Multi-Platform SYCL Profiling with TAU

Nicholas Chaimov

Sameer Shende

Allen Malony

ParaTools, Inc.

IWOCL 2020

ParaTools

Outline

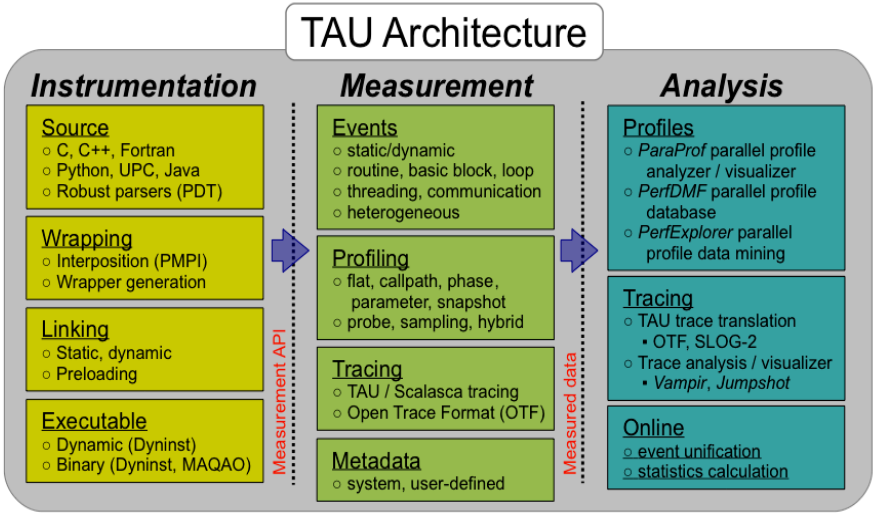
- Motivation: platform-agnostic performance counter profiling
- What is TAU?
- Early Implementation Work
  - NVIDIA: hipSYCL + CUPTI
  - AMD: hipSYCL + rocprofiler
  - Intel: OpenCL library wrapping
  - Intel: oneAPI Level Zero tool interface

Motivation

- Performance portability
  - We want code to be not just portable, but performance portable
  - Analyzing requires ability to make measurements across platforms.
  - Vendor-specific tools are not cross-platform.
  - TAU with SYCL
    - Provide a cross-platform performance tool for a cross-platform programming model

The TAU Performance System®

- Tuning and Analysis Utilities (**25+ year project**)
- Comprehensive performance profiling and tracing
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms



- Integrated performance toolkit
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)

- Integrates with runtimes and application frameworks

TAU Supports All HPC Platforms

C/C++ CUDA UPC Python

Fortran GPI

OpenACC Java MPI

pthreads

Intel MIC OpenMP

Intel GNU

LLVM PGI Cray Sun

MinGW

AIX

Linux Windows

Insert yours here

BlueGene Fujitsu ARM

Android MPC OpenSHMEM

TAU Supports All HPC Platforms

C/C++ CUDA UPC Python GPI

Fortran OpenACC Java MPI

pthreads Intel MIC OpenMP

Intel GNU Sun

LLVM PGI Cray

MinGW AIX

Linux Windows

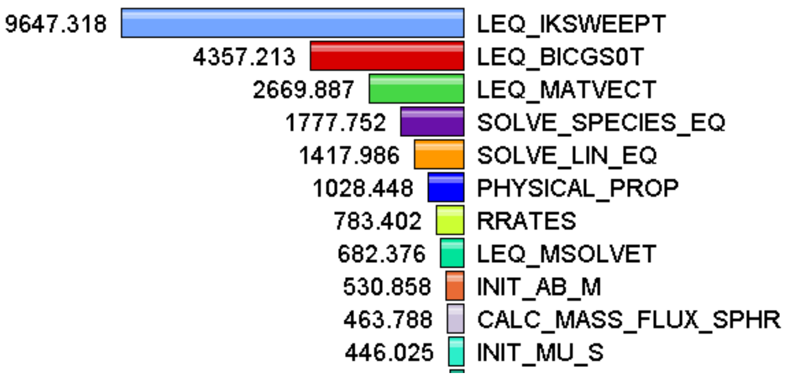
SYCL

BlueGene Fujitsu ARM

Android MPC OpenSHMEM

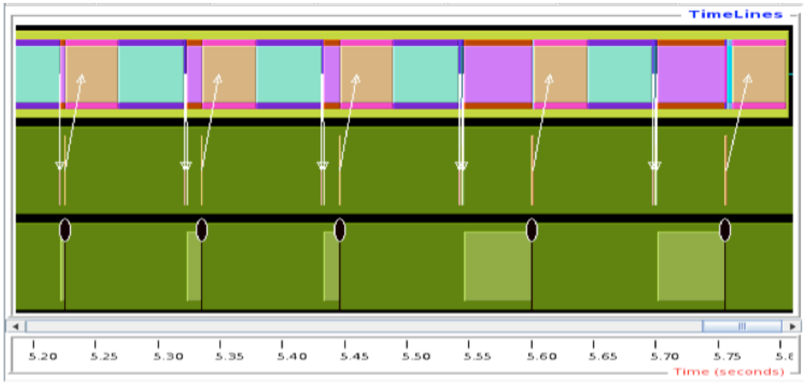
Measurement Approaches

Profiling



Shows **how much** time was spent in each routine

Tracing



Shows **when** events take place on a timeline

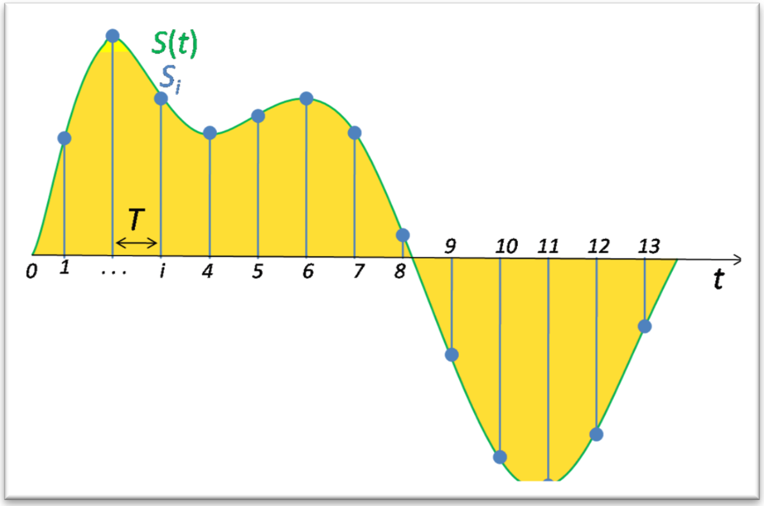
Performance Data Measurement

Direct via Probes

```
call TAU_START('name')
// code
call TAU_STOP('name')
```

- Exact measurement
- Fine-grain control
- Calls inserted into code or runtime

Indirect via Sampling



- No code modification
- Minimal effort
- Relies on debug symbols (**-g** option)

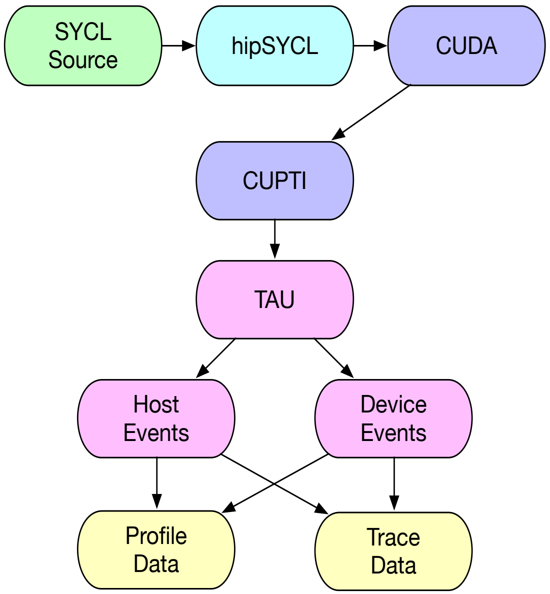
Questions TAU Can Answer

- **How much time** is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*?
- **How many instructions** are executed in these code regions? Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?
- **What is the memory usage** of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?
- **What are the I/O characteristics** of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- **What is the extent of data transfer** between host and a GPU? In applications using various programming models, such as CUDA, HIP, OpenCL, Kokkos, **SYCL**, etc.
- **What is the contribution of each *phase*** of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- **How does the application *scale***? What is the efficiency, runtime breakdown of performance across different core counts?

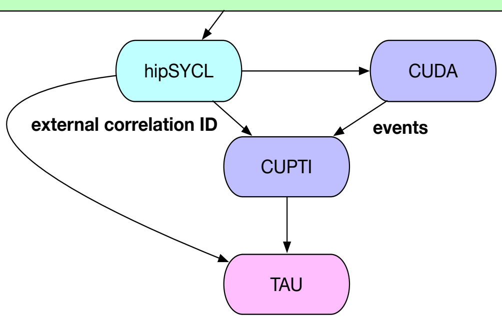
TAU's Support for Runtime Systems

- OpenCL
  - OpenCL profiling interface
  - Track timings of kernels
- OpenACC
  - OpenACC instrumentation API
  - Track data transfers between host and device (per-variable)
  - Track time spent in kernels
- CUDA
  - Cuda Profiling Tools Interface (CUPTI)
  - Track data transfers between host and GPU
  - Track access to uniform shared memory between host and GPU
- ROCm
  - Rocprofiler and Roctracer instrumentation interfaces
  - Track data transfers and kernel execution between host and GPU
- Python
  - Python interpreter instrumentation API
  - Tracks Python routine transitions as well as Python to C transitions

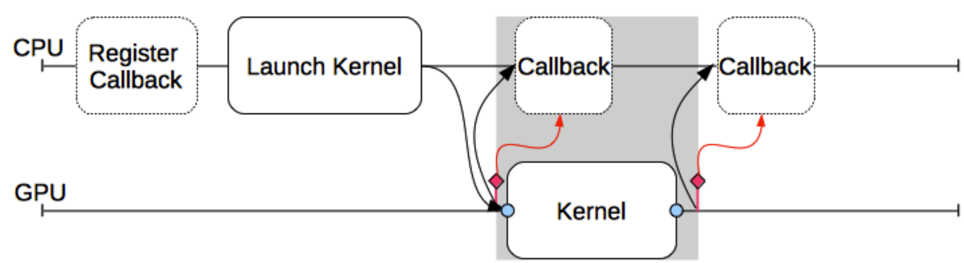
SYCL Profiling on NVIDIA GPUs




```
parallel_for(count, kernel_functor([=](id<> item) {
    int i = item.get_global(0);
    r[i] = a[i] + b[i] + c[i];
}));
```



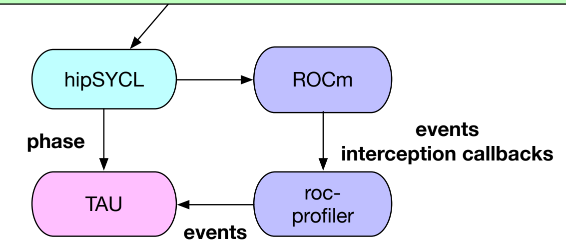
- Proof-of-concept implementation using hipSYCL.
- CUPTI
  - Synchronous callbacks for host-side API calls.
  - Asynchronous callbacks for device-side events.
    - Hardware performance counter access.
- Phase-based profiling to correlate CUDA kernels back to SYCL code.
  - CUPTI external correlation ID



SYCL Profiling on AMD GPUs

- As with NVIDIA, our proof-of-concept implementation uses hipSYCL.

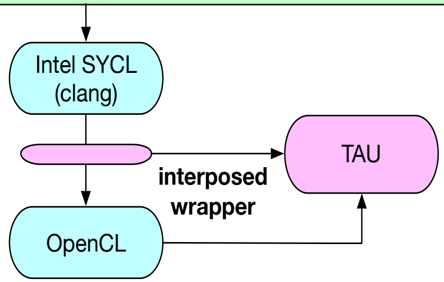
```
parallel_for(count, kernel_functor([=](id<> item) {
    int i = item.get_global(0);
    r[i] = a[i] + b[i] + c[i];
}));
```



- rocProfiler library for callbacks from AMD ROCm.
  – No equivalent to CUPTI’s external correlation IDs.
  – Interception API allows user-provided data to be attached to interception callback.
    - But interception API requires serializing kernel dispatches.

SYCL Profiling on Intel Embedded GPUs (1)

```
parallel_for(count, kernel_functor([=](id<> item) {
    int i = item.get_global(0);
    r[i] = a[i] + b[i] + c[i];
}));
```



- Initial implementation of Intel SYCL based on OpenCL backend.
- TAU provides wrapper libraries around OpenCL API functions which replace the runtime-provided versions.

- Wrapper for **clCreateCommandQueue** and **clCreateCommandQueueWithProperties** force profiling on.
- Each wrapper
  - Starts a timer
  - If relevant, records a context event indicating the size and source line of a transfer
  - Calls the underlying system version of the function
  - Stops the timer
- Loaded into unmodified application with LD_PRELOAD or through linker script at link time

SYCL Profiling on Intel Embedded GPUs (2)

- Event names from OpenCL profiling interface provide mangled name of originating functor from SYCL code.
- Intel Level Zero Tools Interface
  - No external correlation ID support
  - However, event name contains enough context information to avoid need
  - Tracer Markers allow user-provided data to be inserted into the event stream

Download TAU

http://tau.uoregon.edu

http://taucommander.com

https://e4s.io

Free download, open source, BSD license

Questions?
Contact **support@paratools.com**