# A Comparative Analysis of Performance Tools in the Extreme-scale Scientific Software Stack (E4S 25.06)

## 1.0 Introduction: A Curated Ecosystem for Performance Insight

### 1.1 The Exascale Challenge: Managing Unprecedented Complexity

The modern era of scientific computing, driven by the goals of the Exascale Computing Project (ECP) 1, is defined by a fundamental shift in hardware and software complexity. The pursuit of exascale performance has led to heterogeneous system architectures that couple powerful, multi-core Central Processing Units (CPUs) from vendors like Intel, AMD, and Arm, with a diverse and potent array of GPU accelerators from NVIDIA, AMD, and Intel.1 As of late 2024, nine of the top ten fastest supercomputers are heterogeneous systems.2

This hardware heterogeneity has created a corresponding explosion in software complexity. A single scientific application may now simultaneously leverage multiple programming models and runtime systems.2 It is common for a simulation to be built on a foundation of C++ or Fortran, use the Message Passing Interface (MPI) for inter-node communication, OpenMP for on-node CPU threading, and a portable accelerator model like Kokkos, RAJA, or SYCL to target different vendor GPUs (which in turn compile down to vendor-specific models like CUDA, HIP, or oneAPI Level Zero).2

These programming models are layered upon one another, resulting in a "tightly integrated system" 2 whose performance characteristics are extraordinarily difficult to introspect, debug, and optimize. For application teams, understanding the performance characteristics of their codes on these complex architectures is a primary barrier to achieving performance goals.5 Identifying bottlenecks is no longer a simple matter of timing a "hot" function; developers must now ask complex, multi-layered questions: Is performance limited by CPU work, GPU kernel execution, or data movement between them? Is time being wasted in MPI synchronization, and if so, is it due to load imbalance or network contention? Which specific line of CPU source code is responsible for triggering a cascade of stalls in a GPU kernel?.6

### 1.2 E4S 25.06: A Unified Software Stack for HPC and AI

The Extreme-scale Scientific Software Stack (E4S), one of the key legacies of the ECP, was created to address this software challenge.1 The 25.06 release of E4S represents a mature, curated collection of over 130 scientific libraries, development tools, and AI/ML packages, all distributed via the Spack package manager.10 The fundamental purpose of E4S is to "accelerate the development, deployment and use of HPC-AI software, lowering the barriers for... users".1 It achieves this by providing a "consistent environment" 13 that is tested for interoperability and portability, scaling from developer laptops to the world's largest leadership-class supercomputers.1

A key pillar of E4S is its comprehensive support for hardware-agnostic development. The 25.06 release provides one of the largest collections of performance-portable, GPU-enabled libraries, with robust support for NVIDIA, AMD, and Intel GPUs alongside all major CPU architectures.1 This release includes updates for the latest hardware and software, including NVIDIA Blackwell (sm\_120) support, CUDA 12.8, ROCm 6.3.3, and oneAPI 2025.1.10

Within this vast collection, E4S provides a rich portfolio of performance analysis tools designed to help developers navigate the complexity of modern HPC applications. This portfolio includes well-known products such as HPCToolkit, TAU (Tuning and Analysis Utilities), and PAPI (Performance Application Programming Interface) 1, as well as the Caliper introspection system 15, and the integrated Score-P measurement framework 16, which provides data for analysis tools like Vampir 18 and Scalasca.8

### 1.3 An Integrated Toolset, Not a Collection of Tools

A superficial examination of the E4S performance tool portfolio might suggest redundancy. Multiple tools offer profiling, tracing, and GPU analysis capabilities. However, a deeper analysis reveals a sophisticated, highly integrated ecosystem designed around a "separation of concerns" philosophy. These tools are not mutually exclusive competitors; rather, they form an interoperable workflow, with different components specializing in different aspects of the performance analysis life cycle.

This integrated design is a core, albeit subtle, aspect of the E4S "user experience." For example, PAPI is not a standalone profiler but a foundational "middleware" library 20 that provides a universal API for accessing low-level hardware counters. It is the "great enabler" that higher-level tools like TAU and HPCToolkit leverage to provide their vendor-agnostic CPU and GPU counter support.21

Similarly, the Score-P ecosystem was explicitly created as a "joint undertaking" 22 to provide a common, scalable measurement infrastructure. Score-P itself is responsible *only* for instrumentation and data collection.17 This data is then consumed by specialized, and interoperable, analysis tools: CUBE for lightweight profile visualization 23, Scalasca for automated bottleneck analysis 8, and Vampir for deep, interactive trace visualization.18 The tools are so integrated that TAU can be configured to use Score-P as its measurement backend 25, and Vampir can consume traces generated by TAU.26

This report provides an expert-level analysis of the user experiences, technical capabilities, and comparative advantages of the performance tools within the E4S 25.06 stack. It is structured to help a computational scientist or HPC developer navigate this ecosystem, understand the distinct philosophy of each tool, and, most importantly, select the right tool—or workflow—for their specific performance-tuning challenge. The analysis will proceed from the foundational hardware-abstraction layer (PAPI), through the all-in-one analysis toolkits (TAU, HPCToolkit, Caliper), and conclude with the specialized workflow of the Score-P ecosystem (Score-P, CUBE, Vampir, Scalasca).

## 2.0 The Foundation: PAPI (Performance Application Programming Interface)

### 2.1 Philosophy: A Universal Middleware for Hardware Counters

The Performance Application Programming Interface (PAPI) is a foundational component of the E4S development toolset, but its role is often misunderstood. PAPI is not, by itself, a user-facing profiler. Instead, it serves as a critical *middleware* layer.20 Its philosophy is to provide a single, "consistent, operating-system-independent interface" 20 and a "universal interface and methodology" 27 for accessing the myriad, low-level performance counters embedded in modern hardware.

In the complex heterogeneous landscape of HPC, PAPI functions as a "universal translator".28 Modern CPUs, GPUs, and interconnects each expose their own proprietary, low-level interfaces for performance monitoring (e.g., model-specific registers on CPUs, NVIDIA's CUPTI, AMD's ROCprofiler-SDK, and Intel's oneAPI Level Zero).20 Attempting to support all of these in a high-level tool would be an untenable development burden.

PAPI’s role is to manage this complexity. It handles the intricate, vendor-specific details of accessing counters on each component, abstracting them away behind a single, stable, and coherent API. PAPI effectively "handles 30+ APIs with one simple API".28 This abstraction is the "great enabler" of the entire E4S performance tool ecosystem's portability. High-level tools like TAU and HPCToolkit are able to provide "vendor-agnostic" GPU profiling 3 precisely because they leverage PAPI as their backend. When a new hardware component or counter API is released (like the PAPI 7.2.0 rocp\_sdk component for AMD GPUs 27), the PAPI team integrates it, and *all* tools built on top of PAPI "gain the capability to measure data from these emerging architectures".27

### 2.2 User Experience and Capabilities

PAPI's "user" is primarily the *tool developer*.29 The intended audience for the PAPI library is the developer of a tool like HPCToolkit, not the end-user scientist. However, PAPI does expose a direct-call API for application developers who require raw, low-level access. This API is available for both C and Fortran 29 and is split into two levels:

1. **The High-Level API:** This is a simple, 8-function API designed for basic measurements.34 It allows a user to easily start, stop, and read "PRESET events" (a standardized, cross-platform set of events defined by PAPI, like PAPI\_L2\_DCM for L2 Data Cache Misses). This API is designed for simplicity but is limited to CPU-only events.34
2. **The Low-Level API:** This is the flexible, powerful interface that provides full access to all PAPI features. The user manages "Event Sets," which are user-defined groups of hardware events.34 This API allows access to native events (e.g., PAPI\_NATIVE\_<event>), which are specific to a given microprocessor, and is required for accessing GPU and other non-CPU components.32

The sheer breadth of components supported by PAPI is its defining capability. Through its collaborations with industry leaders like AMD, NVIDIA, and Intel 27, PAPI provides a unified interface to:

* **CPUs:** All major architectures, including Intel (Sapphire Rapids, etc.), AMD (Zen4), and ARM (Cortex).27
* **GPUs:** PAPI provides robust components for all three major vendors:
  + **NVIDIA:** via the CUPTI Profiling API.20
  + **AMD:** via the ROCm and ROCprofiler-SDK (rocp\_sdk) interfaces.20
  + **Intel:** via the oneAPI Level Zero interface.20
* **Power and Energy:** PAPI can read power and energy data, including from Intel's RAPL (Running Average Power Limit) 28 and NVIDIA's NVML.28
* **Interconnects and I/O:** PAPI includes components for monitoring interconnects like Cray Gemini/Aries and InfiniBand, as well as I/O systems like Lustre.28
* **Software-Defined Events (SDEs):** This is a powerful recent feature that allows *software libraries* (e.g., math libraries, runtimes) to register their own internal counters and expose them through the standard PAPI interface.20 This allows PAPI to "uncover critical software events" 37 and enables tools to monitor hardware and software events in a uniform way.36

### 2.3 When to Use PAPI (and its Limitations)

An application developer should choose to use the PAPI API *directly* only in a narrow set of circumstances: when they need extremely fine-grained, raw hardware counter data (e.g., "I need to know the exact L2 cache miss count for this specific loop" 38) and a higher-level tool like TAU or HPCToolkit does not provide a sufficient abstraction or metric. For over 99% of users, PAPI is the invisible foundation they use *indirectly* by running HPCToolkit, TAU, or Score-P.

Using PAPI directly comes with a non-trivial, expert-level "user experience" cost. Users must contend with several limitations:

* **Complexity:** As one user noted, "it has been a bit difficult for me to use it".39 Managing Event Sets, querying for event availability, and handling errors is a non-trivial programming task.30
* **Event Multiplexing:** Modern processors contain hundreds of "events" (signals that can be counted) but only a small number (e.g., 4-8) of physical "counters" (registers that do the counting).30 To measure more events than counters, PAPI must use *multiplexing*—periodically stopping the application, swapping which events are being counted, and resuming.40 This allows for comprehensive measurement but can significantly slow down the application, turning an hours-long run into a days-long run.40
* **Event Conflicts:** Not all events can be measured at the same time, even with multiplexing, as some events may use the same underlying hardware resource. A user must use a separate utility like papi\_event\_chooser to determine which event combinations are valid.38
* **Granularity:** Some hardware-provided metrics, such as RAPL power readings, are system-wide or package-wide and "cannot easily be attributed to individual threads".35

These limitations are precisely why the E4S ecosystem provides higher-level tools. These tools are designed to manage PAPI's complexity, handle event multiplexing transparently, and, most importantly, correlate the raw counter data with the application's source code, transforming raw data into actionable performance *insight*.

## 3.0 Core Measurement and Analysis Toolkits: A Comparative Deep Dive

While PAPI provides the foundation, the core of the E4S performance ecosystem consists of several "all-in-one" toolkits. These toolkits—TAU, HPCToolkit, and Caliper—each provide a complete workflow for measurement, analysis, and visualization. However, they are not redundant. Each is built on a fundamentally different philosophy for performance measurement, and the user's experience and the types of problems they can solve differ dramatically as a result.

### 3.1 TAU (Tuning and Analysis Utilities): The Versatile "Swiss Army Knife"

The TAU Performance System® is arguably the most versatile and flexible tool in the E4S portfolio. It is described as a "Swiss army knife" of profiling 41, designed from the ground up to be a portable, scalable toolkit that supports both *profiling* (collecting summary statistics, or "how much" time was spent) and *tracing* (collecting a timeline of individual events, or "when" things happened).42

#### Philosophy and User Experience: A "High-Control Cockpit"

TAU's philosophy is centered on flexibility and user control. It provides the user with a "high-control cockpit" offering multiple, distinct methods for instrumenting and measuring an application. The user *chooses* the method that best fits their application, build system, and analysis needs.

The primary instrumentation methods are:

1. **Runtime Interception (tau\_exec):** This is the easiest and most popular way to start. The user simply prefixes their application launch command (e.g., mpirun tau\_exec./app).46 No recompilation or source modification is required.44 tau\_exec works by preloading a TAU library that "wraps" runtime calls like MPI\_Send or cudaLaunchKernel. It can also be configured to use event-based sampling (TAU\_SAMPLING=1) 42 to profile user-level functions, giving a lightweight profile without instrumentation.
2. **Source Instrumentation (PDT):** For more detailed, automatic instrumentation, TAU provides a source-to-source translator based on the Program Database Toolkit (PDT).43 This tool parses the application's C, C++, or Fortran source code and automatically inserts TAU timer calls (e.g., TAU\_PROFILE\_START()) around functions, loops, and other constructs. This method requires recompiling the code using TAU's compiler-wrapper scripts (e.g., tau\_cc.sh, tau\_f90.sh).44
3. **Compiler-Based Instrumentation:** An alternative to PDT, this method relies on the compiler itself to insert profiling hooks. The user compiles with a special flag (e.g., GCC's -finstrument-functions) and again uses the TAU wrapper scripts, which link in a library that turns those hooks into TAU measurements.44
4. **Manual Instrumentation:** For ultimate control, the developer can directly edit the source code and insert calls to the TAU instrumentation API (e.g., TAU\_PROFILE\_START(), TAU\_PROFILE\_STOP()).46 This is useful for timing specific, developer-defined code regions.
5. **Selective Instrumentation:** Profiling an entire application can create significant overhead. TAU addresses this by allowing *selective instrumentation*.46 The user provides a "filter file" (a simple text file) that tells TAU to *only* instrument (or *not* instrument) specific functions or source files.44 This is a critical feature for focusing analysis on a region of interest and reducing measurement overhead.

#### Analysis, Language, and GPU Portability

After a run, TAU produces data that can be analyzed in two primary ways:

* **Profiling:** By default, TAU generates profile files that are viewed with paraprof.43 ParaProf is a powerful graphical interface for displaying performance results, showing aggregate statistics, per-thread/process data, and 3D visualizations of data on the machine's topology.43 TAU also provides PerfExplorer for cluster and correlation analysis of performance data from multiple runs.50
* **Tracing:** By setting an environment variable (TAU\_TRACE=1) 48, the user can switch TAU from profiling to tracing. This generates event traces that capture the dynamic, time-based behavior of the application.43 These traces can be visualized in tools like Vampir, Paraver, or JumpShot.26 TAU can also use the Score-P system internally to generate OTF2-formatted traces.52

TAU's **language support** is exceptionally broad, which is a key differentiator. It is a portable toolkit for parallel programs written in C, C++, Fortran, UPC, Java, and Python.43 Its support for Python, in particular, is a critical capability for the E4S 25.06 AI/ML stack, which includes packages like PyTorch, TensorFlow, and JAX.1 TAU has recently added support for low-overhead Python 3.12 instrumentation.53

TAU's **GPU portability** is its "killer feature." It is explicitly described as "one of the few tools that supports all vendor GPUs".41 This is achieved by providing instrumentation and measurement support for the full spectrum of HPC GPU programming models:

* **NVIDIA:** Full support for CUDA 42 and OpenACC.53 TAU integrates with CUDA compilers and intercepts runtime library routines to capture GPU computations.55
* **AMD:** Full support for ROCm 41 and its HIP programming model.41
* **Intel:** Full support for oneAPI Level Zero (L0) 41 and DPC++/SYCL applications.42
* **Portable Models:** TAU has deep, first-class support for high-level portable libraries, including Kokkos 41, RAJA 53, and OpenMP Target Offload directives.42

#### When to Use TAU

TAU is the tool of choice in several key scenarios:

* **Hybrid Applications:** When the application is a complex hybrid of programming models, especially **Python with C++**, or **MPI + OpenMP + Kokkos**. TAU's ability to trace and profile all these components simultaneously is unparalleled.
* **Maximum GPU Portability:** When the primary goal is to profile an application that must run across all three major GPU vendors (NVIDIA, AMD, and Intel).
* **Flexibility:** When a developer needs a single toolkit that can seamlessly switch between lightweight sampling (via tau\_exec -ebs) 48, full instrumentation profiling (via PDT), and deep-dive event tracing (via TAU\_TRACE=1).48
* **Power-User Control:** When the developer is a "power user" who is willing to manage TAU's extensive set of configuration options (via environment variables 48 and makefiles 49) to gain fine-grained control over the entire measurement and analysis process.

ParaTools, Inc., whose founders are the original developers of TAU 49, provides commercial support, training, and a hardened "ParaTools Pro for E4S" cloud image.58 ParaTools is the sole licensee of the TAU trademark and contributes extensively to its open-source development.52

### 3.2 HPCToolkit: Low-Overhead Sampling for Optimized Binaries

HPCToolkit offers a fundamentally different approach to performance analysis. Its philosophy is not based on instrumentation (inserting timers into code) but on *asynchronous statistical sampling* of timers and hardware performance counters.6 This design choice has profound and positive implications for the user experience.

#### Philosophy and User Experience: The "As-Is" Profiler

The primary "user experience" benefit of HPCToolkit is its ability to analyze *unmodified, fully optimized, dynamically-linked applications*.6 There is **no need to recompile, reinstrument, or even modify the build system.** This is a massive productivity win, especially for production codes with complex, "heavy" build systems, or for applications that link against binary-only vendor libraries.62

HPCToolkit's sampling-based approach also results in very low, controllable, and predictable measurement overhead, typically in the 1-5% range.6 This makes it an ideal tool for large-scale parallel jobs and long-running simulations, where the high overhead of instrumentation-based tools can be prohibitive.62

The user workflow is straightforward and consists of three steps:

1. **Measurement (hpcrun):** The user simply prefixes their job-launcher command with hpcrun and specifies the events to sample (e.g., hpcrun -e REALTIME -e gpu=nvidia <app>).6 This component uses asynchronous sampling triggered by system timers (for time) and PAPI-managed performance monitoring unit (PMU) events (for hardware counters) to collect call path profiles.6
2. **Binary Analysis (hpcstruct):** Because the binary is optimized, hpcstruct must be run *once* to analyze the application's executable and libraries. It recovers information about the code structure, "relates object code to source code files, procedures, loop nests, and identifies inlined code".6
3. **Analysis (hpcprof-mpi and hpcviewer):** hpcprof-mpi combines the measurement data from hpcrun with the structural data from hpcstruct to generate a performance database. This database is then explored using the hpcviewer (for profiles) or hpctraceviewer (for traces) GUIs.6

#### Key Feature: Heterogeneous Call Path Attribution

HPCToolkit's "killer feature" is its robust and unique ability to "attribute the costs of GPU work to heterogeneous calling contexts".3

Other profilers might only tell a user *that* cudaLaunchKernel was called, or *how long* a kernel ran on the GPU. HPCToolkit answers a much more powerful and specific question: **"Which line of CPU source code is responsible for this GPU bottleneck?"**

It achieves this by collecting call path profiles on the CPU (via sampling) and monitoring GPU operations and instruction-level metrics *within* GPU kernels.4 It can then connect the two. A developer can use hpcviewer to navigate to a "hot" line of CPU code (e.g., a Kokkos parallel\_for) and then see the detailed GPU performance metrics *caused* by that line.4 This includes instruction-level stall metrics from PC sampling on NVIDIA GPUs, such as GINS:STL\_GMEM (stalled on global memory) or GINS:STL\_IDEP (stalled on instruction dependency).4 This precise attribution—from CPU source code line to GPU instruction stall reason—is a capability that is largely unique to HPCToolkit and is invaluable for optimizing heterogeneous codes.

#### GPU Portability and Language Support

As an ECP-funded project, HPCToolkit was designed from the ground up to be "production-ready, vendor-agnostic".3 Its GPU support is comprehensive and robust across all three major vendors, supporting C, C++, and Fortran applications 4:

* **NVIDIA:** Full support for operation-level monitoring and instruction-level PC sampling (e.g., hpcrun -e gpu=nvidia,pc).4
* **AMD:** Full support for operation-level monitoring via roctracer (e.g., hpcrun -e gpu=amd).4 Instruction-level measurement is in progress.69
* **Intel:** Full support for operation-level monitoring via oneAPI Level Zero (e.g., hpcrun -e gpu=level0) and fine-grain measurement via binary instrumentation of GPU kernels.4
* **Portable Models:** It works seamlessly with high-level portable models like Kokkos, RAJA, OpenMP, SYCL, DPC++, and OpenACC.4

#### When to Use HPCToolkit

HPCToolkit's combination of low overhead and zero-effort instrumentation makes it the ideal "first-pass" tool for almost any performance investigation.

* **The Default First Choice:** For any C, C++, or Fortran application (serial, MPI, OpenMP, or hybrid), HPCToolkit should be the **first tool you try**. Its "run-it-as-is" user experience provides the fastest time-to-insight.
* **Production/Optimized Code:** When you need to profile a **production, fully optimized binary** and cannot (or will not) recompile it.
* **Large-Scale/Long-Running Jobs:** When you need to profile a large-scale application where the 1-5% overhead of sampling is acceptable, but the 10-100x+ overhead of instrumentation is not.
* **Heterogeneous Code:** When your primary question is **hotspot analysis in a GPU-accelerated code** and, most importantly, **attributing GPU costs back to your CPU source code.**

The primary trade-off is that its sampling-based nature provides a *statistical summary*.63 It is exceptionally good at finding hotspots (where time is spent), but it is not designed to capture *every single* communication or synchronization event. For debugging rare, transient, or dynamic timeline-based events (like a single stalled MPI message), a trace-based tool is more appropriate.

### 3.3 Caliper: The Developer-Integrated "Toolbox in a Library"

Caliper represents a third, distinct philosophy. It is not an external tool that wraps or instruments an application; it is a "toolbox in a library".70 It is an "application introspection system" 73 that is *linked into* an application by the developer to build performance analysis capabilities directly into the code itself.

#### Philosophy and User Experience: "Always-On" Developer-Led Instrumentation

The Caliper "user experience" is centered on the *application developer*. The developer uses Caliper's C, C++, and Fortran API to *manually* annotate regions of interest in the source code.72

The annotation API is simple and macro-based:

* CALI\_CXX\_MARK\_FUNCTION marks the scope of an entire C++ function.72
* CALI\_MARK\_BEGIN("region\_name") and CALI\_MARK\_END("region\_name") are used to mark arbitrary code blocks.72
* CALI\_CXX\_MARK\_LOOP\_BEGIN(id, "loop\_name") and CALI\_CXX\_MARK\_LOOP\_END(id) are used to profile loops.74

The core benefit of this "always-on" design is that these annotations, by default, add *negligible* performance overhead.72 The Caliper library does not record any data unless it is explicitly activated. This *activation* is the key to the Caliper workflow. A developer can use Caliper's ConfigManager API to programmatically enable profiling *at runtime* 74, often by tying it to a simple command-line argument that they add to their application.72

For example, an end-user of a Caliper-enabled application could simply run ./app --profile, and the application itself would turn on Caliper's "runtime-report" service and print a time profile at the end.72 This makes performance analysis a *feature* of the application, not an external, complex process.

#### Key Feature: Semantic Context and Metadata

Caliper's "killer feature" is its ability to capture *application semantics* and *metadata*. While sampling-based tools like HPCToolkit capture the *call path*, they often miss the application-level context that is only available through source-code annotations.76

Caliper is built on a "flexible key:value data model".71 This allows a developer to record not just timings, but *any* arbitrary metadata and associate it with a profile. For example, a developer can add annotations to record the current time step, the solver algorithm being used, or the size of a data structure:

cali\_set\_global\_int\_byname("config1", config1); 75

cali\_set\_global\_string\_byname("solver\_type", "GMRES");

This capability transforms Caliper from a simple *profiler* into an *introspection framework*. It enables powerful, multi-run analyses that correlate performance with application-specific logic. A user can collect data from hundreds of runs and then ask a question like, "How does the performance of the 'solve' region (from CALI\_MARK\_BEGIN("solve")) change as I vary the 'config1' parameter?".73 This semantic analysis is something most external tools cannot provide.

#### GPU Portability and Language Support

Caliper provides targeted, first-class support for C, C++, and Fortran applications.72 It also supports parallel programming models like MPI, OpenMP, Kokkos, and RAJA.70

Its GPU support is focused and robust for the two most common platforms:

* **NVIDIA (CUDA):** Requires Caliper to be built with CUpti support (-DWITH\_CUPTI=On).78
* **AMD (ROCm):** Requires Caliper to be built with roctracer support (-DWITH\_ROCTRACER=On).78

Caliper's configuration services can then be used to profile specific aspects of GPU execution. Users can enable options to:

* **Profile Host-side API functions:** profile.cuda or profile.hip reports time spent in calls like cudaMemcpy and cudaLaunchKernel.78
* **Profile Device-side Activities:** cuda.gputime or rocm.gputime measures the *actual* time spent on the device for kernel executions and memory copies.78
* **Profile Memory Copy Volume:** cuda.memcpy reports the *volume* of data (in Megabytes) copied between the CPU and GPU.78

#### When to Use Caliper

Caliper is not a general-purpose replacement for HPCToolkit or TAU. It is a specialized tool for a specific, and very important, use case.

* **Application Developers:** When you are the **developer of an application or library** and want to build profiling capabilities *directly* into your code.
* **User-Friendly Profiling:** When you want to provide your *end-users* with a simple, non-invasive way to get a performance report (e.g., ./app --profile).72
* **Semantic Analysis:** When you need to **correlate performance data with application-specific metadata** (e.g., algorithm parameters, problem size, time step).
* **Targeted GPU Metrics:** When you want to programmatically query specific GPU metrics, such as memory-copy volume, from *within* your application.78

## 4.0 The Score-P Ecosystem: A Unified Workflow for Profiling and Tracing

The fourth major component of the E4S performance portfolio is not a single tool, but an integrated *ecosystem* of tools designed to work together. This ecosystem, centered around the Score-P measurement infrastructure, perfectly embodies the "separation of concerns" philosophy. One tool (Score-P) handles all measurement, while other specialized tools (CUBE, Vampir, Scalasca) handle distinct forms of analysis. This approach provides a powerful, scalable, and standardized workflow for deep performance analysis.

### 4.1 Score-P: The Common Measurement Infrastructure

Score-P is a "highly scalable and easy-to-use tool suite" 16 whose sole purpose is to *measure* the performance of parallel applications. It was jointly developed by leading HPC performance tool groups to act as a *unified measurement infrastructure*.22 It does *not* analyze data itself; it generates standardized performance data formats that are then *consumed* by other analysis tools, including CUBE, Vampir, Scalasca, and even TAU.16

#### Philosophy and User Experience: The "Profile First, Then Trace" Workflow

The Score-P user experience is built around a powerful and highly recommended workflow: **"Profile First, Then Filter, Then Trace."**.23 This workflow is designed to manage the primary challenge of tracing: data volume. A full, unfiltered event trace of a large parallel application can be "prohibitively large" 84, often generating terabytes of data and introducing significant measurement overhead.

The Score-P workflow intelligently avoids this problem in a series of steps:

1. **Instrumentation:** The user instruments their application *once* by simply prepending the scorep command to their normal compile and link commands (e.g., scorep mpicc..., scorep-ifc...).23 Score-P's compiler wrapper automatically adds instrumentation for function calls, MPI, OpenMP, and other supported paradigms.
2. **Profiling Run:** The user first performs a *profiling* run. By default, Score-P is configured to generate a lightweight summary profile (SCOREP\_ENABLE\_PROFILING=true, SCOREP\_ENABLE\_TRACING=false).23 This run produces a profile.cubex file 16, which is a compact summary of where time was spent in the application.
3. **Analysis & Filtering:** The user analyzes this small profile (e.g., with the CUBE browser 23) to identify the "hot" functions. More importantly, the user runs the **scorep-score** utility on the profile.23 This critical tool analyzes the profile and *estimates* the size of a trace file if one were to be generated.82 It highlights which functions are called most frequently and contribute most to the trace data. Based on this, the user creates a filter file to exclude short, frequently called, or uninteresting functions from measurement.23
4. **Filtered Tracing Run:** The user re-runs the *exact same instrumented binary*. No recompilation is needed. This time, they set environment variables to enable tracing and apply the filter (SCOREP\_ENABLE\_PROFILING=false, SCOREP\_ENABLE\_TRACING=true, SCOREP\_FILTERING\_FILE=my.filter).81
5. **Data Generation:** This filtered run produces a much smaller, manageable, and low-overhead event trace in the standardized **Open Trace Format 2 (OTF2)**.16

This workflow is the "golden path" for using Score-P. It uses a fast, low-overhead profile to intelligently guide the creation of a targeted, low-overhead trace, saving the user from being "drowned in data."

#### GPU Portability and Language Support

Score-P provides comprehensive, scalable measurement for C, C++, and Fortran applications. Its support for parallel programming models is a key strength, with continuous testing for Serial, MPI, SHMEM, OpenMP, and Pthreads.17

Its GPU portability is also comprehensive, with support for:

* **NVIDIA CUDA** 17
* **AMD HIP** 17
* **OpenCL** 17
* **OpenACC** 17

This means Score-P can capture and record events from all these different models, integrating them into a single, unified trace or profile.

#### When to Use Score-P

A user chooses to use Score-P *when they want to use Vampir or Scalasca*. It is the official, preferred, and integrated data-collection engine for those analysis tools. It is the starting point for any deep-dive *trace analysis* or *automated bottleneck analysis* in the E4S ecosystem. Its robust, scalable, and standardized workflow makes it the gold standard for measuring complex hybrid MPI + OpenMP + GPU applications.88

### 4.2 CUBE: The Profile Analyzer for Score-P and Scalasca

CUBE is the dedicated *profile analysis tool* for the Score-P ecosystem.23 It is a lightweight, graphical browser designed to read and display the profile data generated by Score-P (the \*.cubex files) and the bottleneck reports generated by Scalasca.16

CUBE's "user experience" is based on its powerful "three-dimensional" browsing interface 23:

1. **Performance Property (Metric):** The "what." This pane shows the list of available metrics, such as "Time," "MPI Time," "Visits," or custom bottleneck metrics from Scalasca.23
2. **Call Tree (Program):** The "where." This pane shows the application's call tree, allowing the user to drill down from main() into the specific functions and loops responsible for the cost.23
3. **System Location (System):** The "who." This pane shows the parallel system topology, such as a list of MPI processes and their threads.23

All three panes are coupled. A user can click on a metric (e.g., "Time"), a function (e.g., compute\_rhs), and a process (e.g., "Rank 0"), and immediately see the value. More powerfully, they can click on "Time" and compute\_rhs and see a color-coded "severity matrix" showing how the time for that function is distributed across *all* processes, instantly revealing load imbalance.23

CUBE is the primary tool for the "Analysis & Filtering" step of the Score-P workflow. It provides the quick, high-level overview of hotspots needed to make an informed decision about what to trace.

### 4.3 Vampir: The Interactive "Microscope" for Event Traces

Vampir is the "V" in the "VI-HPS" (Virtual Institute for High Productivity Supercomputing) toolset, and it is the premier tool for *interactive event trace visualization*.18 It is not a measurement tool; it is a "full featured tool suite" that *consumes* the OTF2-formatted event traces generated by Score-P 26 or, in some cases, TAU.18

#### Philosophy and User Experience: Deep, Interactive Timeline Exploration

If CUBE provides the "what" (the hotspot), Vampir provides the "why" (the dynamic behavior). Its philosophy is to provide a "graphical analysis framework" 18 that allows a user to explore the *timeline* of their parallel application's execution in microscopic detail.7

The main Vampir display is a "timeline chart" (often called a "Gantt chart") showing every process/thread as a horizontal line, with time advancing from left to right.7 This timeline is colored by the *activity* the process is engaged in (e.g., "User Code," "MPI\_Send," "MPI\_Recv," "OpenMP Barrier").

Vampir's key "user experience" feature is its "powerful zooming and scrolling functionality".18 A user can view the entire 30-minute run at once, then click-and-drag to zoom into a single 500-millisecond time window, then zoom further to inspect a single 10-microsecond MPI message, and then zoom *back out* instantly. All other windows in the GUI (e.in., function summaries, communication statistics) update in real-time to reflect *only* the currently selected time interval.92

This interactivity is its power. Vampir is the tool a user reaches for to answer questions like:

* "My profile shows Rank 5 is slow, but *why*? (The user can zoom in and *see* that Rank 5 is stuck in MPI\_Recv, waiting for a message from Rank 4, which is still busy in a computation.)"
* "What happens in my application execution during a given time in a given process or thread?" 7
* "What do my application's communication patterns *actually* look like?" 7

For very large traces, the Vampir ecosystem uses a client-server model. The VampirServer backend runs on the HPC system (or a login/analysis node) and performs the data-intensive analysis operations, while the lightweight Vampir GUI client runs on the user's local workstation.90

#### When to Use Vampir

Vampir is not a "first-look" tool. It is the "deep-dive" tool you use *after* a profile has identified a problem that cannot be explained by summary statistics.

* **Dynamic Behavior:** When you need to understand the **dynamic, time-dependent behavior** of your application.
* **Load Imbalance:** When you need to debug the *cause* of **load imbalance** (e.g., processes arriving at a barrier at different times).
* **Communication/Synchronization:** When you need to analyze **complex communication patterns** or debug **synchronization overhead**.7
* **Timeline "Why?":** When your question is not "where is time spent?" but "why is this iteration slow?" or "what is this process *waiting for*?".92

### 4.4 Scalasca: The Automated Bottleneck Detector

Scalasca (Scalable Performance Analysis) is the final piece of the Score-P ecosystem. Like Vampir, it is a trace-based analysis tool. However, its philosophy is completely different. While Vampir is an *interactive* "microscope" for human exploration, Scalasca is an *automated trace analyzer* designed to *programmatically* find performance bottlenecks.8

Scalasca's specialty is identifying "potential performance bottlenecks – in particular those concerning communication and synchronization".8

#### Philosophy and User Experience: Automated Analysis Workflow

The Scalasca "user experience" is that of a powerful, automated analysis command. It *also* uses the Score-P data (both the CUBE4 profile and the OTF2 trace) as its input.19

The workflow is as follows:

1. The user first generates a Score-P profile and trace, following the standard "profile-first, filter-then-trace" workflow.95
2. After the filtered tracing run is complete, the user invokes the Scalasca analysis command (e.g., scalasca -analyze or scan -t).95
3. **Key Feature:** The Scalasca trace analyzer is *itself a parallel program*.24 It runs in parallel (often as part of the same batch job) and performs a "replay-based trace analysis".24 This parallel-analysis design is what allows Scalasca to scale to massive core counts, demonstrated up to 1.8 million threads.98
4. During this analysis, Scalasca "searches for inefficiency patterns, wait states, and the critical path".24 It automatically identifies and quantifies the time all processes spent in states of inefficiency, such as "MPI Wait at Barrier" or "Late Sender / Late Receiver."
5. **Output:** Scalasca does *not* produce its own visualizer. Instead, it outputs a *new CUBE4 file*.24 This file contains all the original profile data from Score-P, but *augmented* with new, high-level bottleneck metrics (e.g., "Wait state time").24

The user's final step is to open this *Scalasca-generated* CUBE4 file in the **CUBE browser**. They can then, for example, click on the "MPI Wait at Barrier" metric and instantly see in the call-tree and system panes exactly *where* in the code and on *which* processes this bottleneck is occurring.89

#### GPU Portability and Language Support

Scalasca is primarily focused on MPI, OpenMP, Pthreads, and hybrid MPI+OpenMP/Pthreads applications.24 Its analysis is centered on host-side communication and synchronization.

This focus is reflected in its GPU support. The latest release (v2.6.2) "add[s] metric hierarchies for HIP and OpenMP target offloading".19 However, the release notes explicitly state a critical limitation: **"The trace analysis still only supports host-side events!"**.19 This means that while Scalasca can analyze the *host-side* overheads of an application that *uses* GPUs (e.g., time spent waiting for an OpenMP target region to complete), it does not currently perform automated bottleneck analysis *within* the GPU kernels themselves.

#### When to Use Scalasca

Scalasca is a specialist tool. It is the right choice when the performance problem is one of *scalability*.

* **Scalability Bottlenecks:** When your application runs well on 10 nodes but its performance collapses on 100 nodes, and you suspect the cause is **MPI or OpenMP overhead**.
* **Communication/Synchronization:** When you want to find and quantify time spent in **wait states** (e.g., waiting at barriers, waiting for messages).8
* **Automated Analysis:** When you want a concise, **automated report** identifying these bottlenecks, rather than trying to find them manually by exploring a complex Vampir trace.
* **Critical Path Analysis:** When you need to identify the "critical path" of execution that is determining your application's total runtime.24

## 5.0 A Guide for Tool Selection: When to Use Which E4S Tool

The E4S 25.06 stack provides a comprehensive but complex portfolio of performance tools. The key to navigating this ecosystem is to understand that these tools are not redundant; they are distinct, specialized, and often interoperable. The choice of tool should be driven by the specific *question* the user is trying to answer, the application's *characteristics* (language, parallel model), and the *analysis* required.

This section synthesizes the preceding analysis into a direct guide for tool selection, beginning with a comprehensive comparison table and followed by scenario-based recommendations.

### 5.1 Comparative Summary of E4S 25.06 Performance Analysis Tools

The following table provides a high-level summary of the primary tools and workflows, their core philosophies, and their suitability for different problem classes.

| **Tool / Ecosystem** | **Primary Philosophy** | **Measurement Method** | **Key User Experience** | **Primary Output** | **GPU Portability (NVIDIA/AMD/Intel)** | **Best For... (Target Problem Class)** |
| --- | --- | --- | --- | --- | --- | --- |
| **PAPI** | Hardware-Abstraction API | Direct Low-Level API Calls | Used *by* other tools; or, an expert-only C/Fortran API 20 | Raw Hardware Counter Data | **Excellent** (Foundation for all 3: CUPTI, ROCP\_SDK, L0) 20 | Tool developers; or, experts needing raw HW counters (e.g., L2 misses) 34 |
| **TAU** | Flexible "Swiss Army Knife" 41 | Instrumentation (Source, Compiler, Runtime) & Sampling 44 | Powerful "cockpit" with many options; tau\_exec for easy start 46 | Profiles (viewed in paraprof) / Traces (OTF2) 43 | **Excellent** (All 3 vendors + Kokkos, SYCL, OpenMP Offload) 41 | Hybrid-language code (Python/C++), max GPU portability, Kokkos/RAJA, switching between profiling & tracing 41 |
| **HPCToolkit** | Low-Overhead Sampling | Asynchronous Statistical Sampling (Time & HW Counters) 6 | Run on optimized binary "as-is"; No recompile; Low (1-5%) overhead 6 | Profiles/Traces (viewed in hpcviewer) 6 | **Excellent** (All 3 vendors, attributes GPU cost to CPU code) 3 | C/C++/Fortran hotspot analysis, profiling production binaries, attributing GPU stalls to CPU source code 3 |
| **Caliper** | Developer-Integrated Toolbox 70 | Manual Source-Code Annotation (C/C++/Fortran) 72 | Link library, add macros to code; "Always-on," activated at runtime 72 | Profiles (JSON/Cali) with Application Metadata 71 | **Good** (CUDA, ROCm) 70 | Application/library introspection, correlating performance with *application metadata* (e.g., solver type) 73 |
| **Score-P + CUBE** | Unified Profiling Workflow | Instrumentation (Source, Compiler, Runtime) 85 | "Profile-first" workflow; View summary in 3D CUBE browser 23 | Summary Profile (CUBE4 format) 16 | **Excellent** (All 3 vendors + OpenACC/OpenCL) 17 | Quick, scalable hotspot analysis for hybrid MPI+OpenMP+GPU codes; 1st step before tracing 23 |
| **Score-P + Vampir** | Interactive Trace Visualization 18 | (Uses Score-P's filtered trace data) 82 | "Microscope" for dynamic behavior; Interactive, zoomable timeline 18 | Event Trace (OTF2 format) 26 | **Excellent** (Visualizes all data Score-P captures, including GPU events) 100 | Deep-dive analysis of dynamic behavior, load imbalance, and complex communication patterns 7 |
| **Score-P + Scalasca** | Automated Bottleneck Analysis 8 | (Uses Score-P's trace data) 19 | Run parallel analysis command; *Automates* trace analysis 24 | Bottleneck Report (CUBE4 format) 19 | **Good** (Host-side analysis of offload; does not analyze *inside* kernels) 19 | Finding and quantifying **MPI/OpenMP synchronization and wait states** (e.g., barrier wait time) 94 |

### 5.2 Scenario-Based Tool Selection Guide

The following scenarios map common user problems to the recommended E4S tool or workflow.

#### "I have a production C++/Fortran code. I need to find the CPU/GPU hotspots with minimal effort and without modifying my build."

* **Recommendation:** Start with **HPCToolkit**.
* **Reasoning:** HPCToolkit is the ideal "first-pass" tool. Its sampling-based approach requires *no source code modification or recompilation*.6 It works on fully optimized, dynamically-linked binaries, avoiding any need to alter a complex production build system. Its low (1-5%) overhead 6 makes it suitable for running on a representative, large-scale problem. It will immediately show you the "hot" functions and loops. If it's a GPU code, HPCToolkit's key strength is its ability to attribute GPU kernel costs and instruction-level stalls back to the *specific line* of CPU source code that launched them, providing uniquely actionable insights.3

#### "My HPCToolkit profile shows a bottleneck, but I don't understand the ***dynamic behavior***. I need to know ***why*** it's slow, not just ***where***."

* **Recommendation:** Use the **Score-P + Vampir** workflow.
* **Reasoning:** HPCToolkit provides a *statistical summary* (a profile) of where time was spent.63 It cannot easily show dynamic, time-dependent interactions. Vampir is a *trace visualizer* that provides an interactive "microscope" into the application's timeline.7 To use it, you must follow the Score-P workflow: 1) Instrument your code with Score-P.85 2) Run a profile and use scorep-score to create a filter.23 3) Run a filtered trace.83 4) Load the resulting OTF2 trace into Vampir.26 This will allow you to *visually scrub* the timeline to see the complex interactions (e.g., process A is idle, *waiting* for a message from process B) that a profile cannot show.

#### "My application is scaling poorly. I suspect my MPI or OpenMP synchronization is the problem."

* **Recommendation:** Use the **Score-P + Scalasca** workflow.
* **Reasoning:** This is the *exact* problem Scalasca was designed to solve.8 While you *could* try to find wait states manually in Vampir, Scalasca *automates* this process.94 It runs a parallel trace analysis 24 that automatically detects and quantifies time wasted in "inefficiency patterns," especially communication and synchronization wait states.24 It produces a CUBE file 24 that explicitly reports metrics like "Time spent waiting at MPI\_Barrier," which you can then view in the CUBE browser to pinpoint the problem's location in your code.89

#### "When should I choose TAU over HPCToolkit or Score-P?"

* **Recommendation:** Use **TAU** in three specific cases.
* **Reasoning:**
  1. **Language:** Your application is a **hybrid involving Python or Java**.43 HPCToolkit and Score-P are primarily focused on C, C++, and Fortran. TAU's first-class support for Python is critical for the AI/ML portion of the E4S stack.1
  2. **Versatility:** You want a *single tool* that can do "good-enough" sampling (via tau\_exec -ebs) 48, full instrumentation profiling (via PDT) 44, and event tracing (via TAU\_TRACE=1) 48 without needing to switch between different tool ecosystems.
  3. **Portability Models:** You are heavily invested in a specific portable model like **Kokkos**, and TAU's deep integration and human-readable naming for Kokkos regions are a "must-have" feature.41

#### "I am an application developer. How can I let my ***users*** get a performance report without them needing to learn a complex tool?"

* **Recommendation:** Use **Caliper**.
* **Reasoning:** Caliper is a *library*, not an external tool.70 As the developer, you link Caliper into your application and add the source-code annotations (e.g., CALI\_CXX\_MARK\_FUNCTION).72 These annotations have negligible overhead when inactive.72 You can then add a simple command-line flag (e.g., --profile) to your application's main() function that *activates* Caliper's runtime-report service.72 This provides the simplest possible "user experience": your user just runs ./app --profile and gets a report. They do not need to learn Score-P, set environment variables, or use wrapper scripts.

#### "I need to correlate my application's performance with its internal parameters, like which solver I'm using or what time step I'm on."

* **Recommendation:** Use **Caliper**.
* **Reasoning:** This is Caliper's unique, "killer feature." Its flexible key:value data model 71 is explicitly designed for this. You can use the Caliper API to embed this *semantic metadata* (e.g., cali\_set\_global\_string\_byname("solver\_type", "GMRES")) directly into the performance data.75 This allows you to perform complex, multi-run analyses to understand how application-level decisions impact performance.73

#### "How do I choose a tool for my NVIDIA/AMD/Intel GPU code? They all seem to support them."

* **Recommendation:** Your choice should be based on the *question*, not the GPU vendor. All major E4S tools have excellent, ECP-mandated portability.
* **Reasoning:** As demonstrated in the E4S 25.06 release, the entire stack is built for cross-platform, GPU-enabled computing.1 PAPI provides the foundational support for CUPTI, ROCP\_SDK, and Level Zero.20 The other tools build on this.
  + To find **hotspots** and **attribute GPU cost to CPU code:** Use **HPCToolkit**.3
  + To get a **versatile profile** of a hybrid GPU/CPU code (e.g., Python+CUDA or Kokkos on ROCm): Use **TAU**.41
  + To see a **timeline** of MPI + GPU (e.g., CUDA/HIP) events: Use **Score-P + Vampir**.17
  + To measure time in specific **API calls** (e.g., cudaMemcpy) or data-transfer *volumes* from *within* your application: Use **Caliper**.78

## 6.0 Concluding Analysis: An Ecosystem for Every Question

The E4S 25.06 release 10 provides a world-class, comprehensive, and, most importantly, *integrated* performance analysis ecosystem. The portfolio of tools—PAPI, TAU, HPCToolkit, Caliper, Score-P, CUBE, Vampir, and Scalasca—is not a redundant collection of similar products. It is a carefully curated and interoperable set of components, each with a distinct philosophy and a specialized role in the performance analysis workflow.

A developer's "user experience" is not about choosing a single, monolithic tool, but about learning the *workflow* and picking the right tool for the right *question*. This analysis of the E4S 25.06 stack reveals a clear mapping from common performance questions to specific tools:

* **The Foundational Question:** "How do I access raw hardware counters?"
  + **The E4S Answer:** **PAPI**, which provides the universal, vendor-agnostic middleware that enables the entire ecosystem.20
* **The First Question:** "Where is my C/C++/Fortran code spending its time?"
  + **The E4S Answer:** **HPCToolkit**. Its "run-as-is," low-overhead sampling 6 and unparalleled heterogeneous attribution 3 make it the clear "first-pass" tool for hotspot analysis.
* **The "Why" Question:** "My profile shows a hotspot, but *why* is it behaving that way dynamically?"
  + **The E4C Answer:** The **Score-P + Vampir** workflow. Score-P provides the filtered, scalable trace 82, and Vampir provides the interactive "microscope" to visually debug timeline-dependent issues.7
* **The "Waste" Question:** "Where is my *parallel* time being wasted?"
  + **The E4S Answer:** The **Score-P + Scalasca** workflow. Scalasca's automated, parallel trace analysis is specifically designed to find and quantify time wasted in MPI and OpenMP synchronization and wait states.8
* **The "Hybrid" Question:** "How do I profile my hybrid Python/C++ or Kokkos-based application?"
  + **The E4S Answer:** **TAU**. Its "Swiss army knife" design, broad language support (especially Python) 43, and deep integration with portability models like Kokkos 41 make it the ideal choice for these complex, multi-layer applications.
* **The "Introspection" Question:** "How do I build profiling *into* my application and correlate it with my *semantic* parameters?"
  + **The E4S Answer:** **Caliper**. Its library-based, "always-on" annotation API 72 and unique key:value metadata model 71 are built for this exact purpose.

The E4S stack, through this curated and interoperable "separation of concerns" philosophy, successfully fulfills its mission: it lowers the barrier 1 for scientific developers, providing a clear and powerful set of tools to diagnose and solve performance problems on the world's most complex heterogeneous supercomputers.

#### Works cited

1. E4S - Home, accessed November 6, 2025, <https://e4s.io/>
2. THAPI: Tracing Heterogeneous APIs - arXiv, accessed November 6, 2025, <https://arxiv.org/html/2504.03683v1>
3. HPCToolkit - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/research-project/hpctoolkit/>
4. Performance Analysis of GPU-accelerated Applications with ..., accessed November 6, 2025, <https://www.alcf.anl.gov/sites/default/files/2024-11/HPCToolkit-ALCF-2024-10.pdf>
5. EXA-PAPI++ - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/research-project/exa-papi/>
6. HPCToolkit Home, accessed November 6, 2025, <https://hpctoolkit.org/>
7. Performance Analysis with Vampir, accessed November 6, 2025, <https://www.alcf.anl.gov/files/Vampir.pdf>
8. Scalasca, accessed November 6, 2025, <https://www.scalasca.org/>
9. E4S – High Performance Software Foundation (HPSF), accessed November 6, 2025, <https://hpsf.io/projects/e4s/>
10. E4S: The Extreme-scale Scientific Software Stack Release 25.06, accessed November 6, 2025, <https://oaciss.uoregon.edu/e4s/talks/E4S_25.06.pdf>
11. E4S: Extreme-Scale Scientific Software Stack, accessed November 6, 2025, <https://bssw.io/items/e4s-extreme-scale-scientific-software-stack>
12. Introduction — E4S 21.08 documentation, accessed November 6, 2025, <https://e4s.readthedocs.io/en/latest/introduction.html>
13. New Extreme-Scale Scientific Software Stack (E4S) Release Delivers Expanded Features and Capabilities: E4S Version 24.02 Now Available - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/new-extreme-scale-scientific-software-stack-e4s-release-delivers-expanded-features-and-capabilities-e4s-version-24-02-now-available/>
14. Releases · E4S-Project/e4s - GitHub, accessed November 6, 2025, <https://github.com/E4S-Project/e4s/releases>
15. E4S-Project/e4s: E4S for Spack - GitHub, accessed November 6, 2025, <https://github.com/E4S-Project/e4s>
16. Projects :: Score-P - VI-HPS, accessed November 6, 2025, <https://www.vi-hps.org/projects/score-p/>
17. Tools :: Score-P - VI-HPS, accessed November 6, 2025, <https://www.vi-hps.org/Tools/Score-P.html>
18. Vampir - | HPC @ LLNL - Lawrence Livermore National Laboratory, accessed November 6, 2025, <https://hpc.llnl.gov/software/development-environment-software/vampir>
19. Scalasca 2.x download, accessed November 6, 2025, <https://www.scalasca.org/scalasca/software/software.html>
20. Advancements of PAPI for the exascale generation - NetLib.org, accessed November 6, 2025, <https://www.netlib.org/utk/people/JackDongarra/PAPERS/adv-papi-ijhpca-2025.pdf>
21. PAPI CUDA Component | NVIDIA Developer, accessed November 6, 2025, <https://developer.nvidia.com/papi-cuda-component>
22. Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir - Parallel Programming, accessed November 6, 2025, <https://www.parallel.informatik.tu-darmstadt.de/zotero-lists/attachments/2012-Score-P_____A_Joint_Performance_Measurement_Run-Time_Infrastructure_for_Periscope__Scalasca__TAU__and_Vampir-XI6TBFJK.pdf>
23. Performance Analysis at Scale: The Score-P Tools Infrastructure, accessed November 6, 2025, <https://www.olcf.ornl.gov/wp-content/uploads/2018/02/SummitDev_Score-P_Tutorial.pdf>
24. Scalasca Trace Tools, accessed November 6, 2025, <https://www.scalasca.org/scalasca/upload/images/content/ScalascaFlyer.pdf>
25. 2.12. Using Score-P with TAU, accessed November 6, 2025, <https://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch02s12.html>
26. Vampir - OLCF User Documentation, accessed November 6, 2025, <https://docs.olcf.ornl.gov/software/profiling/Vampir.html>
27. PAPI, accessed November 6, 2025, <https://icl.utk.edu/papi/>
28. Performance Application Programming Interface - Exascale Computing Project, accessed November 6, 2025, <https://www.exascaleproject.org/wp-content/uploads/2021/01/PAPI_BOF_Presentation-pdf.pdf>
29. PAPI USER'S GUIDE - Innovative Computing Laboratory, accessed November 6, 2025, <https://icl.utk.edu/projects/papi/files/documentation/PAPI_USER_GUIDE.pdf>
30. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters - Department of Computer Science, accessed November 6, 2025, <https://www.cs.tufts.edu/comp/150PAT/tools/papi/pap.pap256.pdf>
31. intro\_papi — HPE Cray Programming Environment final documentation, accessed November 6, 2025, <https://cpe.ext.hpe.com/docs/24.11/performance-tools/man3/intro_papi.html>
32. PAPI USER'S GUIDE - Innovative Computing Laboratory, accessed November 6, 2025, <https://icl.utk.edu/projects/papi/files/documentation/PAPI_USER_GUIDE_23.htm>
33. Roadmap for Refactoring classic PAPI to PAPI++ : Part II - DTIC, accessed November 6, 2025, <https://apps.dtic.mil/sti/trecms/pdf/AD1124430.pdf>
34. PAPI: Performance Application Programming Interface - | HPC @ LLNL, accessed November 6, 2025, <https://hpc.llnl.gov/software/development-environment-software/papi-performance-application-programming-interface>
35. Measuring Energy and Power with PAPI, accessed November 6, 2025, <https://web.eece.maine.edu/~vweaver/papers/pasa2012/pasa2012_power.pdf>
36. icl-utk-edu/papi - GitHub, accessed November 6, 2025, <https://github.com/icl-utk-edu/papi>
37. Papi - Consortium for the Advancement of Scientific Software, accessed November 6, 2025, <https://cass.community/software/papi.html>
38. Introduction to Performance Analysis Concepts, accessed November 6, 2025, <https://www.olcf.ornl.gov/wp-content/uploads/2019/08/2_profiling_introduction.pdf>
39. GPU performance counters - CUDA - NVIDIA Developer Forums, accessed November 6, 2025, <https://forums.developer.nvidia.com/t/gpu-performance-counters/28868>
40. Using PAPI for hardware performance monitoring on Linux systems - NetLib.org, accessed November 6, 2025, <https://www.netlib.org/utk/people/JackDongarra/PAPERS/papi-linux.pdf>
41. ECP Provides TAU, a CPU/GPU/MPI Profiler, for All HPC and ..., accessed November 6, 2025, <https://www.exascaleproject.org/highlight/ecp-provides-tau-a-cpu-gpu-mpi-profiler-for-all-hpc-and-exascale-machines/>
42. Interactive Videos - The TAU Performance System and E4S - Education, accessed November 6, 2025, [https://education.sdsc.edu/training/interactive/index.html?id=202503-The%20TAU%20Performance%20System%20and%20E4S](https://education.sdsc.edu/training/interactive/index.html?id=202503-The+TAU+Performance+System+and+E4S)
43. TAU - Tuning and Analysis Utilities -, accessed November 6, 2025, <https://www.cs.uoregon.edu/research/tau/home.php>
44. Tuning and Analysis Utilities (TAU) Seminar - CERN Indico, accessed November 6, 2025, <https://indico.cern.ch/event/845622/contributions/3551325/attachments/1925879/3187773/CERN19.pdf>
45. How to use TAU for Performance Analysis, accessed November 6, 2025, <https://www.olcf.ornl.gov/wp-content/uploads/2019/08/3_tau_day_1.pdf>
46. TAU: Tuning and Analysis Utilities - | HPC @ LLNL, accessed November 6, 2025, <https://hpc.llnl.gov/software/development-environment-software/tau-tuning-and-analysis-utilities>
47. Tuning and Analysis Utilities (TAU) - OLCF User Documentation, accessed November 6, 2025, <https://docs.olcf.ornl.gov/software/profiling/TAU.html>
48. TAU Performance System® - Stony Brook University, accessed November 6, 2025, <https://www.stonybrook.edu/commcms/ookami/support/_docs/TAU_SBU21-Slides.pdf>
49. Performance Evaluation using TAU - Argonne Leadership Computing Facility, accessed November 6, 2025, <https://www.alcf.anl.gov/files/taucq12-05.pdf>
50. TAU - Tuning and Analysis Utilities - - Computer Science - University of Oregon, accessed November 6, 2025, <https://www.cs.uoregon.edu/research/tau/docs.php>
51. Performance monitoring with TAU and E4S on LUMI and Cloud HPC platforms - ENCCS, accessed November 6, 2025, <https://enccs.se/events/performance-monitoring-with-tau-e4s/>
52. The TAU Performance System - ParaTools, Inc., accessed November 6, 2025, <https://www.paratools.com/tau>
53. TAU - Tuning and Analysis Utilities - - Computer Science - University of Oregon, accessed November 6, 2025, <https://www.cs.uoregon.edu/research/tau/news.php>
54. Performance Monitoring with TAU and E4S - HPC in Europe Portal, accessed November 6, 2025, <https://hpc-portal.eu/news/events/performance-monitoring-tau-and-e4s>
55. TAU Performance System | NVIDIA Developer, accessed November 6, 2025, <https://developer.nvidia.com/tau-performance-system>
56. Multi-platform SYCL profiling with TAU - YouTube, accessed November 6, 2025, <https://www.youtube.com/watch?v=tSSfKNrJ660>
57. ParaTools, Inc. – Performance Engineering, accessed November 6, 2025, <https://www.paratools.com/>
58. Products – ParaTools, Inc., accessed November 6, 2025, <https://www.paratools.com/products/>
59. ParaTools Pro for E4S™: AI/ML & HPC Tools on AWS PCS (x86), accessed November 6, 2025, <https://aws.amazon.com/marketplace/pp/prodview-wryfn3vd5c63k>
60. ParaTools Pro for E4S, accessed November 6, 2025, <https://www.paratools.com/paratoolspro/>
61. accessed November 6, 2025, <https://www.paratools.com/tau#:~:text=ParaTools%20is%20the%20sole%20licensee,related%20to%20the%20TAU%20project.>
62. Analyzing Parallel Program Performance using HPCToolkit, accessed November 6, 2025, <https://www.alcf.anl.gov/files/slides%202018-02%20ALCF%20Dev%20Session%20John%20Mellor-Crummey%20HPCToolkit.pdf>
63. Measurement and Analysis of Parallel Program Performance Using TAU and HPCToolkit, accessed November 6, 2025, <https://www.researchgate.net/publication/291081231_Measurement_and_Analysis_of_Parallel_Program_Performance_Using_TAU_and_HPCToolkit>
64. Hpctoolkit - Consortium for the Advancement of Scientific Software, accessed November 6, 2025, <https://cass.community/software/hpctoolkit.html>
65. Identifying Scalability Bottlenecks with HPCToolkit, accessed November 6, 2025, <https://hpctoolkit.org/pubs/2021-11-SC21-Tutorial-HPCToolkit-Scalability.pdf>
66. Publications - HPCToolkit, accessed November 6, 2025, <https://hpctoolkit.org/publications.html>
67. Analyzing GPU-accelerated Applications with HPCToolkit, accessed November 6, 2025, <https://hpctoolkit.org/pubs/2021-11-SC21-Tutorial-HPCToolkit-GPU.pdf>
68. Performance Analysis of GPU-accelerated Applications with HPCToolkit, accessed November 6, 2025, <https://www.alcf.anl.gov/sites/default/files/2025-05/HPCToolkit-ANL-GPU-Hackathon-2025-05.pdf>
69. Performance Analysis of GPU-accelerated OpenMP Applications using HPCToolkit, accessed November 6, 2025, <https://www.openmp.org/wp-content/uploads/HPCToolkit-OpenMP-2023-08.pdf>
70. LLNL/Caliper: Caliper is an instrumentation and performance profiling library - GitHub, accessed November 6, 2025, <https://github.com/LLNL/Caliper>
71. A Performance Analysis Toolbox in a Library — Caliper 2.12.1 documentation, accessed November 6, 2025, <https://software.llnl.gov/Caliper/>
72. Tool Time: Caliper - A Performance Analysis Toolbox in a Library - POP-CoE.eu, accessed November 6, 2025, <https://pop-coe.eu/blog/tool-time-caliper-a-performance-analysis-toolbox-in-a-library>
73. Caliper | Computing - Lawrence Livermore National Laboratory, accessed November 6, 2025, <https://computing.llnl.gov/projects/caliper>
74. Caliper Basics — Caliper 2.12.1 documentation, accessed November 6, 2025, <https://software.llnl.gov/Caliper/CaliperBasics.html>
75. Caliper vi-hps tutorial - | Computing - Lawrence Livermore National Laboratory, accessed November 6, 2025, <https://computing.llnl.gov/sites/default/files/Caliper%20vi-hps%20tutorial.pdf>
76. Performance Introspection for HPC Software Stacks - Caliper - OSTI, accessed November 6, 2025, <https://www.osti.gov/servlets/purl/1864938>
77. Tutorials 2024: Caliper, Hatchet, and Thicket - YouTube, accessed November 6, 2025, <https://www.youtube.com/watch?v=qVmxDOxM9Ws>
78. GPU profiling — Caliper 2.12.1 documentation, accessed November 6, 2025, <https://software.llnl.gov/Caliper/GPUProfiling.html>
79. Score-P – A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir - VI-HPS, accessed November 6, 2025, <https://www.vi-hps.org/cms/upload/material/tw-score-p/vi-hps-tw-scorep-08_Score_P.pdf>
80. Projects :: Score-P - VI-HPS, accessed November 6, 2025, <https://www.score-p.org/>
81. Score-P - HPC Wiki, accessed November 6, 2025, <https://hpc-wiki.info/hpc/Score-P>
82. Score-P — OLCF User Documentation, accessed November 6, 2025, <https://docs.olcf.ornl.gov/software/profiling/Scorep.html>
83. Performance Analysis at Scale using Score-P and Vampir, accessed November 6, 2025, <https://www.alcf.anl.gov/files/ScoreP_and_Vampir.pdf>
84. Score-P: Introduction - Forschungszentrum Jülich, accessed November 6, 2025, <https://scorepci.pages.jsc.fz-juelich.de/scorep-pipelines/docs/scorep-4.1/html/index.html>
85. Record Course of Events with Score-P - ZIH HPC Compendium - TU Dresden, accessed November 6, 2025, <https://compendium.hpc.tu-dresden.de/software/scorep/>
86. Performance Analysis Workflow Using Score-P, accessed November 6, 2025, <https://scorepci.pages.jsc.fz-juelich.de/scorep-pipelines/docs/scorep-6.0/html/workflow.html>
87. Application Profiling with Score-P and Cube - YouTube, accessed November 6, 2025, <https://www.youtube.com/watch?v=3HtR89L2u9Q>
88. Monitoring with Score-P, accessed November 6, 2025, <https://www.fz-juelich.de/en/jsc/news/events/exhibitions/jsc-media-files/scorep/@@download/file/Monitoring%20with%20Score-P-PDF.pdf>
89. Hands-On Score-P/Scalasca/CUBE - VI-HPS, accessed November 6, 2025, <https://www.vi-hps.org/cms/upload/material/tw45/Hands-On_Score-P-Scalasca-CUBE.pdf>
90. Vampir - Visualization and Analysis of Parallel Applications — Center for Information Services and High Performance Computing (ZIH) - TU Dresden, accessed November 6, 2025, <https://tu-dresden.de/zih/forschung/projekte/vampir/index?set_language=en>
91. Vampir - ParaTools, Inc., accessed November 6, 2025, <https://www.paratools.com/vampir/>
92. Tool Time: Performance Analysis Study with Vampir - POP-CoE.eu, accessed November 6, 2025, <https://pop-coe.eu/blog/tool-time-performance-analysis-study-with-vampir>
93. Trace Analysis with Vampir - VI-HPS, accessed November 6, 2025, <https://www.vi-hps.org/cms/upload/material/tw39/trace-analysis-with-vampir.pdf>
94. Scalasca User's Guide - Forschungszentrum Jülich, accessed November 6, 2025, <http://apps.fz-juelich.de/scalasca/releases/scalasca/1.4/docs/UserGuide.pdf>
95. Scalasca - Docs CSC, accessed November 6, 2025, <https://docs.csc.fi/apps/scalasca/>
96. 21st POP Webinar - The Scalasca Scalable Parallel Performance Analysis Toolset - For POP Assessments and Beyond | Performance Optimisation and Productivity, accessed November 6, 2025, <https://pop-coe.eu/blog/21st-pop-webinar-the-scalasca-scalable-parallel-performance-analysis-toolset-for-pop>
97. Performance Analysis with Scalasca, accessed November 6, 2025, <https://www.olcf.ornl.gov/wp-content/uploads/2019/08/5_scalasca_day_1.pdf>
98. Scalasca, accessed November 6, 2025, <https://icl.utk.edu/scalasca/>
99. Automatic trace analysis with the Scalasca Trace Tools - ARCHER2, accessed November 6, 2025, <https://www.archer2.ac.uk/training/courses/220509-scalasca/Scalasca.pdf>
100. Vampir 10.7, accessed November 6, 2025, <https://vampir.eu/>