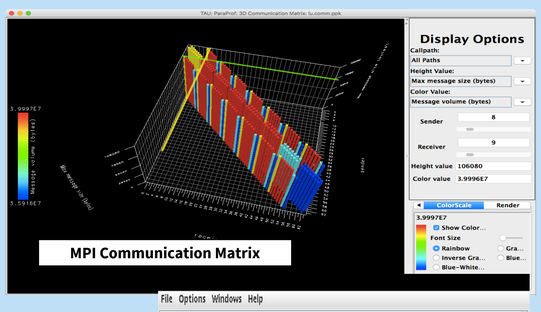
The TAU Performance System®

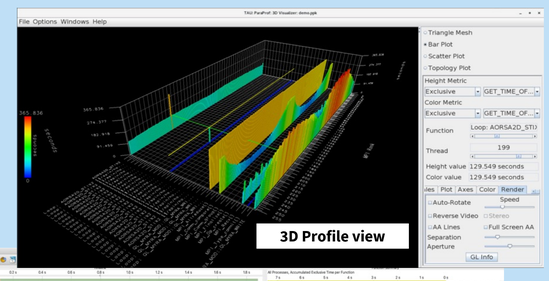
RAPIDS

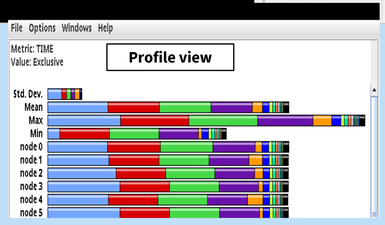
University of Oregon

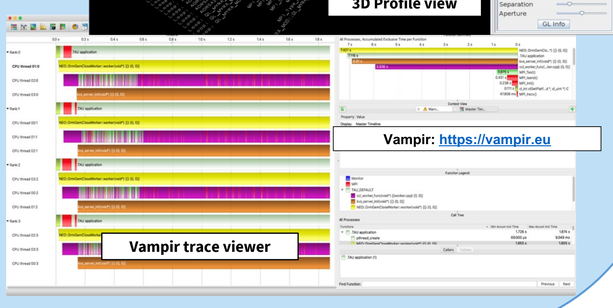
Robust HPC Application Measurement with TAU

The TAU Performance System® is a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, UPC, Java, Python, and utilizing all major programming models such as MPI, SHMEM, OpenMP/ACC/CL, CUDA, HIP, SYCL, and Pthreads. TAU (Tuning and Analysis Utilities) can gather performance information through instrumentation of functions, methods, basic blocks, and statements as well as event-based sampling. The API also provides selection of profiling groups for organizing and controlling instrumentation. The instrumentation can be inserted in the source code using LLVM compiler plugins, dynamically using binary modification, at runtime in the Java Virtual Machine, or manually using the instrumentation API or PerfStubs. Many new features and optimizations were recently added to TAU, including support for the new exascale system architectures and their preferred programming models. The new features include a new plugin API and several plugins, support for the Kokkos and Raja profiling interfaces, updated support for Python, PyTorch, TensorFlow, and Horovod, and removed threading limitations.







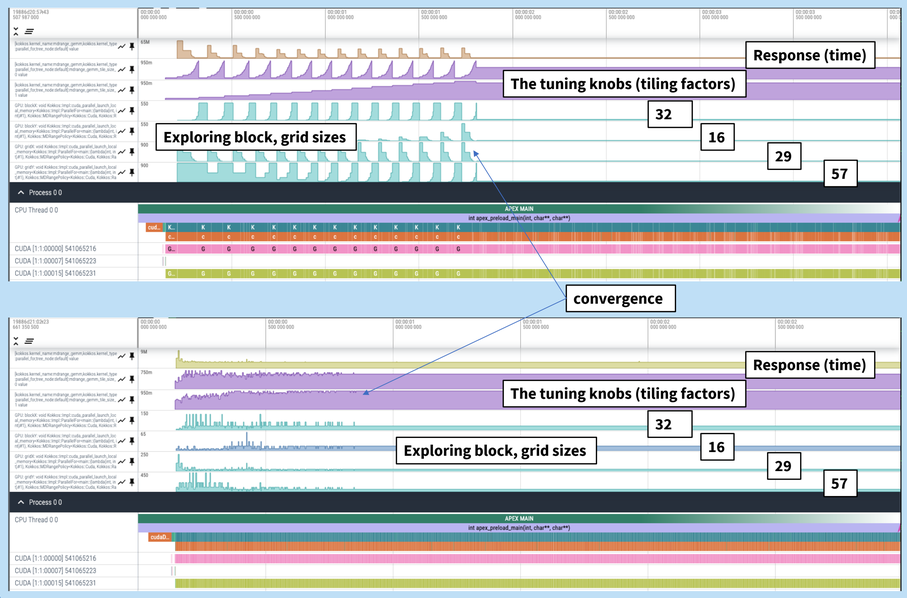


https://tau.uoregon.edu

Measurement and Runtime Optimization with APEX

The Autonomic Performance Environment for eXascale (APEX) is a runtime loadable library that provides **performance measurement** and **runtime adaptation** for HPC applications. APEX was originally designed for asynchronous multi-tasking runtimes (HPX) but also works with both conventional parallel models and asynchronous task schedulers. APEX focuses on the **task dependency graph**, not calling context tree. APEX supports HPX, C/C++ threads, OpenMP, OpenACC, Kokkos, Raja, CUDA, HIP, SYCL, StarPU, and we are currently integrating Iris and PARSEc. Runtime adaptation is provided by search algorithms such as Nelder Mead, simulated annealing, genetic search, hill climbing or other parametric search methods.

Kokkos example: mdrange_gemm 900x900 with exhaustive and simulated annealing search:



Exhaustive Search: Using window size of 2, tiling converges on block dimensions of (32,16,1) and grid dimensions (29,57,1), the defaults are (16,2,1) and (57,450,1) respectively. The runtime improved from a baseline of 3.95 seconds to 3.04 seconds using cached tuning results.

Simulated Annealing: Using window size of 2, tiling converges on block dimensions of (32,16,1) and grid dimensions (29,57,1), the defaults are (16,2,1) and (57,450,1) respectively. The runtime improved from a baseline of 3.95 seconds to 3.17 seconds using runtime simulated annealing. Cached results were equivalent to exhaustive tuning.

https://github.com/UO-OACISS/apex

User Space Monitoring with ZeroSum

Heterogeneous High Performance Computing (HPC) systems are highly specialized, complex, powerful, and expensive systems. Efficient utilization of these systems requires monitoring tools to confirm that users have configured their jobs, workflows, and applications correctly to consume the limited allocations they have been awarded. Historically system monitoring tools are designed for – and only available to – system administrators and facilities personnel to ensure that the system is healthy, utilized, and operating within acceptable parameters. However, there is a demand for user space monitoring capabilities to address the configuration validation and optimization problem. We developed a prototype tool, ZeroSum, designed to provide user space monitoring of application processes, lightweight processes (threads), and hardware resources on heterogeneous, distributed HPC systems.

LWP	Type	stime	utime	nvctx	ctx	CPUs
18351	Main†	1.54	15.17	332905	1838	1
18356	*ZeroSum*	0.42	1.10	194	1007	1
18385	Other	0.00	0.00	0	41	1-127‡
18405	OpenMP	0.31	13.09	232689	5	1
18407	OpenMP	0.44	12.93	353365	11	1
18408	OpenMP	0.21	13.22	92528	3	1
18409	OpenMP	0.47	12.93	394014	10	1
18410	OpenMP	0.37	13.03	302371	7	1
18411	OpenMP	0.41	12.97	348829	10	1

Table 1: Frontier results, default configuration. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

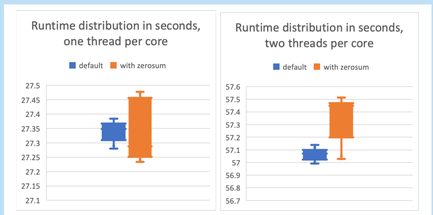
LWP	Type	stime	utime	nvctx	ctx	CPUs
18552	Main†	3.13	88.40	5	704	1-7
18561	*ZeroSum*	0.79	2.64	2	2790	7
18588	Other	0.00	0.00	0	41	1-127‡
18589	OpenMP	1.10	90.00	9	716	1-7
18590	OpenMP	1.10	93.00	8	724	1-7
18591	OpenMP	1.07	90.52	9	692	1-7
18592	OpenMP	1.10	89.83	14	766	1-7
18593	OpenMP	1.10	90.48	7	728	1-7
18594	OpenMP	1.10	91.93	300	849	1-7

Table 2: Frontier results, configuration requesting 7 cores per process. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

LWP	Type	stime	utime	nvctx	ctx	CPUs
18948	Main†	3.07	88.57	2	386	1
18954	*ZeroSum*	0.71	2.57	2	291	7
18981	Other	0.00	0.00	0	41	1-127‡
18992	OpenMP	1.18	96.36	0	422	2
18993	OpenMP	1.14	96.50	1	391	3
18994	OpenMP	1.18	96.46	0	381	4
18995	OpenMP	1.11	93.89	0	324	5
18996	OpenMP	1.14	93.29	0	370	6
18997	OpenMP	1.14	95.54	208	358	7

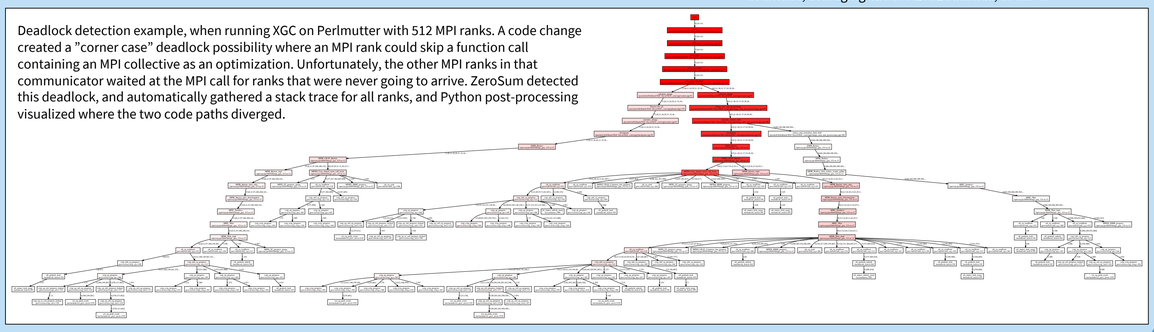
Table 3: Frontier results, configuration requesting 7 cores per process and binding OpenMP threads to cores. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

Three different configurations of miniQMC on Frontier, using 8 MPI ranks and 7 OpenMP threads per rank, 8 cores reserved for the system. The first launch is with without specifying the number of cores per process to srun, the second launch does specify the core count but doesn't use OpenMP binding settings. The third launch binds the threads to cores in a spread configuration. The *nvctx* value represents the number of non-voluntary context switches, and *ctx* represents the number of voluntary context switches (yields). By using the correct srun configuration settings and the correct OpenMP binding, thread contention is reduced and performance is improved overall.



miniQMC time distributions executed 10 times using one OpenMP thread per core (left). In this comparison, the distribution of times with ZeroSum is noisier, but there is no significant observation of measurable overhead. The right figure shows the time distributions using two OpenMP threads per core. In this comparison, the distribution of times with ZeroSum is both noisier and longer tailed, and does show an observation of overhead, averaging about 0.2752 seconds, or 0.5%.

Deadlock detection example, when running XGC on Perlmutter with 512 MPI ranks. A code change created a "corner case" deadlock possibility where an MPI rank could skip a function call containing an MPI collective as an optimization. Unfortunately, the other MPI ranks in that communicator waited at the MPI call for ranks that were never going to arrive. ZeroSum detected this deadlock, and automatically gathered a stack trace for all ranks, and Python post-processing visualized where the two code paths diverged.



https://github.com/UO-OACISS/zerosum

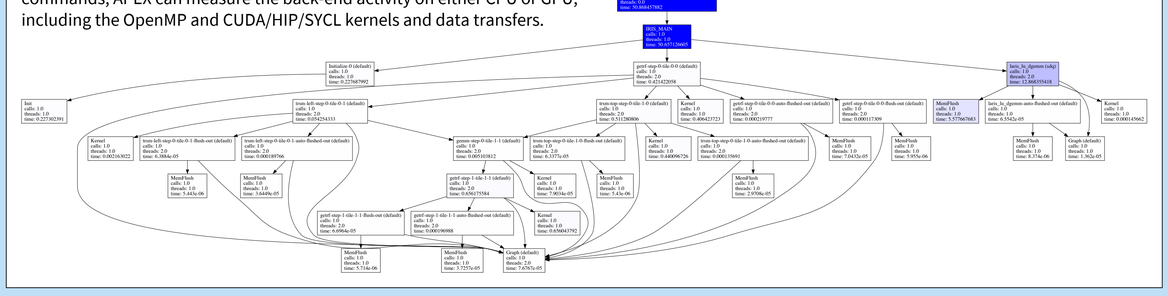
Software Sustainability Updates: Python 3.12, PerfStubs, and TaskStubs

PerfStubs is a "frictionless" instrumentation library and provides a plugin interface for performance tools. It can be disabled at configuration or run time, and it is integrated into several libraries as a git submodule: CAMTIMERS, PETSc, Ginkgo, ADIOS2. Provides *runtime* integration with TAU & APEX.
TaskStubs is a similarly designed instrumentation library for asynchronous tasking runtimes. We are working on integration with several libraries as a git submodule: IRIS, PaRSEC, StarPU. Provides *runtime* integration with APEX.
Python updates: Python 3.12 introduced a new, low-overhead tracing callback interface, PerfStubs is being extended to support it, provide PyTorch, Spark, and TensorFlow support, and have backwards compatibility for older runtime versions.

Left example: APEX profile output from the XGC Fusion simulation running on Fronter. CAMTIMERS events are passed to TAU and APEX through PerfStubs, and the profile includes measurement support provided by TAU/APEX including Kokkos, MPI, HIP API calls and asynchronous GPU activity.

Timer	total	calls	tot/call	yields	threads	tot/thr	%total	%wall
APEX MAIN	588.25	1.00	588.25	0.00	1.00	588.25	100.00	100.00
TOTAL	587.73	1.00	587.73	0.00	1.00	587.73	99.91	99.91
MAIN_LOOP	377.68	10.00	37.77	0.00	1.00	377.68	64.20	64.20
IPC_LOOP	235.68	20.00	11.78	0.00	1.00	235.68	40.06	40.06
hipDeviceSynchronize	149.40	168600.08	0.00	0.00	1.00	149.40	25.40	25.40
MPI Collective Sync	132.45	20064.08	0.01	0.00	1.00	132.45	22.52	22.52
INIT	125.85	1.00	125.85	0.00	1.00	125.85	21.39	21.39
i:SETUP	125.85	1.00	125.85	0.00	1.00	125.85	21.39	21.39
F0_GRID	106.06	10.00	10.61	0.00	1.00	106.06	18.03	18.03
F_SOURCE	106.06	10.00	10.61	0.00	1.00	106.06	18.03	18.03
F_COLLISON	97.18	5.00	19.44	0.00	1.00	97.18	16.52	16.52
SHIFT	86.86	20.00	4.34	0.00	1.00	86.86	14.77	14.77
Kokkos::parallel_for [HIP, Dev:0] pus...	86.76	230.00	0.38	0.00	1.00	86.76	14.75	14.75
GPU: void Kokkos::Impl::hip_parallel_...	84.64	210.00	0.40	0.00	1.00	84.64	14.39	14.39
ipc2:PUSH	76.56	10.00	7.66	0.00	1.00	76.56	13.02	13.02
ipc2:PUSHE	71.81	10.00	7.18	0.00	1.00	71.81	12.21	12.21
SHIFT_E	68.95	23.00	3.00	0.00	1.00	68.95	11.72	11.72

Integration example: APEX measurement of the IRIS runtime executing an LU factorization with the MatRIS library. In addition to the IRIS tasks and commands, APEX can measure the back-end activity on either CPU or GPU, including the OpenMP and CUDA/HIP/SYCL kernels and data transfers.



https://github.com/UO-OACISS/perfstubs
https://github.com/khuck/taskStubs

SciDAC Institute for Computer Science, Data, and Artificial Intelligence

Argonne NATIONAL LABORATORY · BERKELEY LAB · University of Delaware · kitware · Los Alamos NATIONAL LABORATORY · Northwestern University · University of Oregon · Brookhaven National Laboratory · UF University of Florida · Lawrence Livermore National Laboratory · The Ohio State University · OAK RIDGE National Laboratory · SCI www.sci.utah.edu · U.S. DEPARTMENT OF ENERGY · Office of Science