The Performance API (PAPI) provides tool designers and application engineers with a consistent interface and methodology for the use of low-level performance counter hardware found across the entire system (i.e., CPUs, GPUs, on/off-chip memory, interconnects, I/O system, energy/power, etc.). PAPI enables users to see, in near real time, the relationship between software performance and hardware events across the entire system.

**PULSE SCOPE**

PULSE builds on the latest PAPI project and extends it with software-defined events (SDE) that originate from the HPC software stack and are currently treated as black boxes (i.e., communication libraries, math libraries, task-based runtime systems, applications).

The objective is to enable monitoring of both types of performance events—hardware- and software-related events—in a uniform way, through one consistent PAPI interface. Therefore, third-party tools and application developers have to handle only a single hook to PAPI to access all hardware performance counters in a system, including the new software-defined events.

### Performance overhead studies

**Synthetic Benchmarks**

![Synthetic Benchmarks](image)

**Real applications and runtimes**

![Real applications and runtimes](image)

### Software-Defined Events in PAPI

**GOAL**

Offer support for software-defined events (SDE) to extend PAPI’s role as a standardizing layer for performance counter monitoring.

**VISION**

Enable NSF software layers to expose SDEs that performance analysts can use to form a complete picture of the entire application performance.

**BENEFIT**

Scientists will be better able to understand the interaction of the different applications layers, and interactions with external libraries and runtimes.

### PAPI’s Basic SDE API

- API for reading SDEs remains the same as the API for reading hardware events, i.e., `PAPI_start()`, etc.
- SDE API calls are only meant to be used inside libraries to export SDEs from within those libraries.
- All API functions are available in C and FORTRAN.

```c
void papi_sde_register_counter(papi_handle_t handle, const char *event_name, int mode, int type, void *counter);
```

Initializes internal data structures and returns an opaque handle that must be passed to all subsequent calls to PAPI SDE functions.

```c
void papi_sde_register_fp_counter(papi_handle_t handle, const char *event_name, void *func_ptr_t, void *param);
```

Registers a function pointer to an accessor function provided by the library. Allows the user to export an event whose value does not map to the value of a single program variable/metric of the library.

```c
int papi_sde_init(const char *lib_name);
```

Integrates integration of PAPI SDE in PaRSEC

![Integration of PAPI SDE in PaRSEC](image)

**Porlofing Example: Integration of PAPI SDE & 3rd party tools**

![Porlofing Example](image)

**CASE STUDY: Integration of PAPI SDE in PaRSEC**

- As our application case study, we chose the task scheduling runtime PaRSEC.
- We created several Software Defined Events, some to expose the internal state of the runtime (such as the length of the task queues) and some to expose events that occur during scheduling and can affect performance (such as task stealing between different cores, or work starvation)

![Porlofing Example](image)