

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.

Software-Defined Events in PAPI

GOAL

Offer support for software-defined events (SDE) to extend PAPI's role as a standardizing layer for perfor -mance 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 under -stand the interaction of the different applications layers, and interactions with external libraries and runtimes.

PULSE SCOPE

PULSE builds on the latest PAPI project and



PAPI's Basic SDE API

• API for reading SDEs remains the same as the API for reading hardware events, i.e., PAPI_start(), etc.

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).

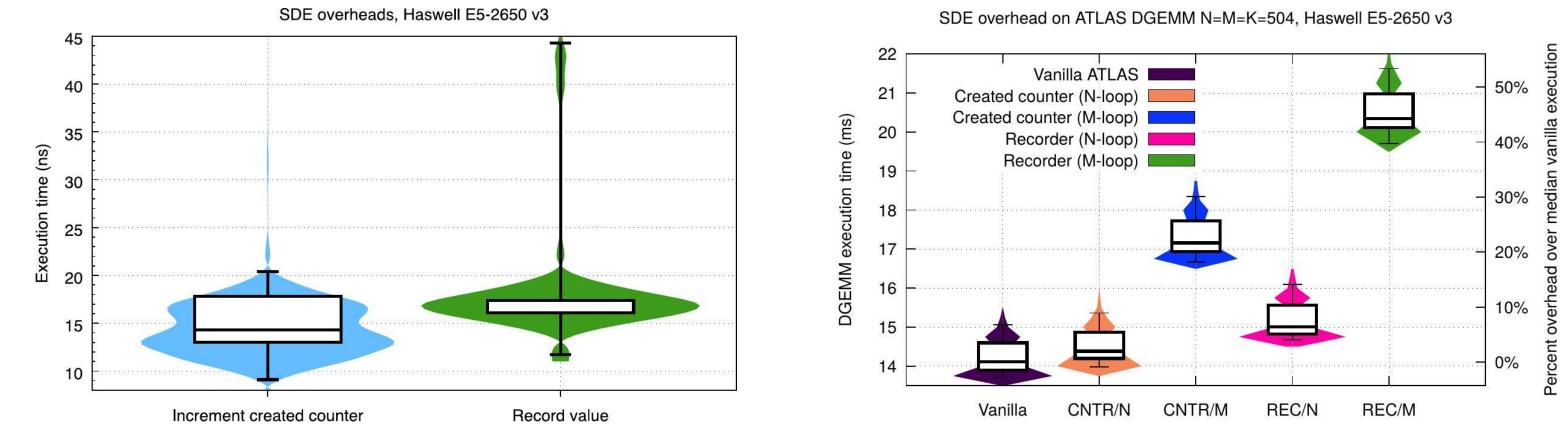
The objective is to enable monitoring of both types of performance events---hardware- and software-related events---in a uniform way, through one consistent PAPI interface. Therefore, third-party tools and application developers have to handle only a single hook to PAPI to access all hardware performance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

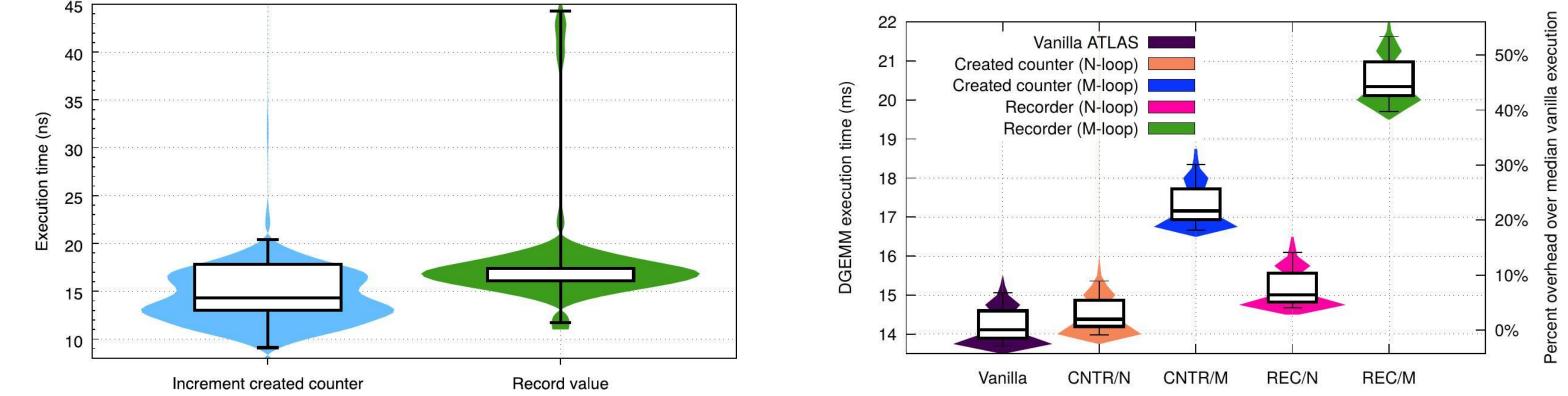
Application Vendors Research Centers Communities Community **PERFORMANCE TOOLS** TAU, HPCToolkit, Vampir, Scalasca, CrayPat, Active Harmony...

PAPI Unifying Layer for Software-defined Events **SOFTWARE EVENTS** LAPACK Task-based MPI **OpenMP** MAGMA Runtimes

PULSE

Performance overhead studies Synthetic Benchmarks





- 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.

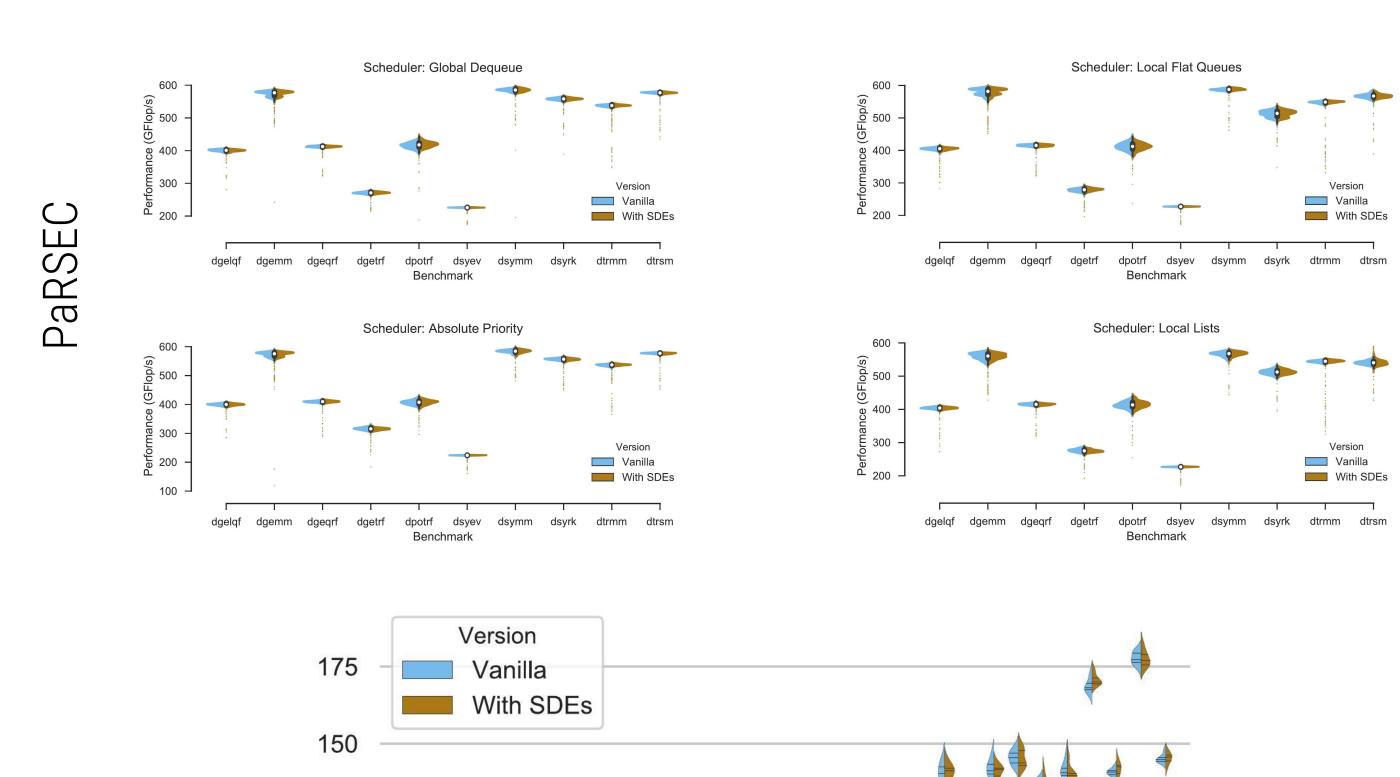
fp counter is a pointer to the accessor function.

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

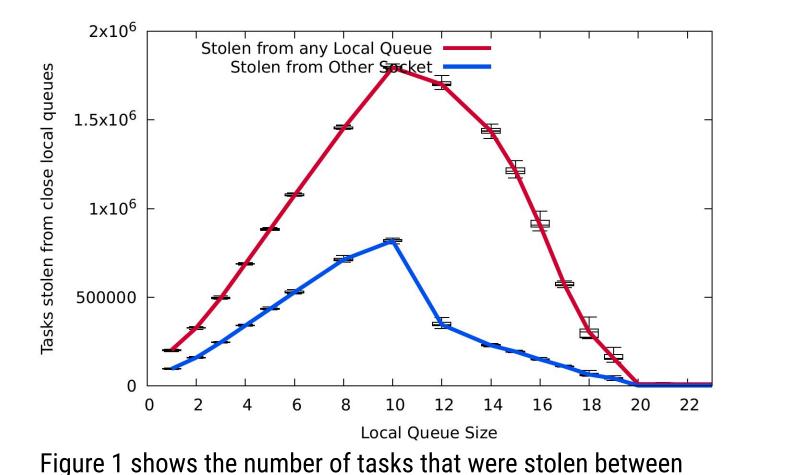
Real applications and runtimes



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)



threads residing in different cores (red curve), or different sockets

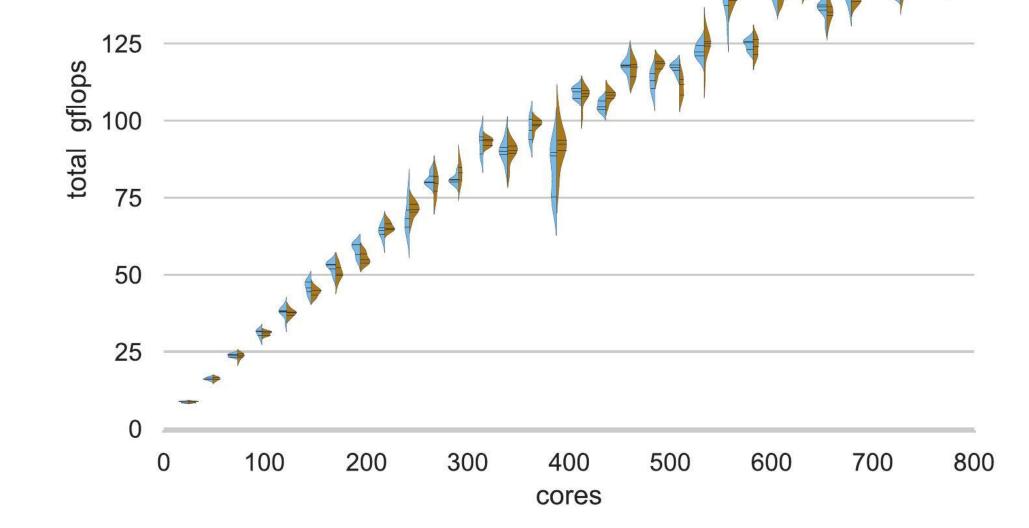
the quality of design decisions inside the runtime.

(blue curve) for different queue sizes. Studying such curves reveals

Median Value 400000 350000 300000 250000 200000 150000 100000 50000 12 14 16 18 20 22 10 Local Queue Size 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







94	<pre>for(i=0; i<elem count;="" i++){<="" pre=""></elem></pre>
94	if(0 = i 84096)
96	gamum_do_work_();
97	$gamdm_do_work_()$; ptr = (uintptr t *) *ptr;
98	$p(r) = (u)(p(r)_{r})^{2}p(r)$
99	sum += (*ptr)%425361;
100	junk += junk2+(double)sum/100125.7;
101	
102	return;
103 }	
104	
105 /*	
106 *	All sizes are in "uintptr_t" elements, NOT in bytes
107 *	Note: It is wise to provide an "array" that is aligned to the cache line size.
108 *,	
109 st	atic void prepareArray_sections_random(uintptr_t *array, int len, int stride, int secSize){
110	int elemCnt, maxElemCnt, sec, i;
111	int currElemCnt, uniqIndex, taken;
112	uintptr_t **p, *next;
113	int currSecSize = secSize;
114	int secCnt = 1+len/secSize;
115	int *availableNumbers, remainingElemCnt;
116	
117	p = (uintptr_t **)&array[0];
118	
119	<pre>maxElemCnt = currSecSize/stride;</pre>
4	
Callir	ng Context View 🕱 🗞 Callers View 抗 Flat View
A .	- 🍝 foo 🕅 📰 At 🗛 🖬 - 😹

Scope	L	▼ L2_RQSTS:MISS.[0,0] (I)	L2_RQSTS:MISS.[0,0] (E)	sde:::Gamum::i1.[0,0] (I)
E	xperiment Aggregate Metrics	7.16e+07 100 %	7.16e+07 100 %	8.00e+04 100 %
<program root=""></program>		7.16e+07 100 %		8.00e+04 100 %
▼ 🛤	main	7.16e+07 100 %	5.50e+06 7.7%	8.00e+04 100 %
~	loop at overflow_test_no_PAPI.c: 65	4.42e+07 61.7%		8.00e+04 100 %
	Ioop at overflow_test_no_PAPI.c: 69	4.42e+07 61.7%		8.00e+04 100 %
		3.88e+07 54.2%	3.52e+07 49.2%	8.00e+04 100 %
	 loop at overflow_test_no_PAPI.c: 94 	3.88e+07 54.2%	3.52e+07 49.2%	8.00e+04 100 %

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.