

ExaWind: A Wind Farm Simulator

- **Simulate a wind farm with tens of megawatt-scale wind turbines**
- **Code leverages all CPU cores and GPU tiles on a node**
  - AMR-Wind handles the structured grid flow between turbines with one core per GPU
  - Nalu-Wind handles the unstructured CFD over the turbine geometry, running on the node's remaining 56 CPU cores

ExaWind: Wakes from Three Wind Turbines Over Time

Figure credit: Jon Rood, NREL

ExaWind: Visualization of a Wind Farm Simulation

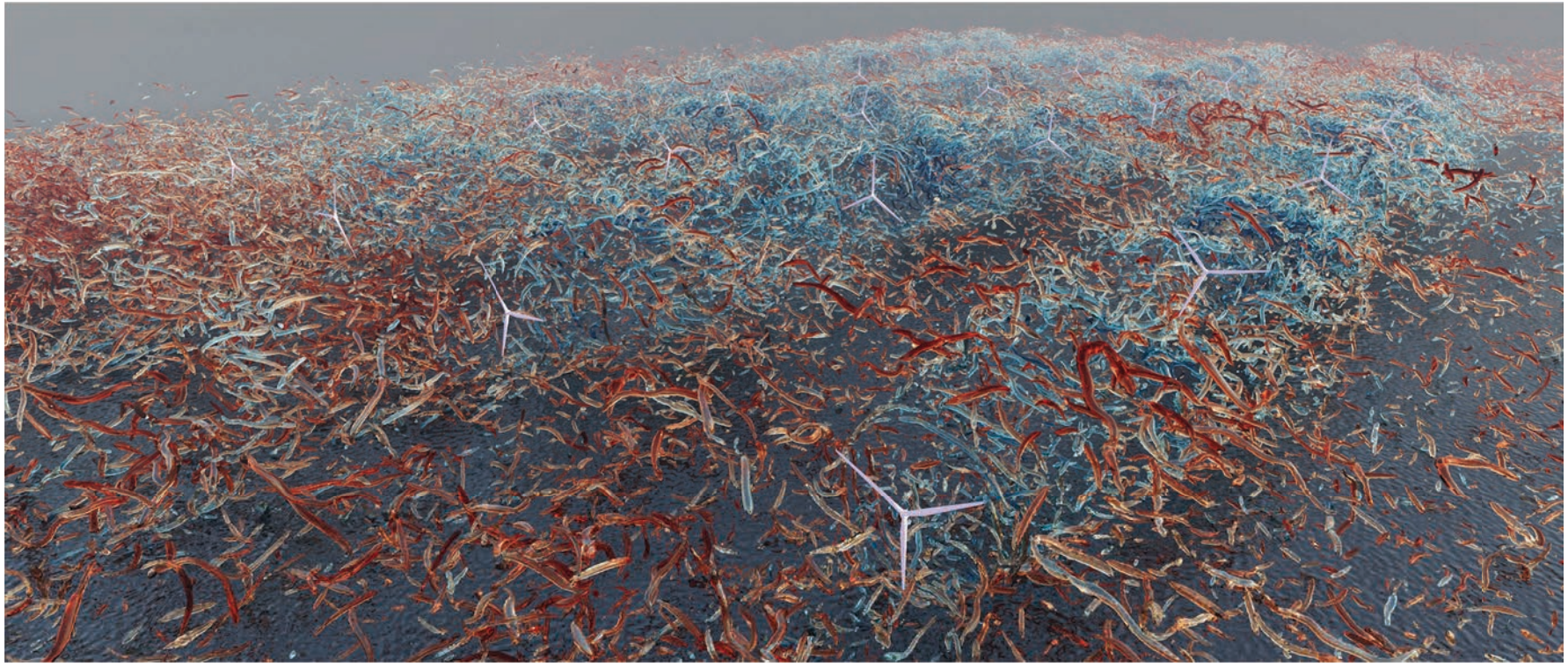
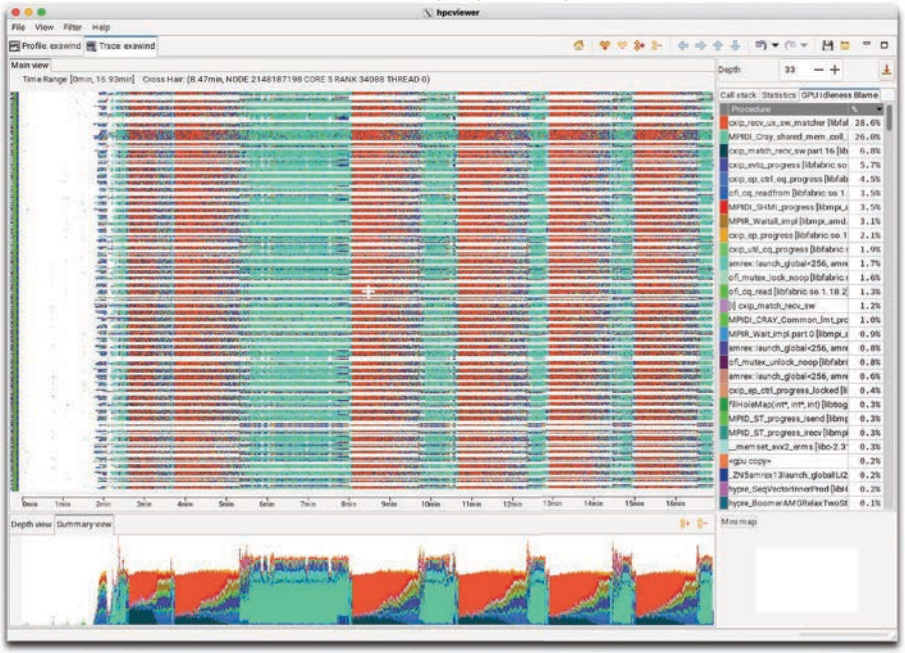


Figure credit: Jon Rood, NREL

ExaWind: Execution Traces on Frontier Collected with HPCToolkit

Traces on roughly ~70K MPI ranks for ~17 minutes

Before: MPI waiting (bad), shown in red



After: MPI overhead negligible*

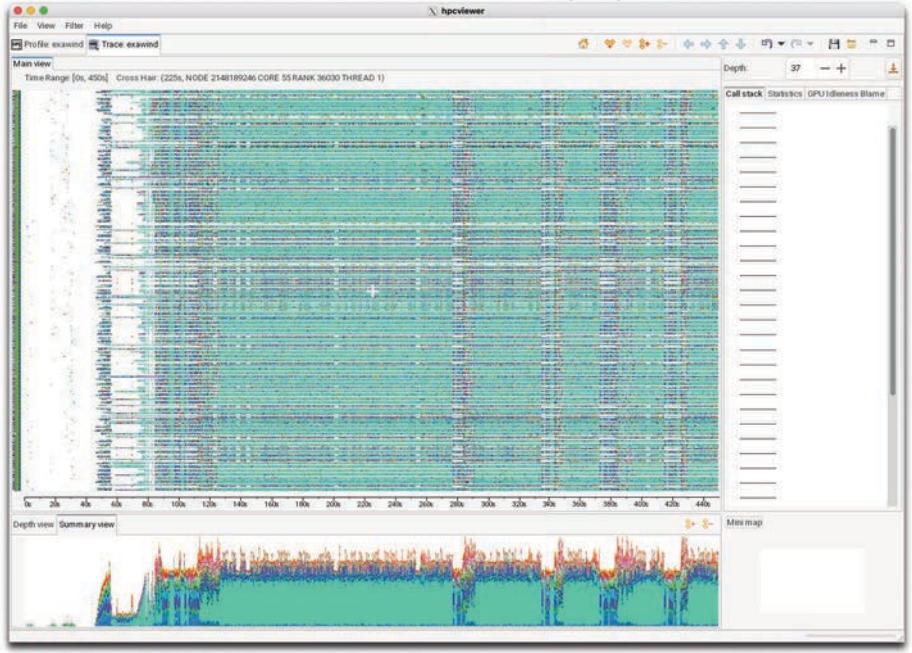


Figure credits: Jon Rood, NREL

*replaced non-blocking send/recv with ialltoallv

ExaWind Testimonials for HPCToolkit

… we can't overstate how complicated ExaWind is in general, and how complicated it is to build, so finding out that HPCToolkit could easily profile our entire application without a ton of instrumentation during the build process, and be able to profile it on a huge amount of Frontier with line numbers and visualizing the trace was really amazing to us.

- Jon Rood NREL (6/3/2024)

… during a recent hackathon we had, we improved our large scale performance by 24x by understanding our code better with HPCToolkit and running it on 1000s of nodes while profiling. We also recently improved upon this by 10% for our total runtime.

- Jon Rood NREL (5/31/2024)

Outline

- **Experiences with HPCToolkit and Exawind on Frontier**
- **HPCToolkit and Rocprofiler-sdk**
- **HPCToolkit's new support for PC sampling on Intel GPUs**

Motivation: Shortcomings of AMD's Rocprofiler and Roctracer

- **Like other GPU runtimes, ROCm runtime and tool libraries created threads without notice**
  - Monitoring may be ill-advised or unsafe
- **No well-defined point for tool initialization**
- **Environment variables were used to pass key information**
- **Rocprofiler and roctracer were separate libraries; composing them is non-trivial**
- **Lack of attention to compatibility across different ROCm versions**
- **Hard for tool projects to interoperate (e.g. HPCToolkit and PAPI)**
- **Unable to correlate kernel launches with machine code on systems with heterogeneous GPUs**
- **No support for PC sampling**

Design Recommendations (Mellor-Crummey CORAL2 Meeting 6/23)

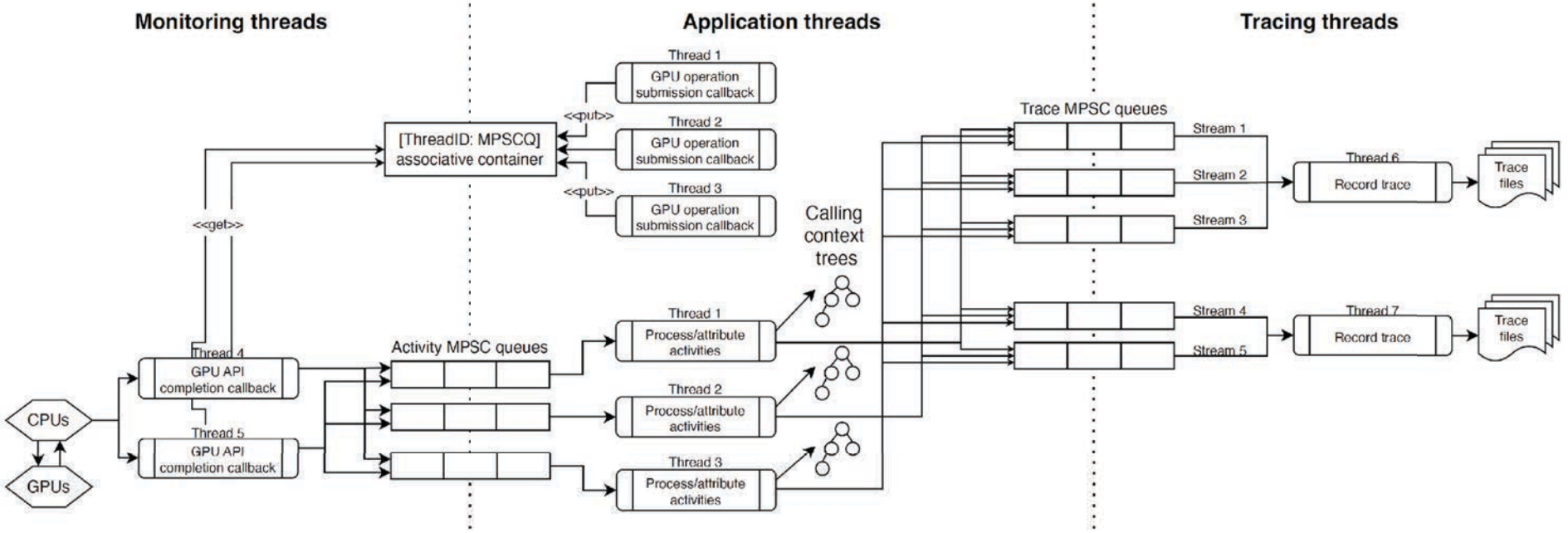
- **Everything should go through an API**
  - avoid use of environment variables to specify libraries, metrics files, etc.
- **One version of the software should support all recent ROCm versions**
  - compile a tool with the newest header files
  - use tool with shared library matched to the runtime version compiled into an application
- **Tool library should**
  - support measurement of kernels using HW counters and activity tracing in the same execution
  - support PC sampling for non-root users
  - provide a simple mechanism to correlate measurement data with GPU binaries and kernels
  - support measurement of both HIP and HSA-based OpenMP
- **Need a mechanism to explain thread creation**
- **Need a mechanism to guarantee tool initialization**
- **Tool library should report all significant GPU events, event those not user initiated**
  - e.g. scratch space reclamation on GPUs

Rocprofiler-sdk (ROCm 6.2, released 8/7/2024) addresses most of these issues

Interfacing HPCToolkit and Rocprofiler-sdk

- **Rocprofiler-sdk measures employs various threads to measure performance**
  - a sampling thread per GPU tile to receive PC samples
  - an API thread to receive operation trace records (kernel execution, memory copies, …)
  - a counter thread to receive counter measurements of GPU kernels
- **In HPCToolkit, each application thread records its own GPU measurements in profiles and traces**
- **HPCToolkit ships performance measurements between tool threads and application threads using Multi-Producer, Single-Consumer (MPSC) queues**
  - avoids the need for all data structures a thread uses to record its performance to be concurrent
    - the only thread that records data in a thread's profile is the thread itself
  - wait-free data structure for performance measurement (!)

Interthread Communication in HPCToolkit with Rocprofiler-sdk



HPCToolkit + Rocprofiler-sdk: Tool Initialization (Mostly Complete)

- **Implemented rocprofiler_configure, which coordinates initialization with rocprofiler-sdk**
  - registers to receive tool initialization and finalization callbacks
  - registers to receive callbacks before/after thread creation
    - Skip monitoring of runtime and tool threads
- **When rocprofiler-sdk triggers the tool initializer**
  - guarantee a rendezvous of HPCToolkit's initialization and Rocprofiler-sdk initialization
  - initialize hpcrun, which enables GPU monitoring specified by command line arguments to hpcrun
- **Thread pre-create/post-create callbacks enable tool to ignore runtime/tool threads**
- **Remaining issue: need to selectively configure monitoring rather than monitoring everything**

HPCToolkit + Rocprofiler-sdk: API Tracing (Mostly Complete)

- **Initialization**
  - register for callback when a GPU operation is initiated
  - register to have API thread receive buffer of records detailing execution of GPU operations (kernels, copies, …)
- **Measurement**
  - when a GPU operation is launched, application thread
    - unwinds the call stack to determine the calling context
    - assigns the operation an external correlation id: [thread id, sequence number]
    - records mapping between correlation id and calling context
  - when the API thread receives a buffer of trace records
    - decodes each record in the buffer
    - sends information about a GPU operation back to the application thread identified by the correlation id
  - application thread
    - receives measurement data & looks up the associated calling context using the correlation id
    - records appropriate GPU operation metrics in its profile (time, bytes copied, …)
    - sends notice of event to a tracing thread
  - tracing thread
    - enters the operation in a trace line for the GPU queue
- **Remaining issue: need to integrate support for operations beyond kernel launch and memory copies**

HPCToolkit + Rocprofiler-sdk: PC Sampling (Full Implementation)

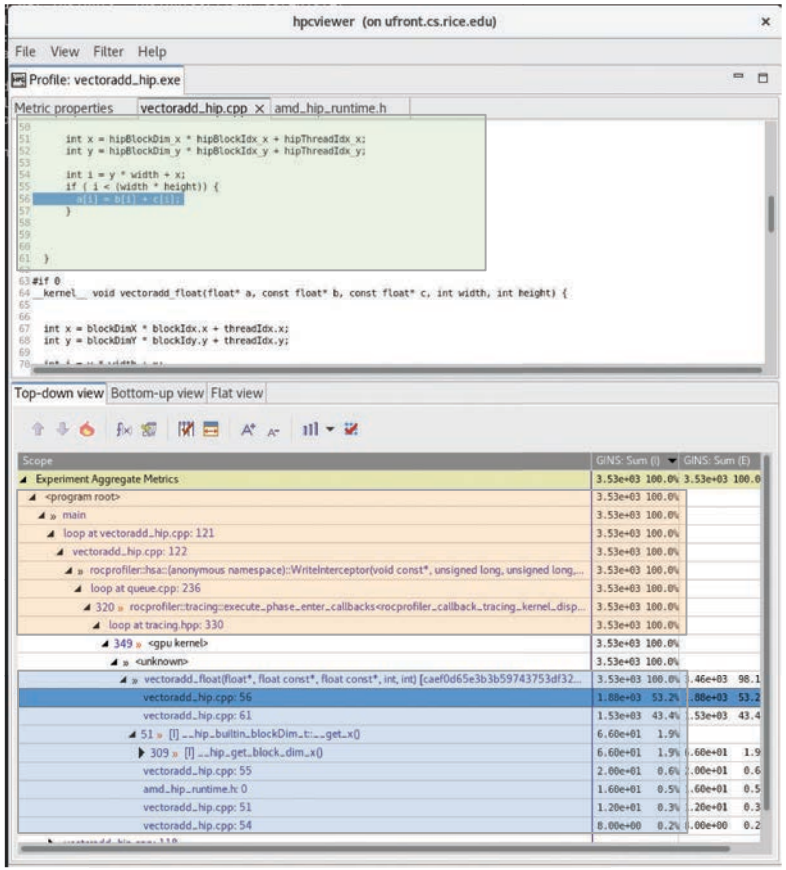
- **Initialization**
  - iterate over all agents
  - configure MI210 GPU to record PC samples as kernels execute and report PC samples to a *sampling thread* for that GPU
- **Measurement**
  - *sampling thread*
    - decodes each PC sample in a trace buffer
    - constructs a normalized PC by mapping to a code object using load map
    - sends the sample back to the application thread identified by the correlation id for the sample
  - *application thread*
    - uses correlation id from the sampling thread to identify the calling context for the kernel launch
    - records *normalized PC* as a child of the calling context where it was invoked

HPCToolkit + Rocprofiler-sdk: PC Sampling (Full Implementation)

GPU kernel source

Host (CPU) call stack

Device (GPU) call stack



HPCToolkit + Rocprofiler-sdk: GPU Counters (Substantially Complete)

- **Initialization**
  - iterate over all agents to assess the available counters
  - assemble a counter ‘profile’ for each GPU agent to record requested counter values
- **Measurement**
  - counter thread receives counter values in a trace buffer and sends them back to the application thread identified by the associated kernel launch correlation id
  - application thread will use the correlation id from the counter thread to identify the callpath where the kernel was launched
  - application thread will record the counters in the profile
- **Remaining issues**
  - ship counter values to application thread
  - use correlation id to locate the calling context
  - augment profile for calling context

HPCToolkit + Rocprofiler-sdk: Managing GPU Code Objects

- **Initialization**
  - register for callbacks when code objects loaded/unloaded
  - sampling thread receives unloading notifications
- **Measurement**
  - API thread receives code object notifications and records code objects during execution
- **Post processing**
  - hpcstruct analyzes AMD GPU binaries
- **Remaining issues**
  - record call site information to enable reconstruction of static call graphs
  - investigate why attribution of PC samples by hpcprof without hpcstruct files omits some source lines

Findings using Rocprofiler-sdk Pre-release

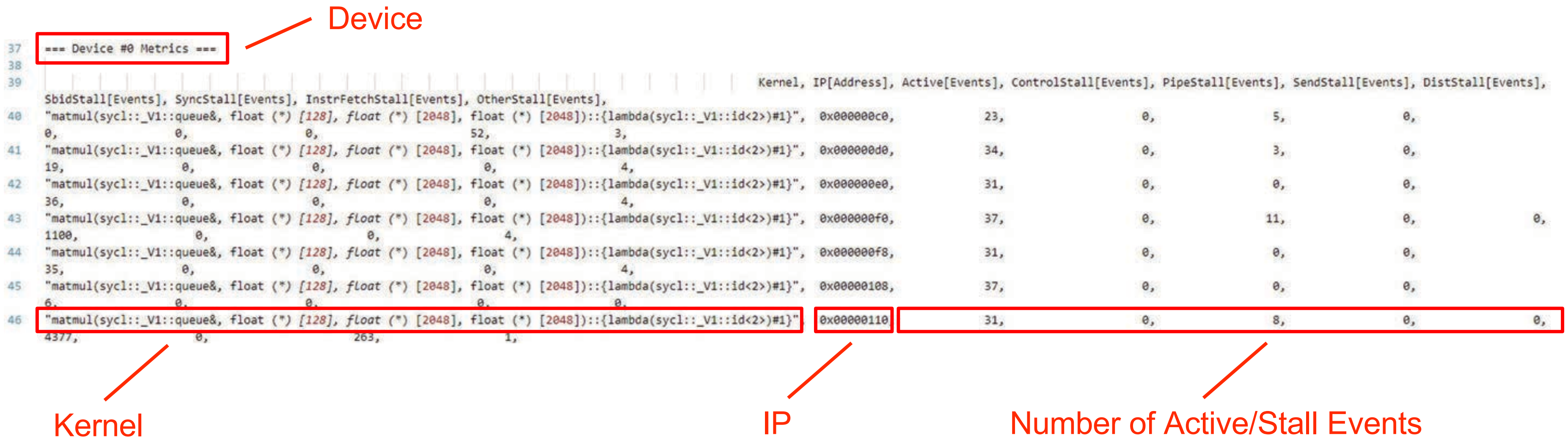
- **Agent iterator exposes ALL AMD GPUs, not just those visible using ROCR_VISIBLE_DEVICES**
- **API ENTER/EXIT callbacks don't always occur on the application thread**
  - when using counters, we saw that often these callbacks occur on an HSA thread
- **Corresponding API ENTER and EXIT callbacks don't always occur on the same thread**
- **Rocprofiler-sdk destructors may be called too early**
  - destructors for C++ static objects are triggered on exit
  - HSA asynchronous was still at work and later had a SEGV trying to use rocprofiler-sdk data structures
- **External correlation id callback added as an alternative to the callback ENTER/EXIT**
  - would prefer a pair of callbacks BEFORE/AFTER enqueue
- **Awkward to configure monitoring**
  - monitor KERNEL_DISPATCH, MEMORY_COPY
  - a collection of calls to HIP functions that match regular expressions

Outline

- **Experiences with HPCToolkit and Exawind on Frontier**
- **HPCToolkit and Rocprofiler-sdk**
- **HPCToolkit's new support for PC sampling on Intel GPUs**

PC Sampling on Intel GPUs: Flat Samples without Context

- **Instruction samples are collected separately on each active tile/sub device and merged in a buffer returned by Intel's LevelZero API**
- **Intel Unitrace only attributes PC samples to kernel names, not individual kernel launches**



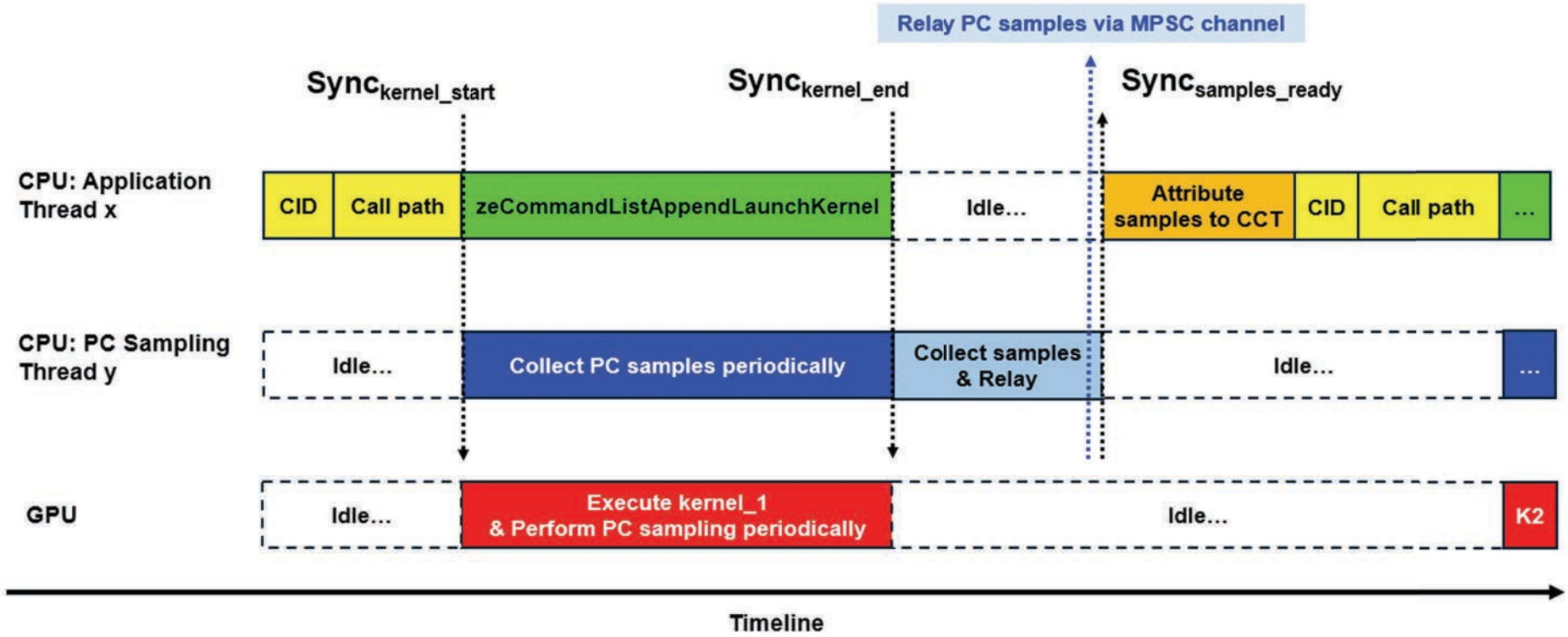
HPCToolkit's Support for Intel GPU PC Sampling

Goal: collect and attribute PC samples for multiple concurrent application threads

Approach: serialize access by application threads to each GPU tile

- **Two kinds of threads**
  - Application Threads (N per process)
  - Monitoring Thread (1 per GPU tile)
- **Operation**
  - Application thread acquires a lock for a GPU tile before launching a GPU kernel
  - Application thread signals the monitoring thread for the GPU tile
  - Monitoring thread collects samples for the kernel
  - Application thread notifies monitoring thread of kernel completion
  - Monitoring thread flushes samples from the GPU, passes them to appl thread, releases lock
  - Application thread records the samples

Sampling an Execution of a GPU Kernel on an Intel GPU

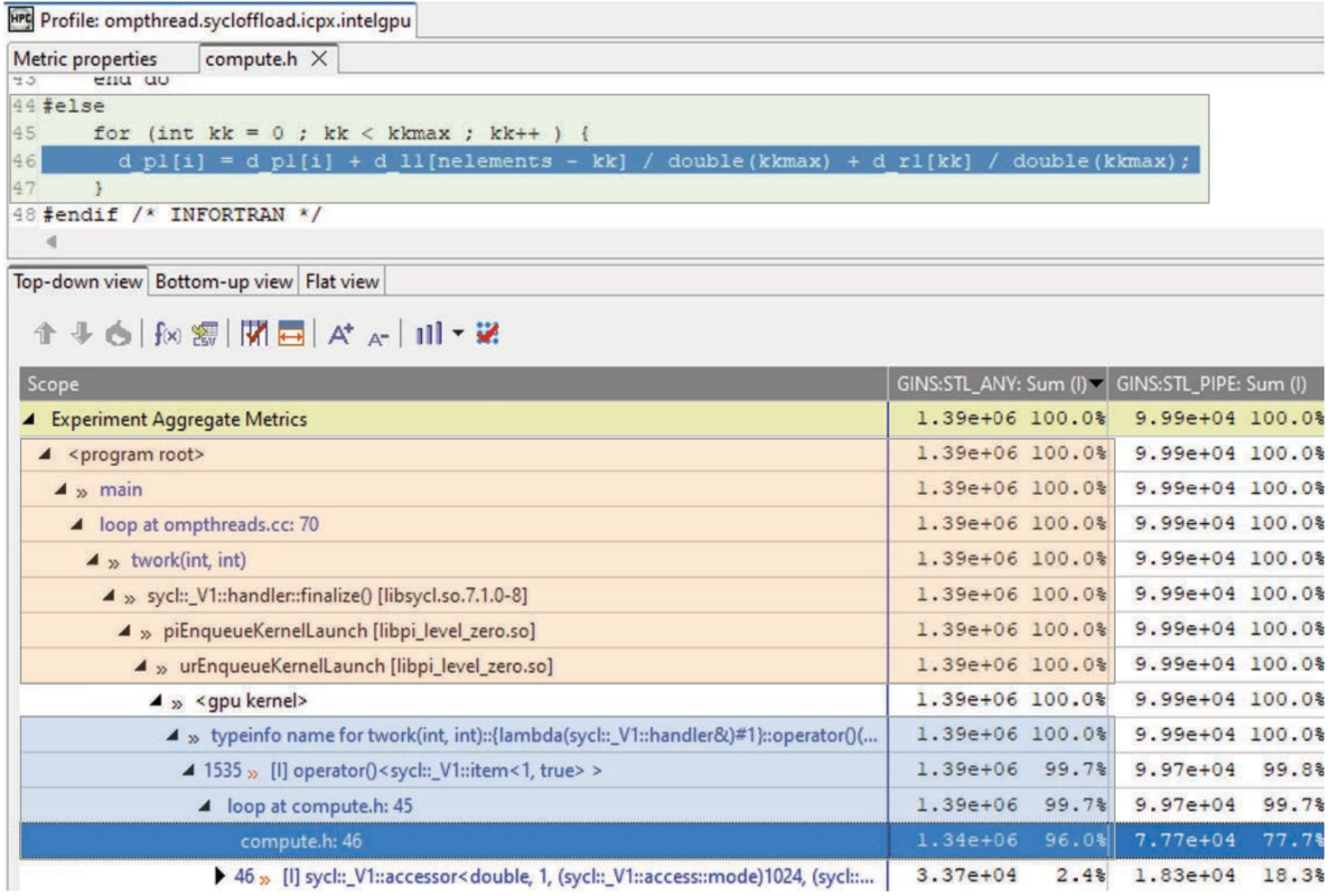


Example

GPU kernel source

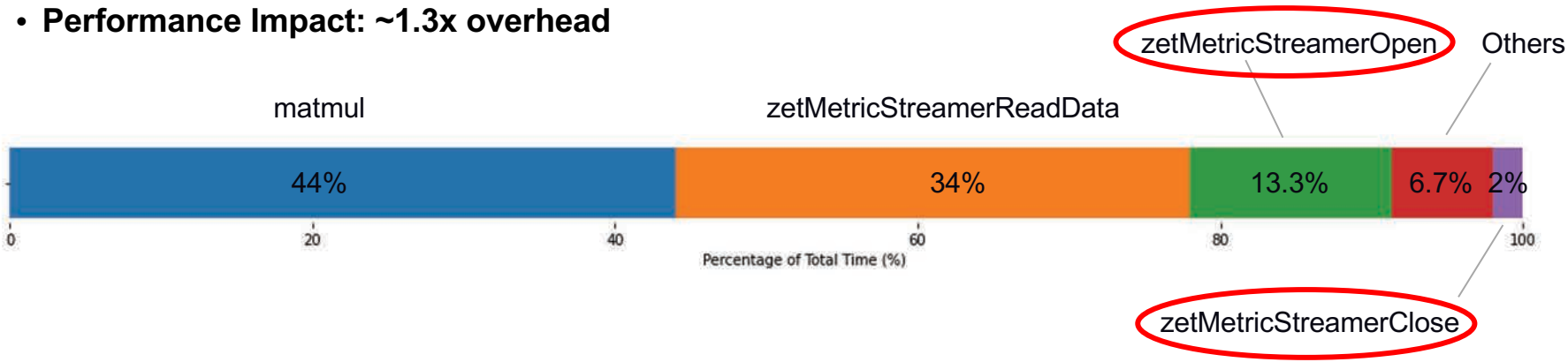
Host (CPU) call stack

Device (GPU) call stack



Analyzing Overhead of PC Sampling on Intel GPUs in HPCToolkit

- **Test Configuration**
  - AMD CPU + Intel Data Center GPU Max 1500
  - Intel(R) oneAPI DPC++/C++ Compiler 2024.1.0
- **Test Case**
  - Single thread launching 4800 matrix multiplication kernels on one GPU
  - Kernel dimensions: 12288x128x2048
- **Performance Impact: ~1.3x overhead**



Intel's Level Zero lacks a routine to flush PC Samples: repeatedly open+close instead

Next Steps

- **AMD GPUs**
  - Finish integration and testing of Rocprofiler-sdk in HPCToolkit
  - Evaluate measurement overhead for PC sampling and tune implementation
  - Add PC sampling support for MI300A once infrastructure available from AMD
- **Intel GPUs**
  - Implement PC sampling of concurrent kernels on Intel GPUs
    - support simultaneous execution of multiple kernels without serialization
  - lower monitoring overhead
    - use gprof-style statistical attribution to kernel launch contexts
    - trade off between accuracy and performance cost
- **Evaluate PC sampling capabilities on AMD, Intel, and NVIDIA GPUs**