INNOVATIVE
COMPUTING LABORATORY

ECP EXA-PAPI

SOFTWARE AVAILABLE AT

http://icl.utk.edu/exa-papi/

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 relations between software performance and hardware events across the entire system.

ECP Scope

Exa-PAPI builds on the latest PAPI project and we will extend it with:

GOAL 1 Performance counter monitoring capabilities for new and advanced ECP hardware, and software technologies.

GOAL 2 Fine-grained power management support.

GOAL 3 Functionality for performance counter analysis at "task granularity" for task-based runtime systems.

GOAL 4 "Software-defined Events" that originate from the ECP software stack and are currently treated as black boxes (i.e., communication libraries, math libraries, task-based runtime systems, etc.)

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. That implies, 3rd-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 Counter Monitoring Capabilities

SUPPORTED ARCHITECTURES

AMD	arm Cortex A8, A9, A15, ARM64	CRAY THE SUPERCOMPUTER COMPANY Gemini and Aries interconnect, power	IBM Blue Gene Series, Q: 5-D Torus, I/O System, EMON power, energy	IBM Power Series
IBM Power9 NEST event support via Performance Co-Pilot (PCP) PAPI component	intel Westmore, Sandy/Ivy Bridge, Haswell, Broadwell, Skylake(-X), Kaby Lake	intel KNC, Knights Landing including power/energy	intel RAPL (power/energy), power capping	INFINIBAND TRADE ASSOCIATION
lustre	NVIDIA Tesla, Kepler: CUDA support for multiple GPUs; PC Sampling	NVIDIA NVML	KVM Virtual Environment	vmware Virtual Environment

ECP PROJECTS AND 3RD PARTY TOOLS APPLYING PAPI

ECP DTE (PaRSEC) UTK http://icl.utk.edu/parsec/	**ECP LLNL-ATDM (Caliper)** LLVM github.com/LLNL/caliper-compiler	**ECP SNL-ATDM (Kokkos)** SNL https://github.com/kokkos	**ECP Proteas (TAU)** University of Oregon http://tau.uoregon.edu/
ECP HPCToolkit (HPCToolkit) Rice University http://hpctoolkit.org	**Score-P** http://score-p.org	**Vampir** TU Dresden http://www.vampir.eu/	**Scalasca** FZ Juelich, TU Darmstadt http://scalasca.org/
PerfSuite NCSA http://perfsuite.ncsa.uiuc.edu/	**Open|Speedshop** Open|SpeedShop https://openspeedshop.org/	**SvPablo** RENCI at UNC www.renci.org/research/pablo	**ompP** LMU Munich http://www.ompp-tool.com/

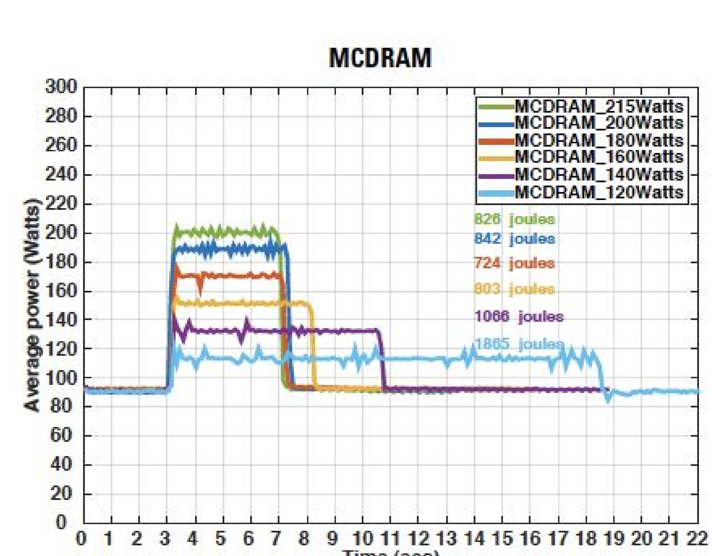
PAPI for power-aware computing

- PAPI's latest v. 5.6.0 ships with a **powercap component** for power/energy measurement, control.
- In the past, PAPI power components supported **only reading** power information.
- New component exposes RAPL functionality to allow users to **read and write** power.
- Study numerical building blocks of varying computational intensity.
- Use PAPI powercap component to detect power optimization opportunities.
- Cap the power on the architecture to reduce power usage while keeping the execution time constant → **energy savings**.

POWER AWARENESS EXAMPLE: JACOBI

68 cores KNL, Peak DP = 2,662 Gflop/s Bandwidth MCDRAM ~425 GB/s DDR4 ~90 GB/s

Solving Helmholtz equation with Jacobi Iterative Method (grid size 12,800 x 12,800, 2D 5-point stencil):
→ requires multiple memory accesses per update → high-memory BW, low computational intensity



LESSONS

- Computation is about 3.5X faster when the data is allocated in MCDRAM compared to DDR4.
- MCDRAM: Cap at 170 W improves energy efficiency by ~14% without any loss in time to solution.
- DDR4: Cap at 135 W improves energy efficiency by ~25% without any loss in time to solution.

REFERENCE

A. Haidar, H. Jagode, P. Vaccaro, A. YarKhan, S. Tomov, and J. Dongarra, **"Investigating Power Capping toward Energy-Efficient Scientific Applications"**, *Concurrency and Computation: Practice and Experience (CCPE): Special Issue on Power-Aware Computing 2017* (In Review).

Software-defined Events in PAPI

GOAL — Offer support for **software-defined events (SDE)** to extend PAPI's role as a standardizing layer for monitoring performance.

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

BENEFIT — ECP application scientists will be able to better understand the interaction of the different layers of their applications, as well as the interaction with external libraries and runtimes.

PAPI'S NEW 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 will be available in C and FORTRAN.

```
void  *papi_sde_init(char *lib_name, int event_count);
```

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.
`event_count` is an integer declaring the number of events that the library wishes to register.

```
void  papi_sde_register_counter(void *handle, char *event_name, int type,
      int mode, 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.
`type` is an enumeration of the type of the event.
`mode` is an integer declaring whether a counter is read-only, or read-write.
`counter` is a pointer to the actual variable that serves as the counter for this event.

```
typedef void *(*func_ptr_t)(void *);
void papi_sde_register_fp_counter(void *handle, char *event_name, int type,
func_ptr_t fp_counter, void *param);
```

Registers a function pointer to an accessor function provided by the library. Allows 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 with return type `void *` to support user-defined event types.
`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(void *handle, char *event_name, char *
event_description);
```


SPONSORED BY

ICL INNOVATIVE COMPUTING LABORATORY — THE UNIVERSITY OF TENNESSEE KNOXVILLE — U.S. DEPARTMENT OF ENERGY Office of Science — ECP EXASCALE COMPUTING PROJECT — National Nuclear Security Administration