EXA-PAPI++

Understanding the performance characteristics of exascale applications is necessary in order to identify and address the barriers to achieving performance goals. This becomes more difficult as the architectures become more complex. The Performance Application Programming Interface (PAPI) provides both library and application developers with generic and portable access to low-level performance counters found across the exascale machine, enabling users to see the relationships between software performance and hardware events. These relationships provide a critical step toward improving performance.

The Exascale Performance Application Programming Interface (Exa-PAPI++) project is developing a new C++ Performance API (PAPI++) software package from the ground up that offers a standard interface and methodology for using low-level performance counters in CPUs, GPUs, on/off-chip memory, interconnects, and the I/O system, including energy/power management. PAPI++ is building upon classic-PAPI functionality and strengthening its path to exascale with a more efficient and flexible software design, one that takes advantage of C++'s object-oriented nature but preserves the low-overhead monitoring of performance counters and adds a vast testing suite.

In addition to providing hardware counter-based information, a standardizing layer for monitoring software-defined events (SDE) is being incorporated that exposes the internal behavior of runtime systems and libraries, such as communication and math libraries, to the applications. As a result, the notion of performance events is broadened from strictly hardware-related events to include software-based information.

Enabling monitoring of both hardware and software events provides more flexibility to developers when capturing performance information.

In summary, the Exa-PAPI++ team is preparing PAPI support to stand up to the challenges posed by exascale systems by (1) widening its applicability and providing robust support for exascale hardware resources; (2) supporting finer-grain measurement and control of power, thus offering software developers a basic building block for dynamic application optimization under power constraints; (3) extending PAPI to support software-defined events; and (4) applying semantic analysis to hardware counters so that the application developer can better make sense of the ever-growing list of raw hardware performance events that can be measured during execution. The team will be channeling the monitoring capabilities of hardware counters, power usage, software-defined events into a robust PAPI++ software package. PAPI++ is meant to be PAPI's replacement—with a more flexible and sustainable software design.

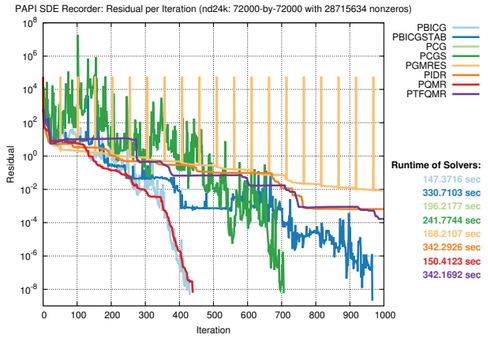

Figure (top) PAPI SDE-Recorder logging convergence of different ILU-preconditioned MAGMA-sparse Krylov solvers for a 2D/3D Problem; **(bottom)** PAPI SDE-Recorder logging the status of different task queues in PaRSEC.

PI: Jack Dongarra, University of Tennessee – Knoxville

Co-PIs: Heike Jagode and Anthony Danalis, University of Tennessee – Knoxville

Collaborators: University of Tennessee – Knoxville

Progress to date

- On the **software event** front, the team began with the design and implementation of a new API to expose any kind of software-defined events. It extends PAPI's role so that it becomes the de facto standard for exposing performance-critical events from different software layers.
- Because the concept of software-defined events is new to PAPI, the team worked closely with developers of different libraries and runtimes that serve as natural targets for early adoption of the new SDE API. To date, the team has integrated SDEs into the sparse linear algebra library MAGMA-Sparse, the tensor algebra library TAMM (NWChemEx), the task-scheduling runtime PaRSEC, and the compiler-based performance analysis tool BYFL.
- The plot on the top in the figure illustrates how the convergence of Krylov solvers can be visualized with the help of PAPI SDEs. Each of these solvers behave very differently for different problems and matrices, which, once more, stresses the importance of exposing these details in a standardized way. This allows the domain scientist to quickly identify the fastest and most robust method of choice for their very unique problems. Most importantly, this information can now be obtained without expert knowledge about algorithm-specific characteristics, and without having to instrument MAGMA library code, simply by calling PAPI_read() in the top-level application.
- The plot on the bottom of the figure serves as a second showcase, illustrating the evolution of various task queues during the execution of a Cholesky factorization in PaRSEC. With SDEs in PARSEC, a user can get a view of what is happening inside the runtime by simply calling PAPI_start() and PAPI_stop() in their application, without the need to instrument the PaRSEC runtime code.
- On the **hardware counter** front, the team has developed a new PAPI component called "PCP" for IBM POWER9 hardware counters. It adds support for (1) core performance events, which are specific to each core, and (2) shared events, which monitor the performance of node-wide resources that are shared between cores. Access to shared events requires elevated privileges. However, IBM's official route for providing access to shared events is through the Performance Co-Pilot (PCP) for non-root users. The new PAPI-PCP component enables all users to access POWER 9 shared events through PAPI.