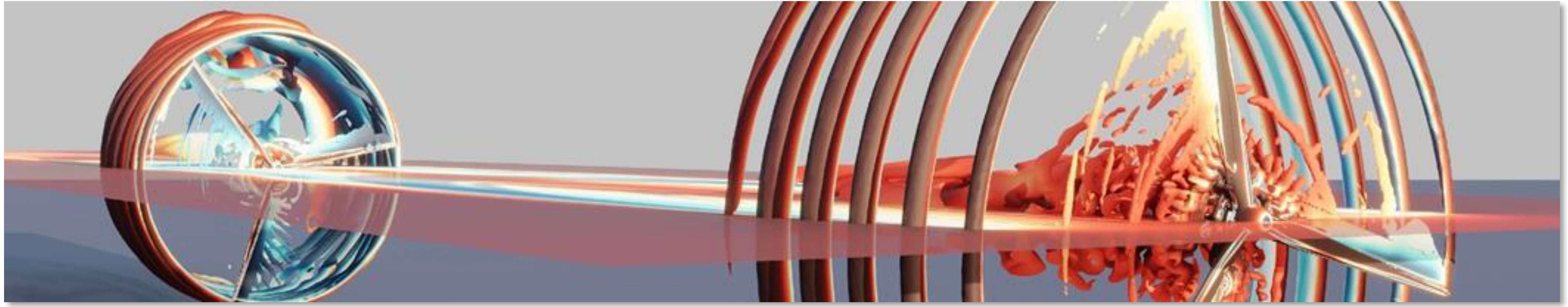
HPCToolkit

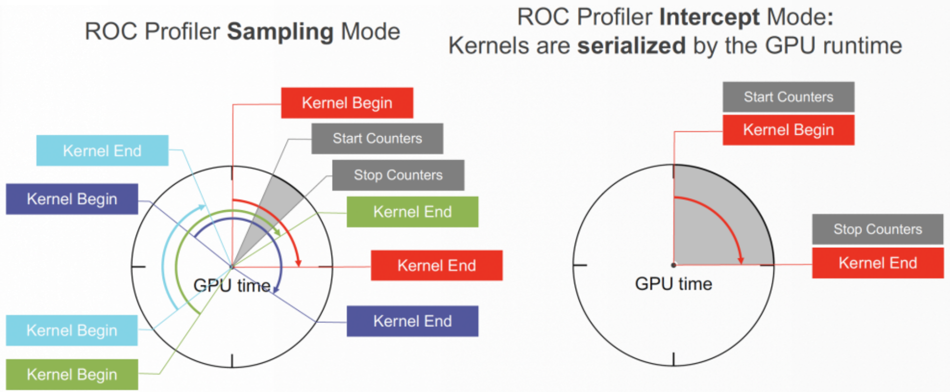
I just wanted to mention we've been using HPCToolkit a lot for our ExaWind application on Frontier, which is a hugely complicated code, and your profiler is one of the only ones we've found that really lets us easily instrument and then browse what our application is doing at runtime including GPUs. As an example, 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)



PAPI

The Performance Application Programming Interface (PAPI) provides an important “standardized” API for accessing performance counters across a wide variety of hardware components from different vendors. By using this kind of universal layer, performance tools and other software desiring access to these counters can be written in a common fashion, ***making the addition of support for new hardware components far easier*** than directly using vendor provided APIs. PAPI enables this while adding minimal overhead for accessing these counters. *– Larry Kaplan HPE (3/13/2025)*



Start and stop counters at any time

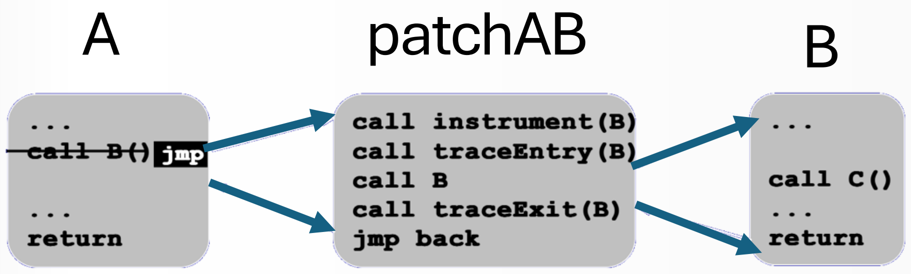
Start and stop counters at kernel boundaries

Dynist

The HPCToolkit performance tools, used by scientists for pinpointing and diagnosing performance and scalability problems in applications executing on exascale supercomputers, rely on the Dyninst infrastructure developed by the University of Wisconsin - Madison. HPCToolkit uses Dyninst to analyze CPU and GPU binaries to attribute metrics back to application source code. Without Dyninst, HPCToolkit would be unable to attribute performance to inlined code or loop nests in CPU and GPU code. **Such attribution is critical for understanding** the performance of modern template-based programming models, such as Sandia's Kokkos and LLNL's Raja.

– John Mellor-Crummey, Rice (3/16/2025)

Propagation and Trace Collection



```
...
~~call B()~~ jmp

...
return
```

```
call instrument(B)
call traceEntry(B)
call B
call traceExit(B)
jmp back
```

```
...

call C()
...
return
```

Overwrite call sites in the code

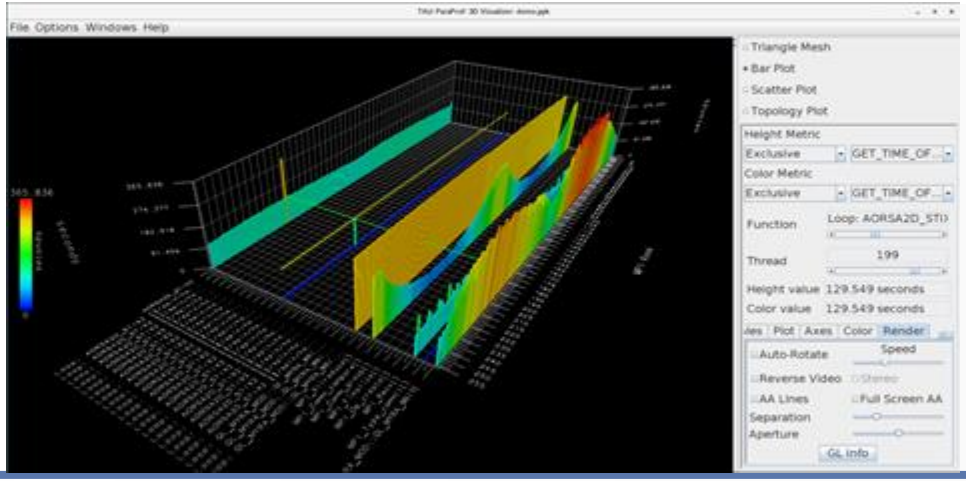
- Fast: done with local memory operations

Invoke tool routines before and after each call

TAU

Sarah Elliott, a Computational Chemist at Argonne National Laboratory, ***improved the total runtime of her application MESS by a factor of 3.8x in one day*** using the TAU performance toolset.

– work performed by *Sarah Elliott, ANL*



○ Triangle Mesh
● Bar Plot
○ Scatter Plot
○ Topology Plot

Height Metric: Exclusive, GET_TIME_OF...

Color Metric: Exclusive, GET_TIME_OF...

Function: Loop: AORSA2D_STD

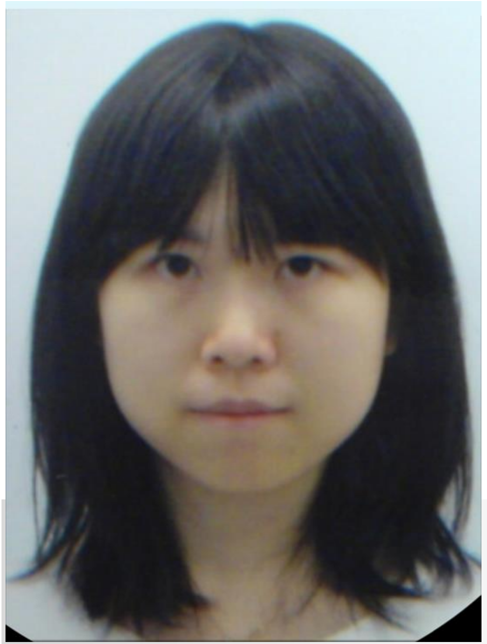
Thread: 199

Height value: 129.549 seconds

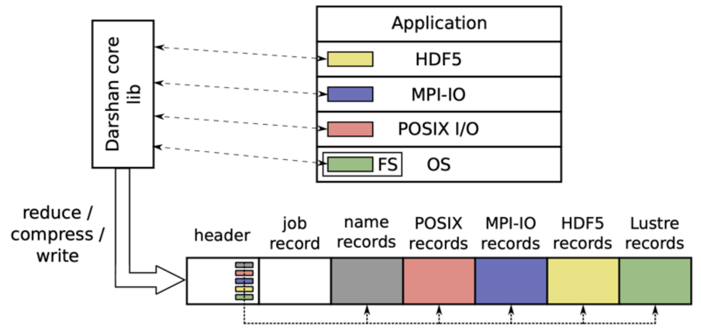
Color value: 129.549 seconds

□ Auto-Rotate
Speed
□ Reverse Video
□ Stereo
□ AA Lines
□ Full Screen AA
Separation
Aperture

Darshan



The Darshan I/O characterization tool has provided key insights into High Energy Physics (HEP) experiments such as ATLAS. For example, collaboration between our teams in the HEP-CCE (High-Energy Physics Center for Computational Excellence) project has led to **new insights that enable more efficient management of the exabyte-scale data produced by the ATLAS experiment**, with Darshan data helping drive critical data management design decisions for the ATLAS Athena software framework. Furthermore, practical experience with using Darshan in this framework has led to key enhancements to Darshan that should help broaden its impact to other software frameworks employing complex multi-process workflows. ***-Rui Wang, ANL (Mar-13-2025)***



High-level overview of Darshan's modularized runtime architecture and self-describing log file format