INNOVATIVE
COMPUTING LABORATORY

SI2-SSE: PULSE
PAPI Unifying Layer for Software-Defined Events

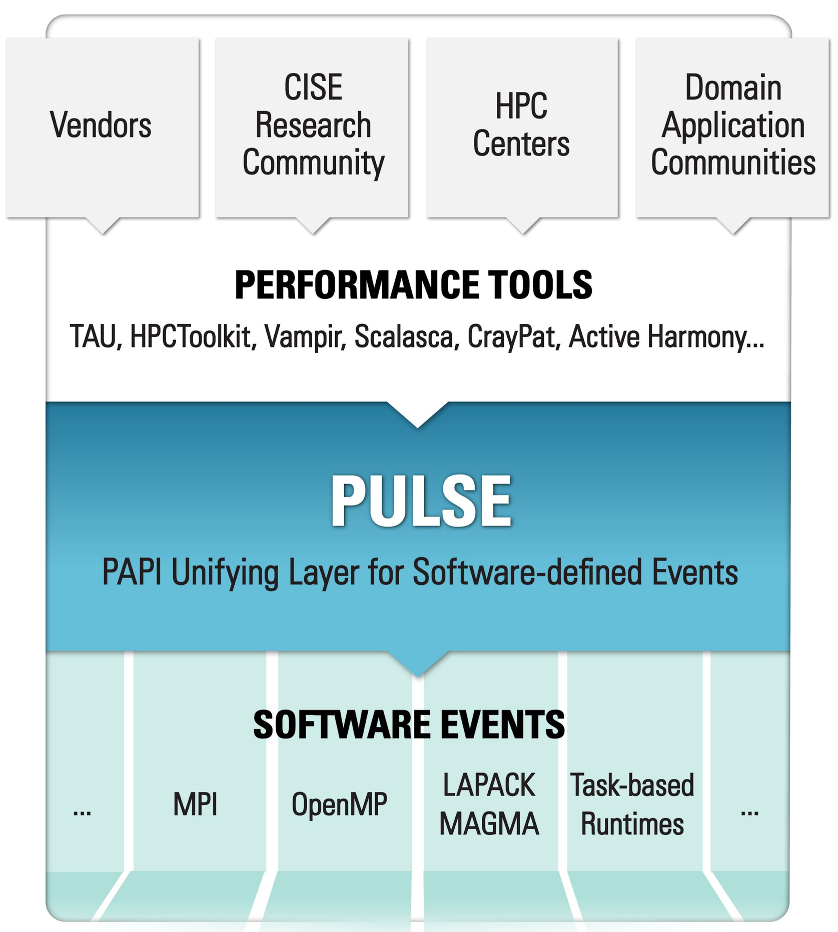
Heike Jagode
Anthony Danalis
UNIVERSITY OF TENNESSEE

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

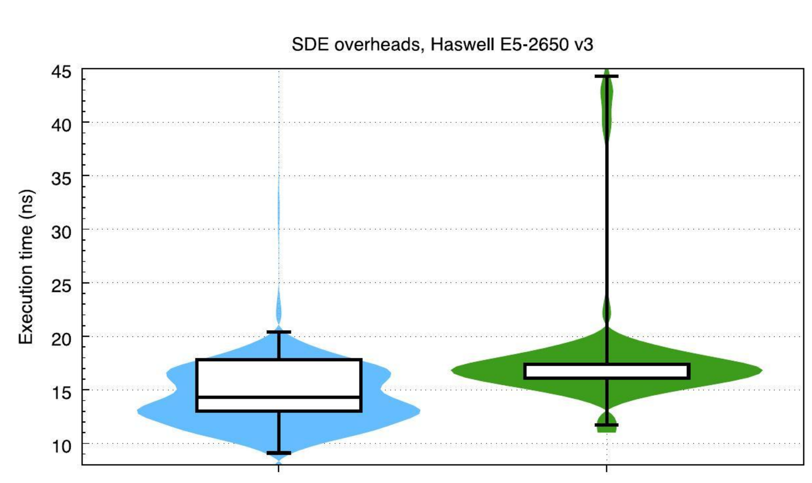
PULSE builds on the latest PAPI project and extends it with **software-defined events (SDE)** that originate from the HPC software stack and are currently treated as black boxes (i.e., com -munication libraries, math libraries, task-based runtime systems, applications).

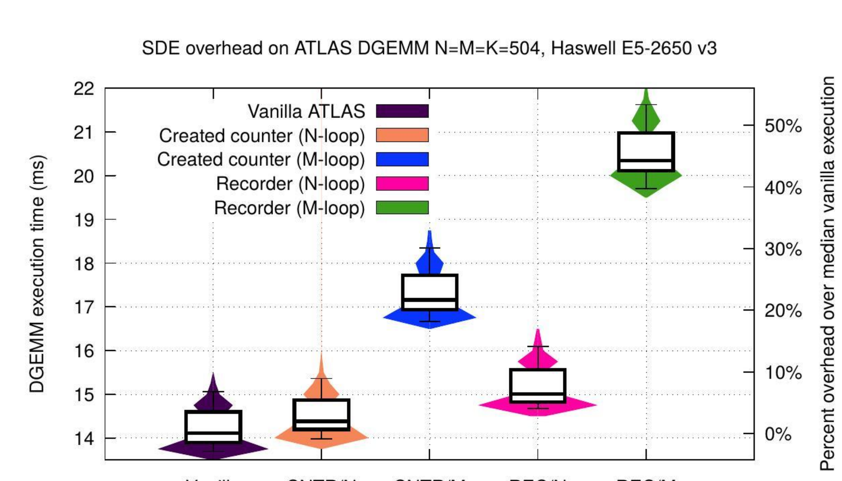
The objective is to enable **monitoring of both types of performance events**—hardware- and software-related events—**in a uniform way**, through one consistent PAPI interface. Therefore, third-party tools and application developers have to handle only **a single hook to PAPI** to access all hardware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



Performance overhead studies

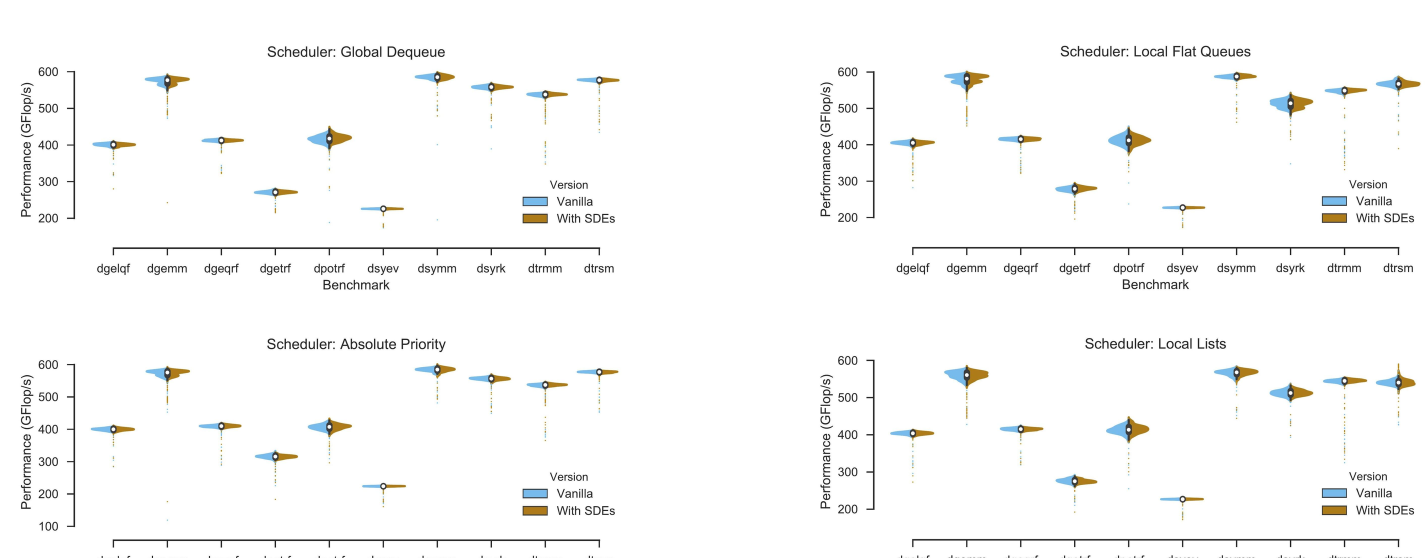
Synthetic Benchmarks



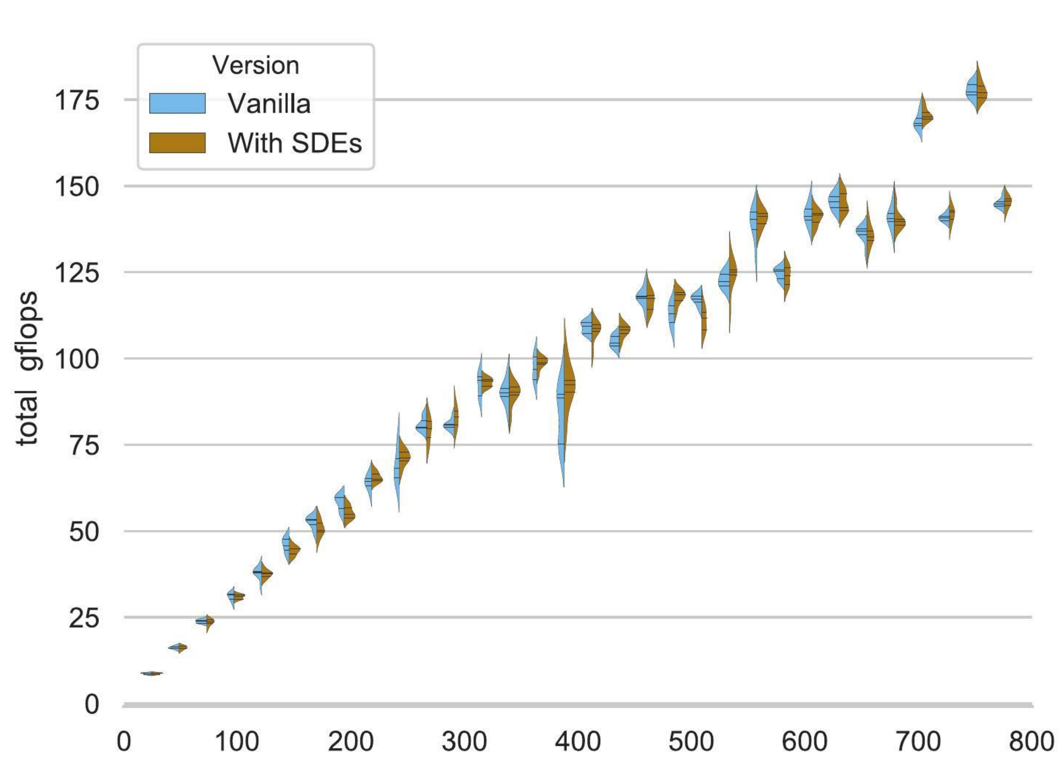


Real applications and runtimes

PaRSEC



HPCG



Software-Defined Events in PAPI

GOAL	VISION	BENEFIT
Offer support for software-defined events (SDE) to extend PAPI's role as a standardizing layer for perfor -mance counter monitoring.	**Enable NSF software layers to expose SDEs that performance analysts can use to form a complete picture of the entire application performance.**	**Scientists will be better able to under -stand 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.

```
papi_handle_t papi_sde_init(const char *lib_name);
```

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

`lib_name` is a string containing the name of the library.

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

Must be called for every program variable/metric that the library wishes to register as an event.

`handle` is the opaque handle returned by `papi_sde_init()`.
`event_name` is a string containing the name of the event being registered.
`mode` is an integer declaring whether a counter is read-only or read-write.
`type` is an enumeration of the type of the event.
`counter` is a pointer to the actual variable that serves as the counter for this event.

```
typedef long long int (*func_ptr_t)(void *);
void papi_sde_register_fp_counter(void *handle, const char *event_name,
                int mode, int type, func_ptr_t fp_counter, 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.

`fp_counter` is a pointer to the accessor function.
`param` is an opaque object that the library passes to PAPI, and PAPI passes it as a parameter to the accessor function.

```
void papi_sde_describe_counter(papi_handle_t handle, const char *event_name
        const char *event_description);
```

Associates a longer description with an event. This description will be shown by the utility papi_native_avail so that users can be informed about an event's meaning.

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)

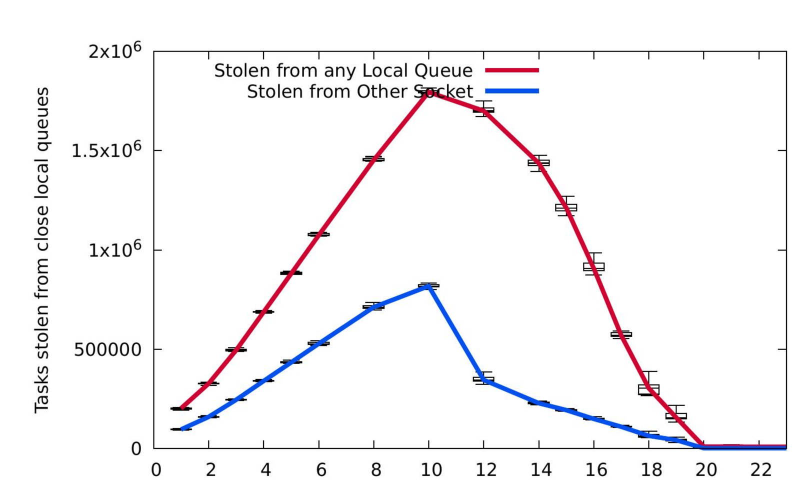


Figure 1 shows the number of tasks that were stolen between threads residing in different cores (red curve), or different sockets (blue curve) for different queue sizes. Studying such curves reveals the quality of design decisions inside the runtime.

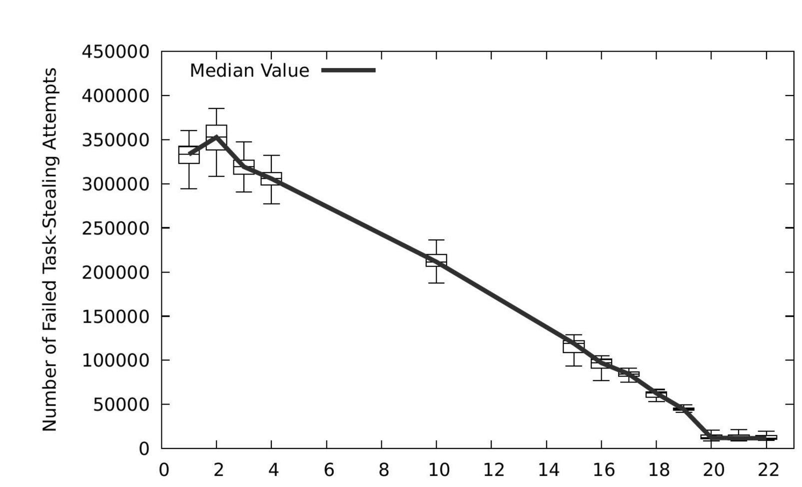


Figure 2 shows the number of times a thread tried to steal work and failed to do so. This value reveals starvation problems, which lead to performance degradation.

Overflowing Example: Integration of PAPI SDE & 3rd party tools

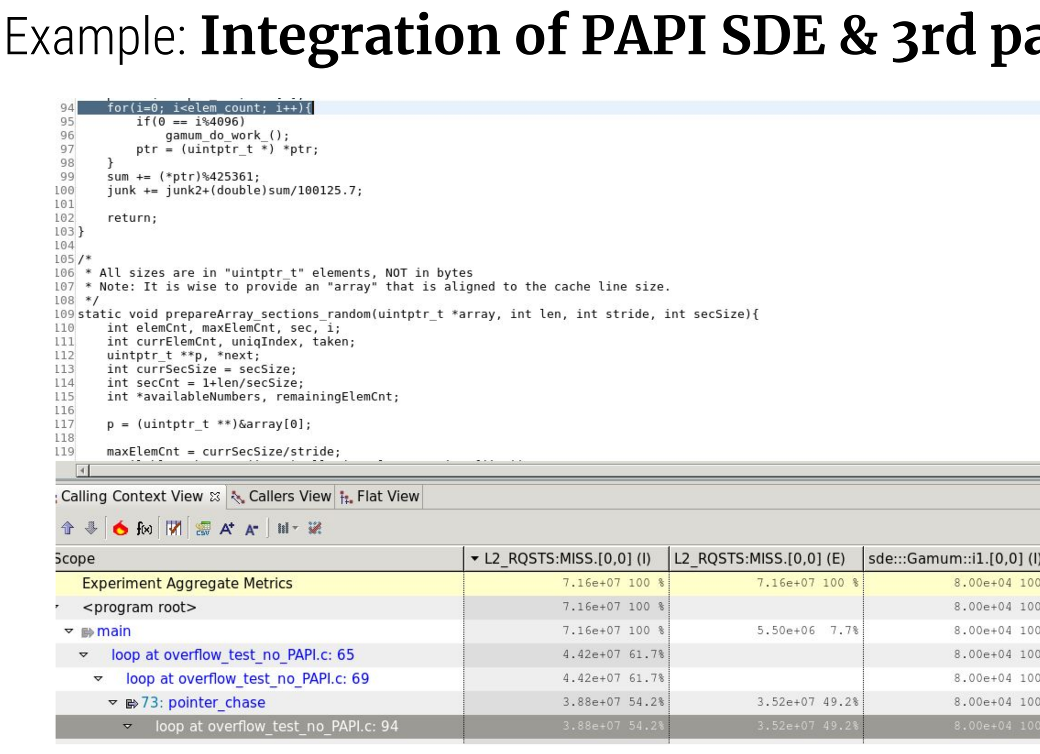


Figure 3 shows an unmodified version of the 3rd party tool HPCToolkit being able to read a PAPI Software Defined Event, and show where in the test code it occurs.

ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY OF TENNESSEE KNOXVILLE
NSF