THE UNIVERSITY of TENNESSEE
NEW HORIZONS IN SUPERCOMPUTING

SC14 New Orleans, LA | hpc matters.

PAPI

Performance Application Programming Interface

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

Heike McCraw, **Asim YarKhan**, Jack Dongarra
Innovative Computing Laboratory
University of Tennessee, Knoxville

A special Thank You to Vince Weaver, Peter Gaultney
And Thank You to all our collaborators and contributors!

Adaptation of the PAPI SC14 Presentation for an ICL Friday Seminar
Innovative Computing Lab – University of Tennessee
Dec 5 2014

Hardware Counters

Hardware performance counters available on most modern microprocessors

Can provide insight into:

1. Whole program timing
2. Instructions per cycle
3. Floating point efficiency
4. Cache behaviors
5. Branch behaviors
6. Memory and resource access patterns
7. Pipeline stalls

Hardware counter information can be obtained with:

1. Subroutine or basic block resolution
2. Process or thread attribution

PAPI: Performance API

- Middleware that provides a **consistent interface** and **methodology**
- Enables software engineers to see the relation between **software performance** and **hardware events**

SUPPORTED ARCHITECTURES:

- AMD
- CRAY
- **IBM Blue Gene Series, Q: 5D-Torus, I/O system, CNK, EMON power**
- IBM Power Series
- Intel Sandy Bridge, Ivy Bridge, **Haswell**, **Knights Corner**, (coming Soon: **Broadwell**)
- ARM Cortex A8, A9, A15, **X-Gene (ARM64)**
- NVidia Tesla, Kepler, NVML
- Infiniband
- **Intel RAPL (power/energy)**
- **Intel Xeon Phi power/energy**

Component PAPI

- provides access to components that expose performance measurement opportunities across the system as a whole, including **networking**, **I/O system**, **Compute Node Kernel**, **power/energy, CUDA, MIC, virtual machines**

PAPI - High level interface

```
#include "papi.h”
#define NUM_EVENTS 2
long_long values[NUM_EVENTS];
unsigned int Events[NUM_EVENTS]={PAPI_TOT_INS,PAPI_TOT_CYC};

 /* Start the counters */
 PAPI_start_counters((int*)Events,NUM_EVENTS);

 /* What we are monitoring… */
 do_work();

 /* Stop counters and store results in values */
 retval = PAPI_stop_counters(values,NUM_EVENTS);
```

PAPI Low level interface

```
#include "papi.h"
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC};
int EventSet;
long_long values[NUM_EVENTS];
/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* Add Flops and total cycles to the eventset */
retval = PAPI_add_events(EventSet,Events,NUM_EVENTS);
/* Start the counters */
retval = PAPI_start(EventSet);

do_work();  /* What we want to monitor*/

/*Stop counters and store results in values */
retval = PAPI_stop(EventSet,values);
```

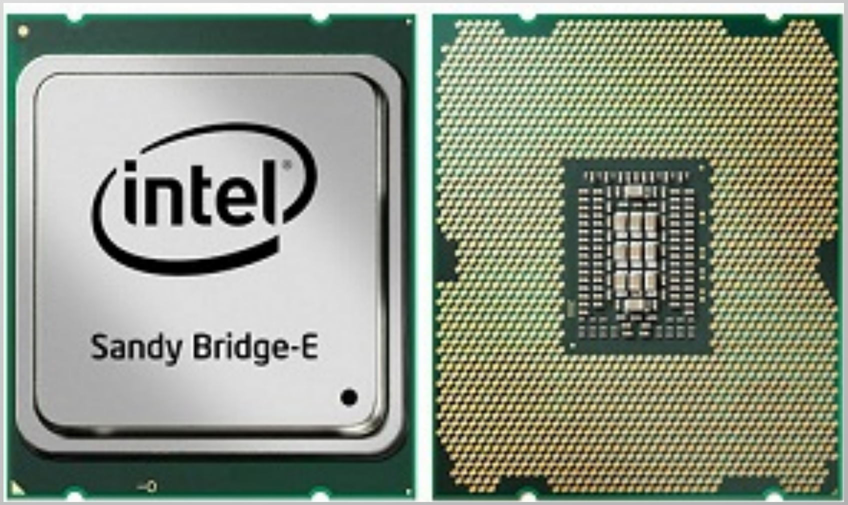
Work in Progress

PAPI POWER – LIBMSR

Scheduling Power with Processor Hardware: Intel's RAPL

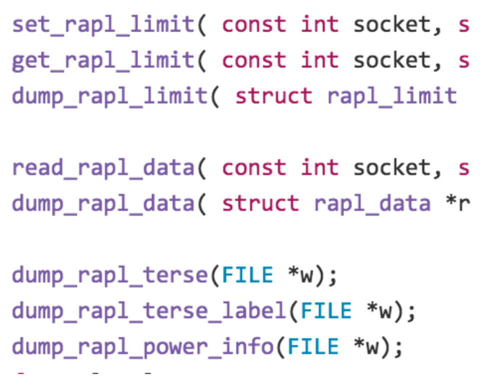
- Runtime Average Power Limit (RAPL)
  - Measures cumulative joules (power x time)
  - Three separate power meters
  - Clamping on package and DRAM power
- Turbo suppression
- Effective frequency

Can place user-specified limit on average power over a user-specific time window.



Beyond DVFS: A First Look at Performance Under a Hardware-Enforced Power Bound, HPPAC 2012, Barry Rountree, Dong H. Ahn, Bronis R. de Supinski, David K. Lowenthal, Martin Schulz

In Progress: MSR and libmsr



```
set_rapl_limit( const int socket, s
get_rapl_limit( const int socket, s
dump_rapl_limit( struct rapl_limit

read_rapl_data( const int socket, s
dump_rapl_data( struct rapl_data *r

dump_rapl_terse(FILE *w);
dump_rapl_terse_label(FILE *w);
dump_rapl_power_info(FILE *w);
```

- Power measurement and control
  - Working with LLNL (Barry Rountree)
- Uses information from /dev/cpu/*/msr
  - Kernel modules: msr and msr-safe (done)
  - Library interface with more control: libmsr (todo in PAPI)
    - Concerns of root access, safety, read-write, read-only
  - Power handling using RAPL (Running Average Power Limit)
    - monitor, control, and get notifications on SOC power
  - Set power envelope (multiple time scales) and other things
    - Hardware-Controlled Performance States (HWP),
  - PAPI does not normally "set" events … only measurement

libMSR: Flexible Access to MSR on x86/64 Linux Systems

- Modern processors offer a wide range of control and measurement features. While those are **traditionally accessed through libraries like PAPI**, some newer features do no longer follow the traditional model of counters that can used to only read the state of the processor. For example, Precise Event Based Sampling (PEBS) can **generate records that requires a kernel memory for storage**. Additionally, new features like **power capping** and **thermal control** require similar new access methods. All of these features are ultimately controlled through **Model Specific Registers (MSRs)**. We therefore need new mechanisms to make such features available to tools and ultimately to the user. libMSR provides a convenient interface to access MSRs and to allow tools to utilize their full functionality.
- In particular, libMSR will provide access to the following features:
  - **clocks.c**—Access to the tsc and other clocks
  - **pebs.c**—Support for PEBS hardware counters
  - **counters.c** - More traditional performance counters
  - **cpuid.c**—Read and parse hardware capabilities
  - **turbo.c**—Enable/suppress turbo, calculate effective frequency
  - **uncore.c**—Support for Uncore (unified core, package-wide)
  - **rapl.c—Measure and cap package and DRAM power**
  - **thermal.c**—Read processor and DRAM temperature
- libMSR is **currently** mainly **targeting Intel SandyBridge** processors, but we are planning to add support for other (mainly newer) processor generations as well as support for similar features in AMD processors.
- libMSR **requires user-level access to /dev/cpu/X/msr to read and write MSR registers**. While this is often no problem on individual workstations, it is a **significant security risk on multi-user systems and large scale compute clusters**. The libMSR package therefore **provides** a matching "**safe**" version of the "**msr**" kernel module that allows controlled access to MSRs.

- **Available at github: https://github.com/scalability-llnl/libmsr**

Algorithm Awareness:

PAPI FOR PARSEC

PARALLEL **R**UNTIME **S**CHEDULING AND **E**XECUTION **C**ONTROLLER

PAPI and PaRSEC

Parallel Runtime Scheduling and Execution Controller

PaRSEC: (UTK framework for managing a compact-DAG of tasks)

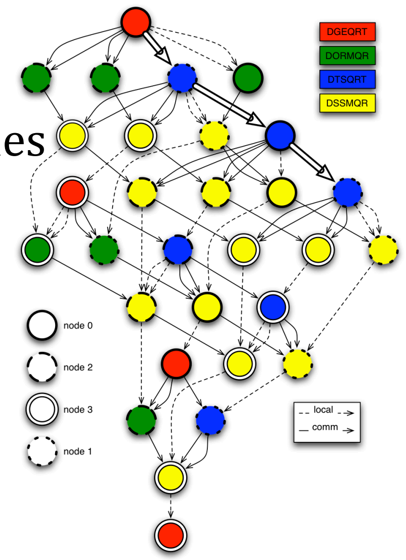
- Framework for architecture-aware scheduling of data-dependent micro-tasks on distributed many-core heterogeneous architectures

→ **Performance tools become more and more important for task-based dataflow and execution systems**

→ **Analysis features that show the connection of the dataflow and the execution profile/trace is extremely beneficial**

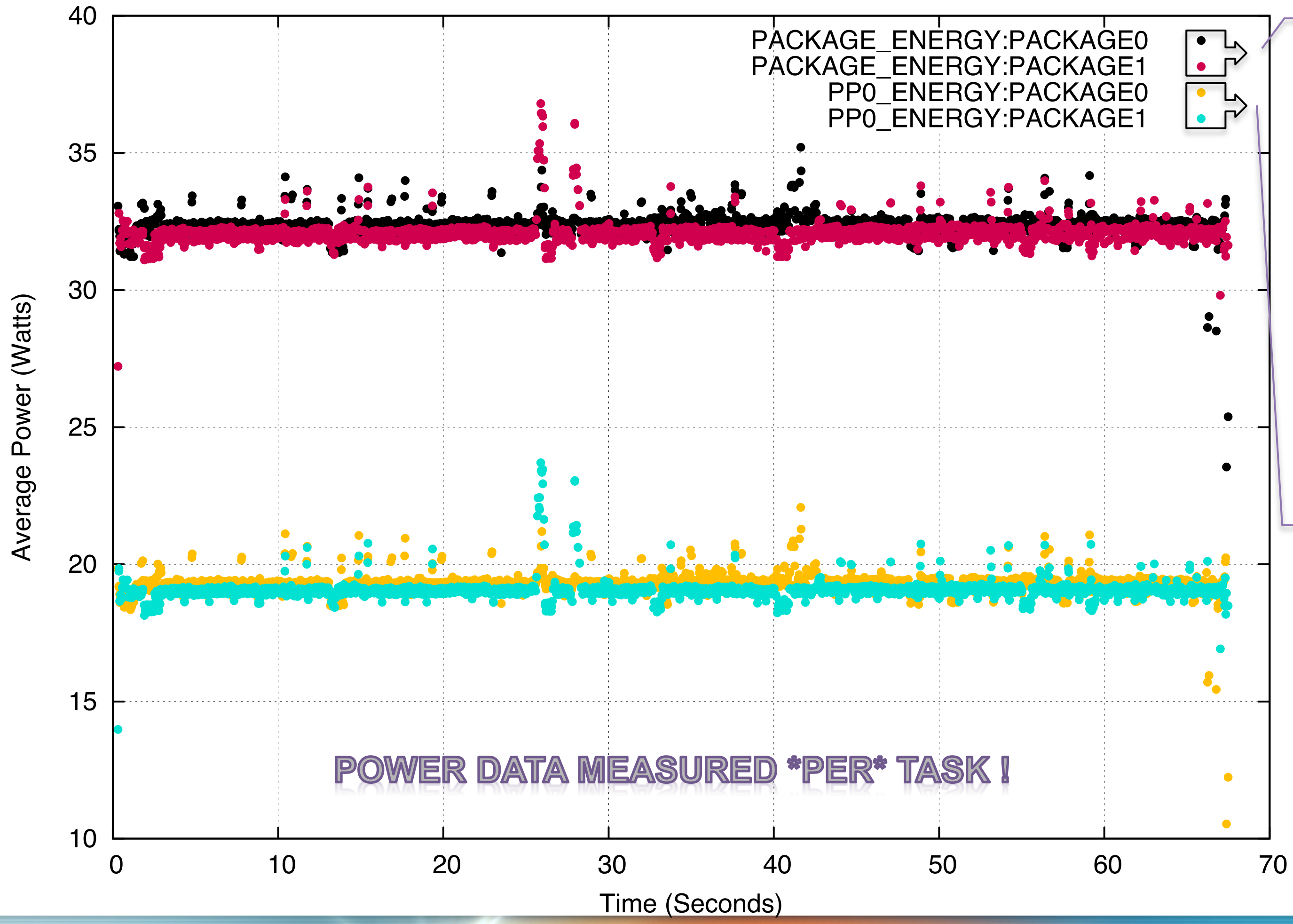
PAPI in PaRSEC:

- Integrated in **PaRSEC's P**erformance **INS**trumentation modules
- **PINS** modules can be selectively loaded and used by users
- Enables users to measure performance counter data for **each task/node in a DAG** (Directed Acyclic Graph)
- Everything supported by PAPI can be measured in PaRSEC at "per task" granularity



Power per Task: PaRSEC

Average Power Usage for **dgeqrf @ 30.8 GFLOPs** -- MatrixSize = 11,584 -- TileSize = 724
Sandy Bridge EP 2.60GHz, 2 sockets, running on 1 (out of 8) core per socket

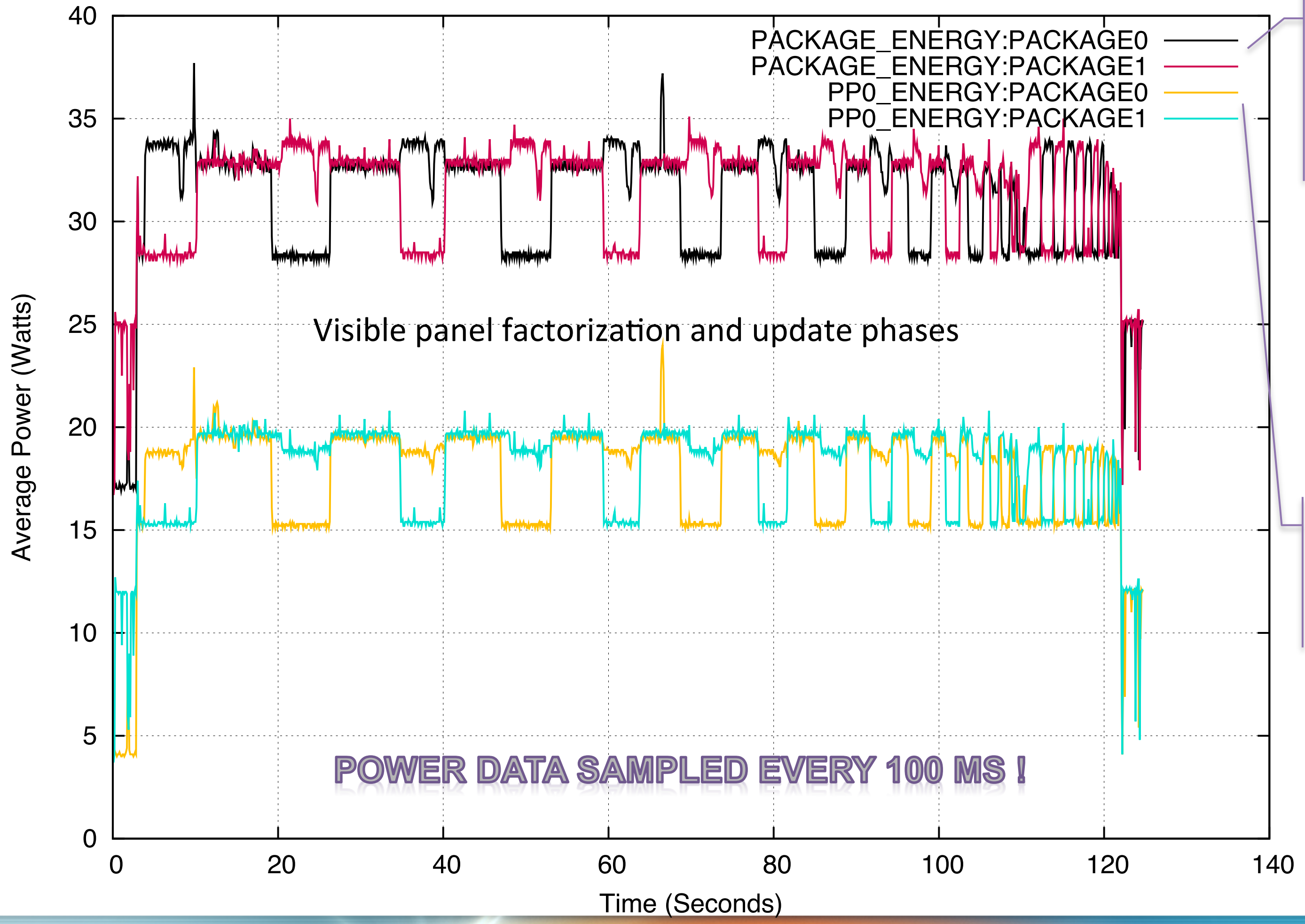


Total Power on Socket 0, 1 (includes **everything**)

Power of **cores only** on Socket 0, 1

Power Sampling: ScaLAPACK

Average Power Usage for **pdgeqrf @ 19.6 GFLOPs** -- MatrixSize = 11,584 -- TileSize = 724
Sandy Bridge EP 2.60GHz, 2 sockets, running on 1 (out of 8) core per socket

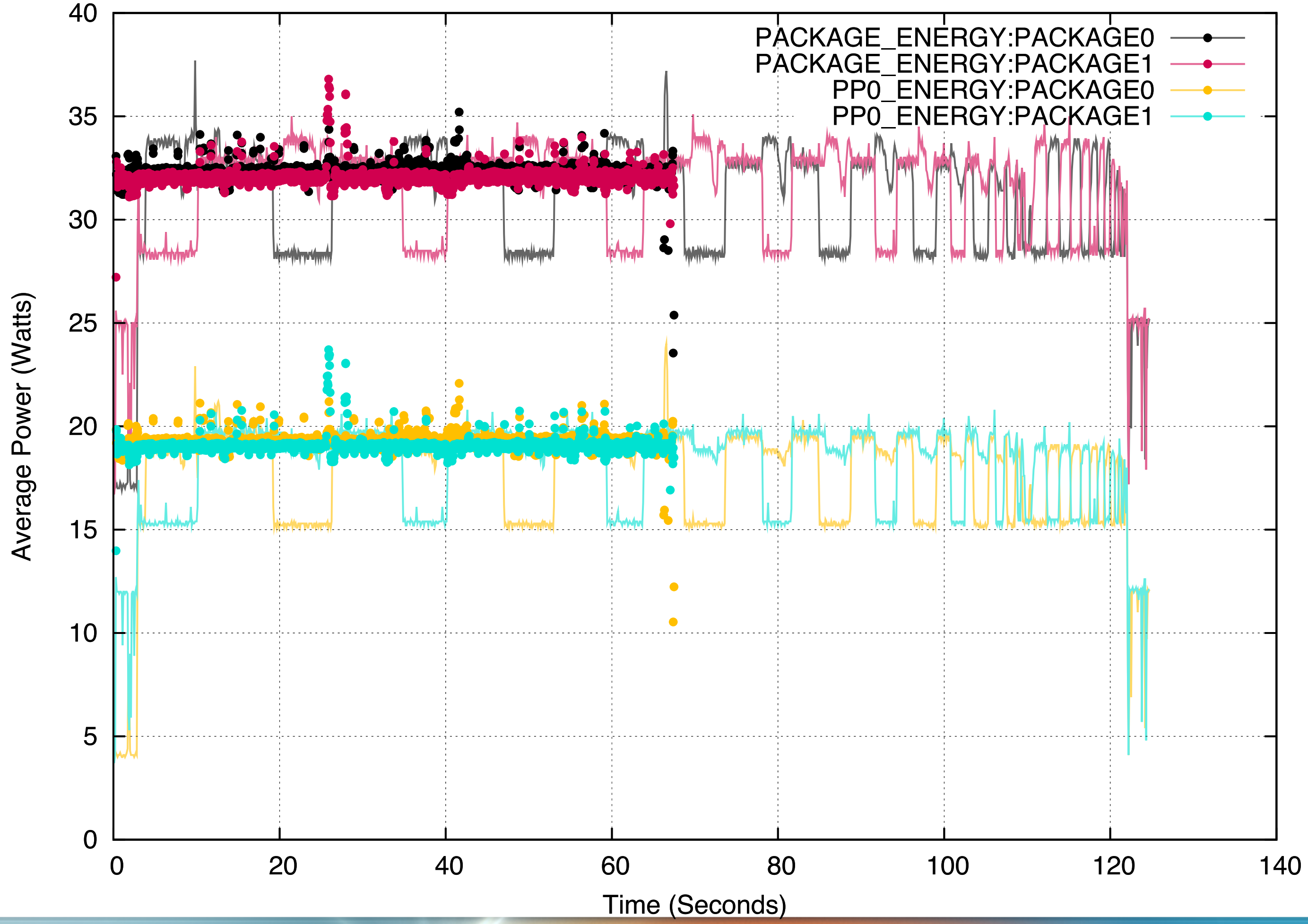


Total Power on Socket 0, 1 (includes **everything**)

Power of **cores only** on Socket 0, 1

PAPI Power Measurements

Average Power Usage for **(p)dgeqrf** -- MatrixSize = 11,584 -- TileSize = 724
Sandy Bridge EP 2.60GHz, 2 sockets, running on 1 (out of 8) core per socket



PaRSEC:

30.8 GFLOPs

Total Energy:

4.35 kWs

ScaLAPACK:

19.6 GFLOPs

Total Energy:

7.79 kWs

Architecture Awareness:

PAPI POWER MEASUREMENTS ON INTEL® XEON PHI™

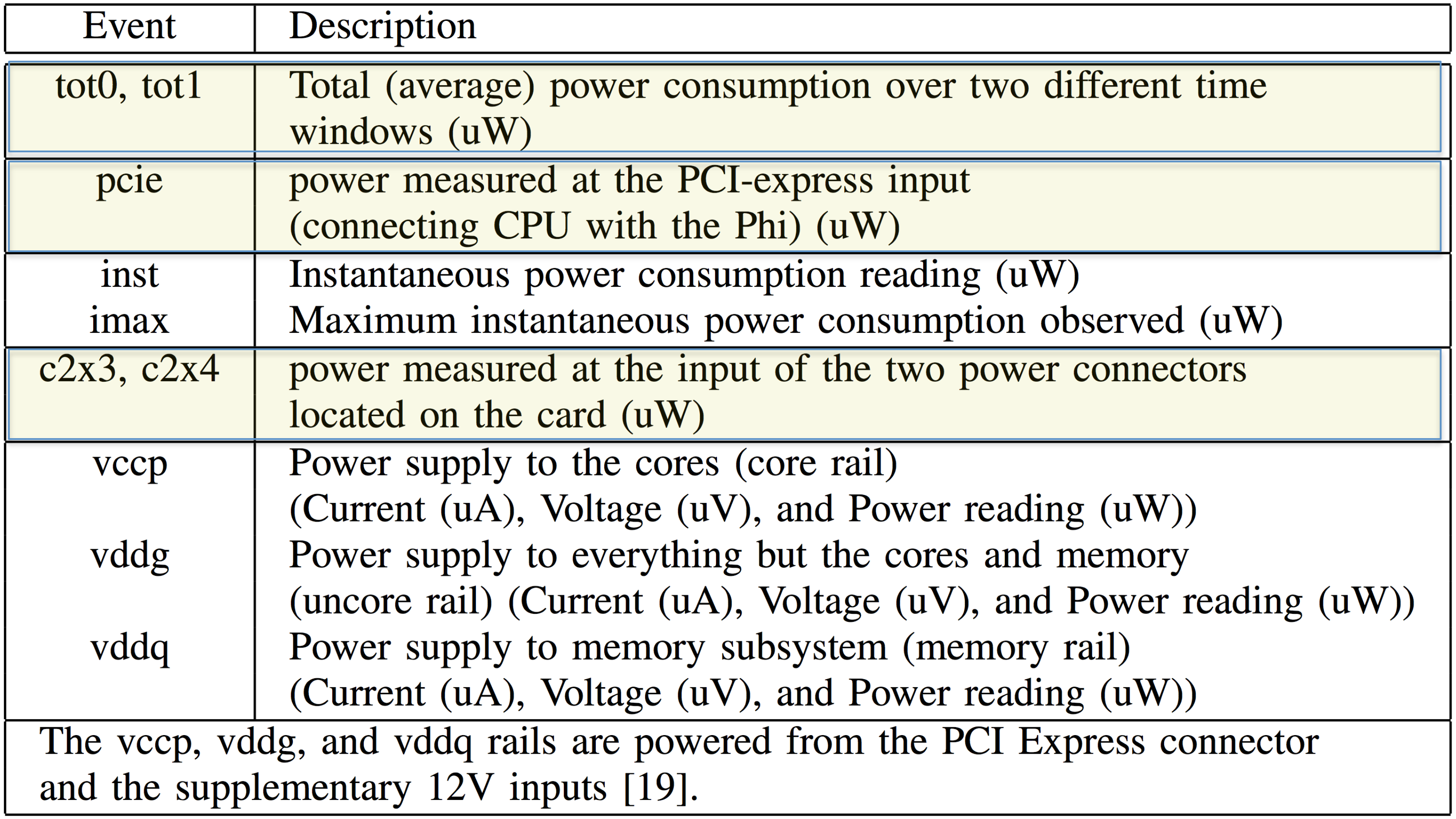
Power and Energy on Xeon Phi™

PAPI offers two components for the Xeon Phi (aka Intel Many Integrated Core architecture)

micpower	host_micpower
Measurements in **native-mode**: Both application and PAPI run directly on coprocessor and its Linux OS Values reported in /sys/class/micras/power	Measurements in **offload-mode**: Application runs directly on coprocessor; PAPI is offloaded from the host system Power information exported via MicAccessAPI (distr. with Intel MPSS)

Example:
Hessenberg Reduction
using host_micpower with a sampling rate of 100ms

Power Events on Xeon Phi™



Event	Description
tot0, tot1	Total (average) power consumption over two different time windows (uW)
pcie	power measured at the PCI-express input (connecting CPU with the Phi) (uW)
inst	Instantaneous power consumption reading (uW)
imax	Maximum instantaneous power consumption observed (uW)
c2x3, c2x4	power measured at the input of the two power connectors located on the card (uW)
vccp	Power supply to the cores (core rail) (Current (uA), Voltage (uV), and Power reading (uW))
vddg	Power supply to everything but the cores and memory (uncore rail) (Current (uA), Voltage (uV), and Power reading (uW))
vddq	Power supply to memory subsystem (memory rail) (Current (uA), Voltage (uV), and Power reading (uW))
The vccp, vddg, and vddq rails are powered from the PCI Express connector and the supplementary 12V inputs [19].	

Power on Xeon Phi™: Hessenberg

Intel® Xeon Phi™ (Knights Corner), 244 cores

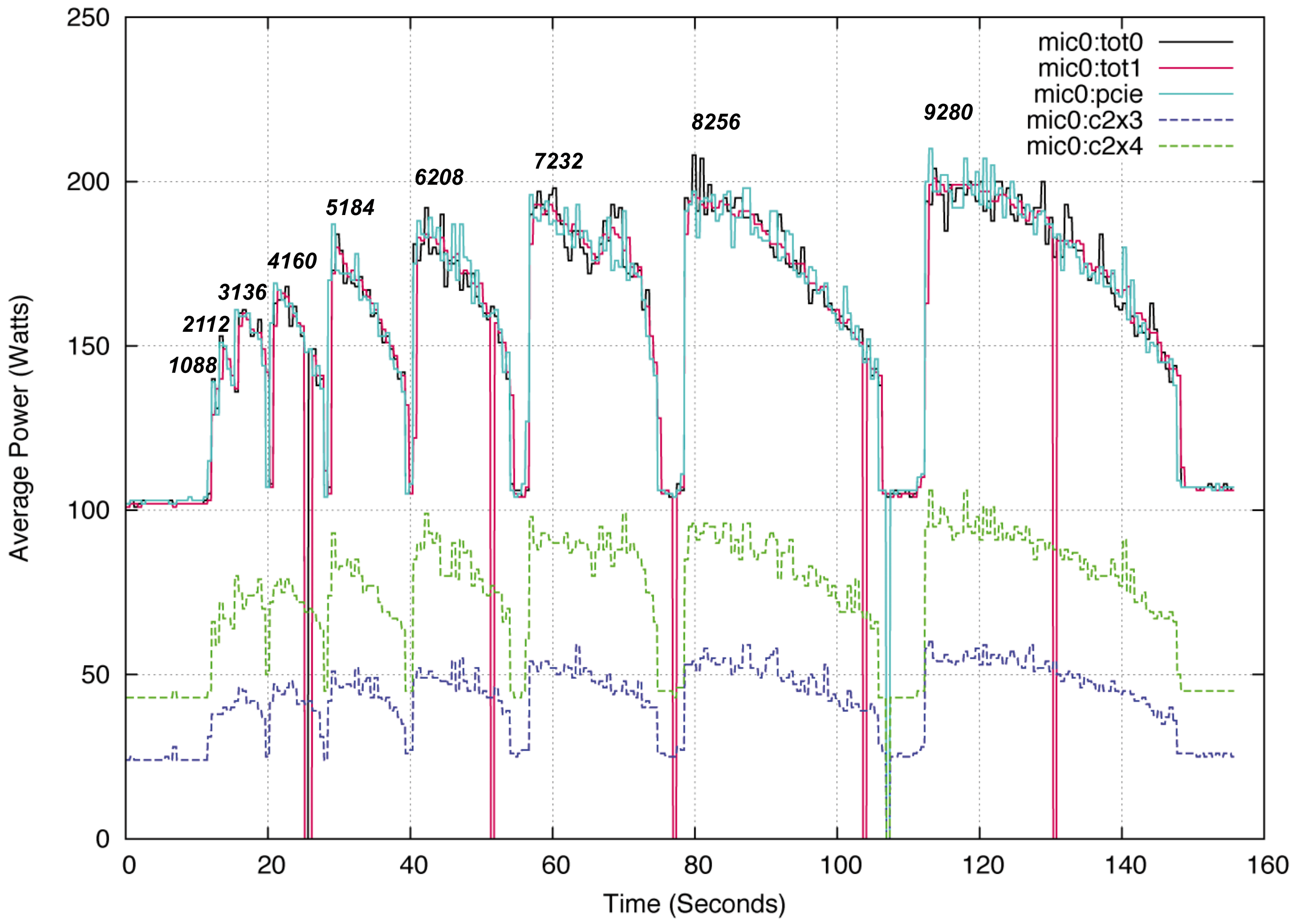
+ memory-bound kernel (GEMVs and GEMMs)

+ 9 computations with different matrix sizes

<u>power usage mimics computational intensity:</u>

+ Factorization starts on entire matrix → consumes most power

+ As factorization progresses, it operates on smaller matrices → consumes less power



Energy on Xeon Phi™: Hessenberg

Intel® Xeon Phi™ (Knights Corner), 244 cores

+ memory-bound kernel
  (GEMVs and GEMMs)

+ 9 computations with different
  matrix sizes

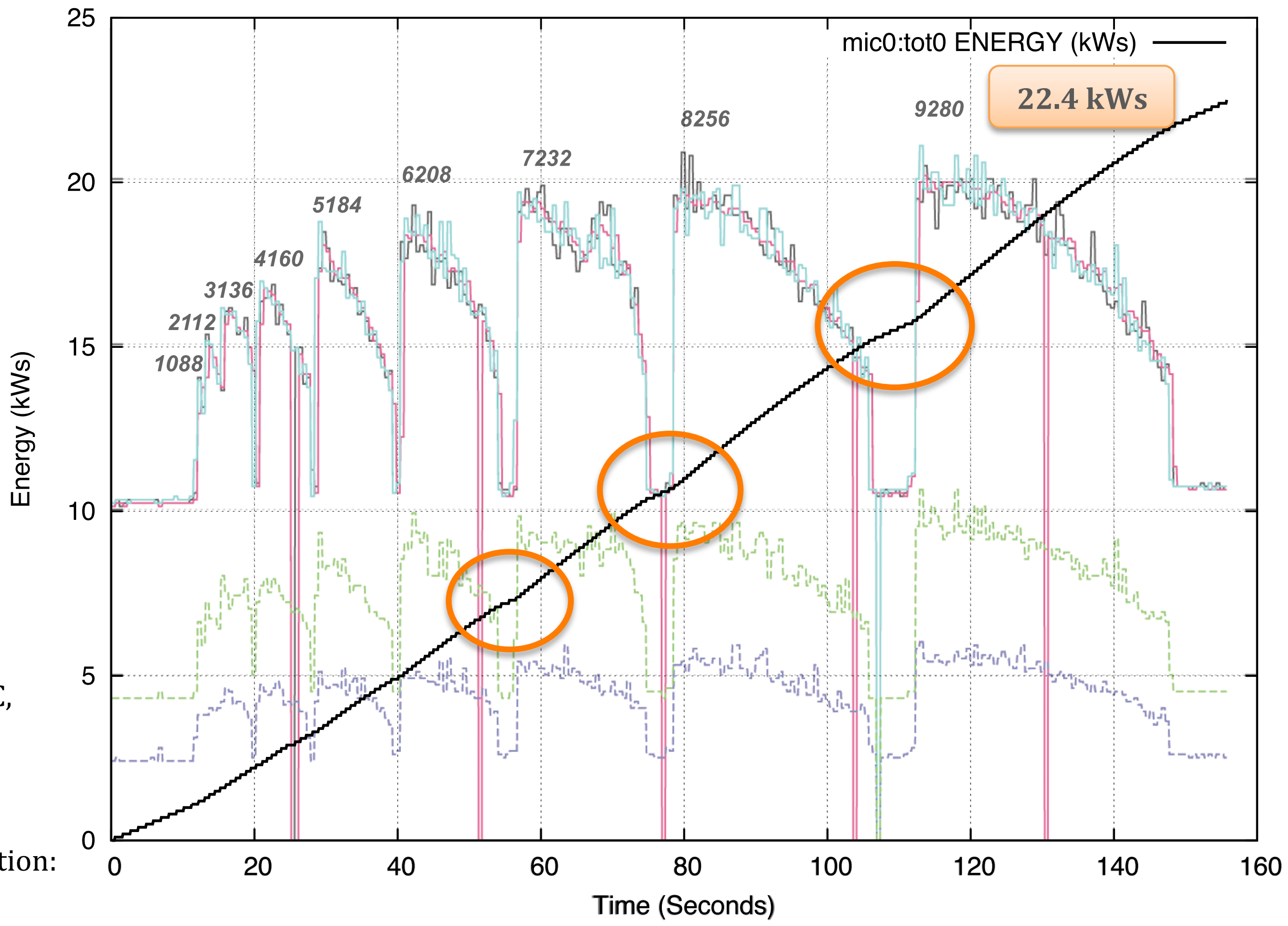
power usage mimics
computational intensity:

+ Factorization starts on
  entire matrix
  → consumes most power

+ As factorization progresses, it
  operates on smaller matrices
  → consumes less power

+ in between computations,
  data exchanged with host,
  power usage goes down on MIC,
  slop of energy curve decreases

+ Total Energy consumption for
  entire computation till completion:
  22.4 kWs (Total: Window 0)



Summary

- With larger and more complex HPC systems on the horizon, energy efficiency has become one of the critical constraints

PAPI's new components:

- Perf counter monitoring at task granularity for dataflow runtimes
  - Including Power monitoring per task
- Power/Energy components
  - for Intel® Xeon Phi™; for IBM Blue Gene/Q
- Work in progress: MSR, MSR-safe and libmsr

PAPI users can monitor power (in addition to traditional HW perf counter data):

- without modifying their applications
- without learning a new set of library and instrumentation primitives

3rd Party Tools applying PAPI

- PaRSEC (UTK) http://icl.utk.edu/parsec/
- TAU (U Oregon) http://www.cs.uoregon.edu/research/tau/
- PerfSuite (NCSA) http://perfsuite.ncsa.uiuc.edu/
- HPCToolkit (Rice University) http://hpctoolkit.org/
- KOJAK and SCALASCA (FZ Juelich, UTK) http://icl.utk.edu/kojak/
- VampirTrace and Vampir (TU Dresden) http://www.vamir.eu
- Open|Speedshop (SGI) http://oss.sgi.com/projects/openspeedshop/
- SvPablo (UNC Renaissance Computing Institute) http://www.renci.org/research/pablo/
- ompP (UTK) http://www.ompp-tool.com

Contact

Innovative Computing Lab at the University of Tennessee

- Asim YarKhan (yarkhan@icl.utk.edu)
- Heike McCraw (mccraw@icl.utk.edu)

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

THE UNIVERSITY of TENNESSEE
NEW HORIZONS IN SUPERCOMPUTING