Recent Advances in the Performance API (PAPI)

Collaborators:
Heike Jagode
Asim Yarkhan
Jack Dongarra

10th Scalable Tools Workshop
Anthony Danalis

Lake Tahoe, California
August 1-4, 2016

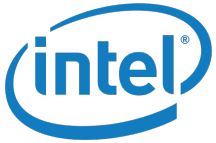
INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

PAPI

- Middleware that provides a **consistent interface** and methodology for the performance counter hardware found in most major microprocessors
- PAPI enables software engineers to see, in near real time, the relation between **SW performance** and **HW events**

SUPPORTED ARCHITECTURES:

- AMD
- CRAY: Aris, Gemini, power
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, CNK, EMON power/energy
- IBM Power Series
- Intel Westmere, Sandy|Ivy Bridge, Haswell, Broadwell, **Skylake**, Knights Corner | **Landing**
- ARM Cortex A8, A9, A15**, ARM64**
- NVidia Tesla, Kepler, NVML: **CUDA support for multiple GPUs; PC Sampling**
- Infiniband
- Intel RAPL (power/energy); **power capping**
- Intel KNC**, KNL power/energy**



COMPONENT PAPI:

- provides access to a collection of components that expose performance measurement opportunities across the system as a whole, including network, the I/O system, the Compute Node Kernel, power/ energy

PAPI CPU Components: KNC vs. KNL

PAPI offers two components to the CPU counters:

PAPI Components	Knights Corner	Knights Landing
perf_event: Linux perf_event CPU **core** events	**PMU's supported**: perf, perf_raw, knc **# of Native Events**: 140 **# of Preset Events**: 14 **# of Counters**: 2	**PMU's supported**: perf, perf_raw, knl, ix86arch **# of Native Events**: 182 **# of Preset Events**: 26 **# of Counters**: 5
perf_event_uncore: Linux perf_event CPU **uncore** and northbridge events	---	**PMU's supported**: e.g. Memory, on-die interconnect, IO, Memory-to-PCIe event support **# of Native Events**: 894

See next slide

List of the 26 PAPI Preset Events for KNL

Preset Events	*Description*
PAPI_L1_DCM	*Level 1 data cache misses*
PAPI_L1_ICM	*Level 1 instruction cache misses*
PAPI_L1_TCM	*Level 1 cache misses*
PAPI_L2_TCM	*Level 2 cache misses*
PAPI_TLB_DM	*Data translation lookaside buffer misses*
PAPI_L1_LDM	*Level 1 load misses*
PAPI_L2_LDM	*Level 2 load misses*
PAPI_STL_ICY	*Cycles with no instruction issue*
PAPI_BR_UCN	*Unconditional branch instructions*
PAPI_BR_CN	*Conditional branch instructions*
PAPI_BR_TKN	*Conditional branch instructions taken*
PAPI_BR_NTK	*Conditional branch instructions not taken*
PAPI_BR_MSP	*Conditional branch instructions mispredicted*
PAPI_TOT_INS	*Instructions completed*
PAPI_LD_INS	*Load instructions*
PAPI_ST_INS	*Store instructions*
PAPI_BR_INS	*Branch instructions*
PAPI_RES_STL	*Cycles stalled on any resource*
PAPI_TOT_CYC	*Total cycles*
PAPI_LST_INS	*Load/store instructions completed*
PAPI_L1_DCA	*Level 1 data cache accesses*
PAPI_L1_ICH	*Level 1 instruction cache hits*
PAPI_L1_ICA	*Level 1 instruction cache accesses*
PAPI_L2_TCH	*Level 2 total cache hits*
PAPI_L2_TCA	*Level 2 total cache accesses*
PAPI_REF_CYC	*Reference clock cycles*

PAPI POWER Components: KNC vs. KNL

PAPI offers four components for Power/Energy monitoring:

PAPI Components	KNC	KNL
powercap: Reads RAPL results via Linux POWERCAP interface	---	**# of Native Events**: 15 (requires no special permissions)
rapl: RAPL results (raw access to the underlying MSRs)	---	**# of Native Events**: 14 (requires root privilege)
micpower: Reading power in *native* mode	Power values reported in /sys/class/micras/power **# of Native Events**: 16	---
host_micpower: Reading power in *offload* mode	Power values exported via MicAccessAPI (MPSS) **# of Native Events**: 16	---

PAPI POWER on Knights Landing

The following power domains are supported:

- PACKAGE: Processor die
- DRAM (Memory): Directly-attached DRAM
- PP0 (Power Plane 0): Processors cores subsystem

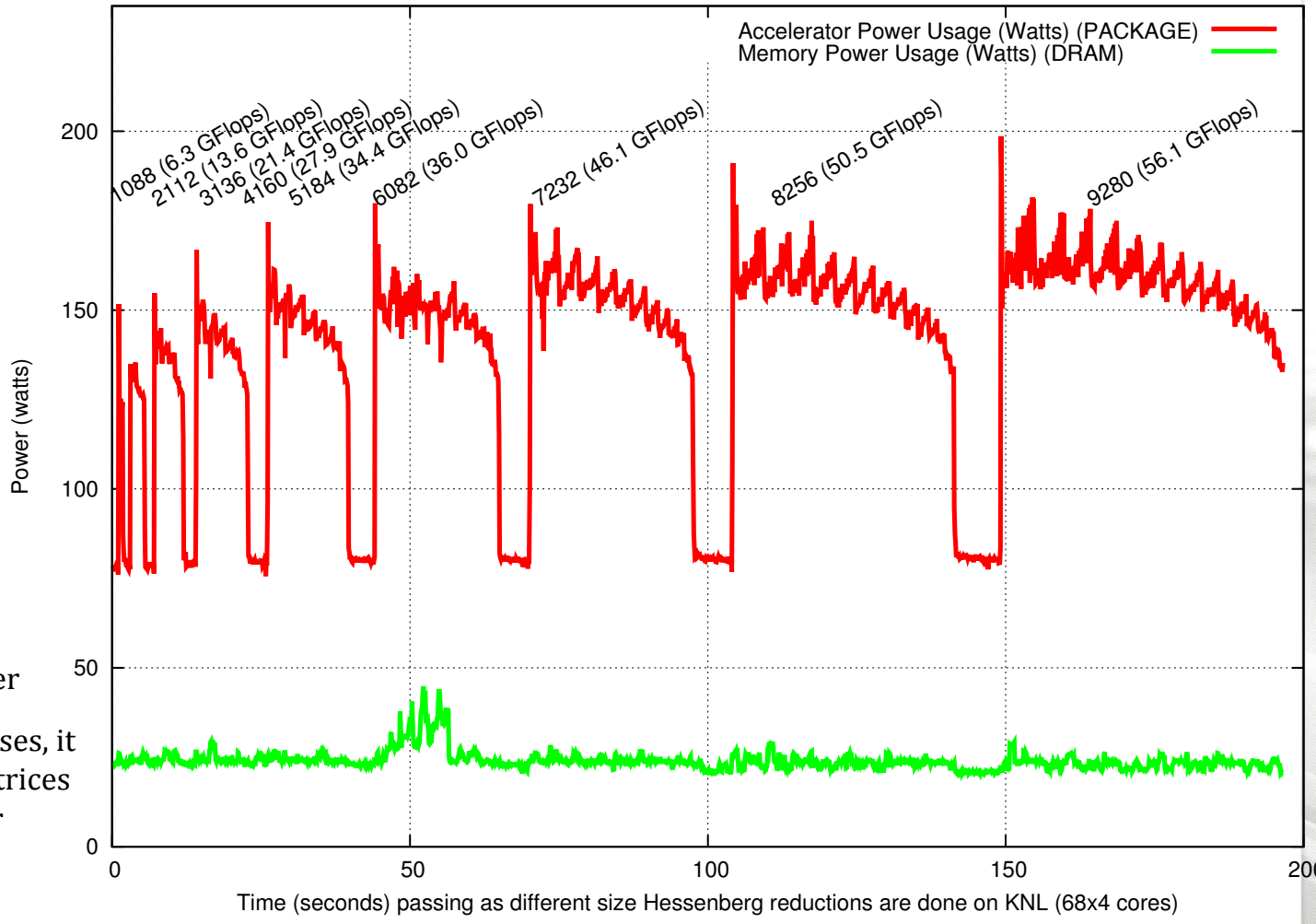
Simple Verification Test: Naive MMM (1024x1024):

```
Scaled Energy Measurements:
          PACKAGE_ENERGY           8216.792419 J  (Average Power 84.5W)
          DRAM_ENERGY              2539.645264 J  (Average Power 26.1W)
Energy Measurement Counts:
          PACKAGE_ENERGY_CNT        134623927
          DRAM_ENERGY_CNT            41609548
Scaled Fixed Values:
          THERMAL_SPEC                 215.000 W
          MAXIMUM_POWER                258.000 W
          MAXIMUM_TIME_WINDOW            0.046 s
Fixed Value Counts:
          THERMAL_SPEC_CNT              1720
          MAXIMUM_POWER_CNT             2064
          MAXIMUM_TIME_WINDOW_CNT         47
```

PAPI POWER on KNL: Hessenberg (MKL's dgehrd)

Intel® Xeon Phi™ Knights Landing, 68 cores (4 HW threads/core)

+ memory-bound kernel (GEMVs and GEMMs)
+ 9 computations with different matrix sizes



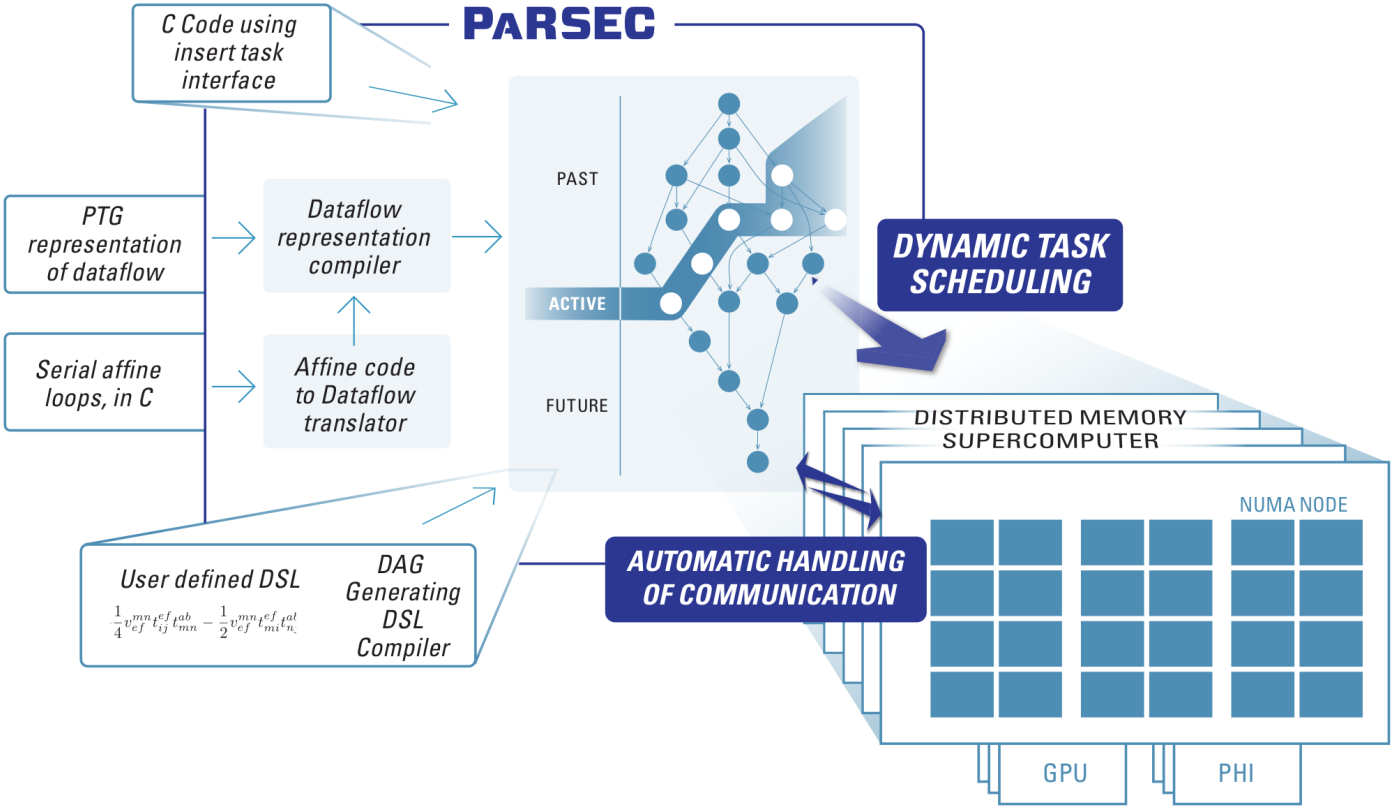
<u>power usage mimics computational intensity:</u>

+ Factorization starts on entire matrix
  → consumes most power
+ As factorization progresses, it operates on smaller matrices
  → consumes less power

PAPI FOR PARSEC

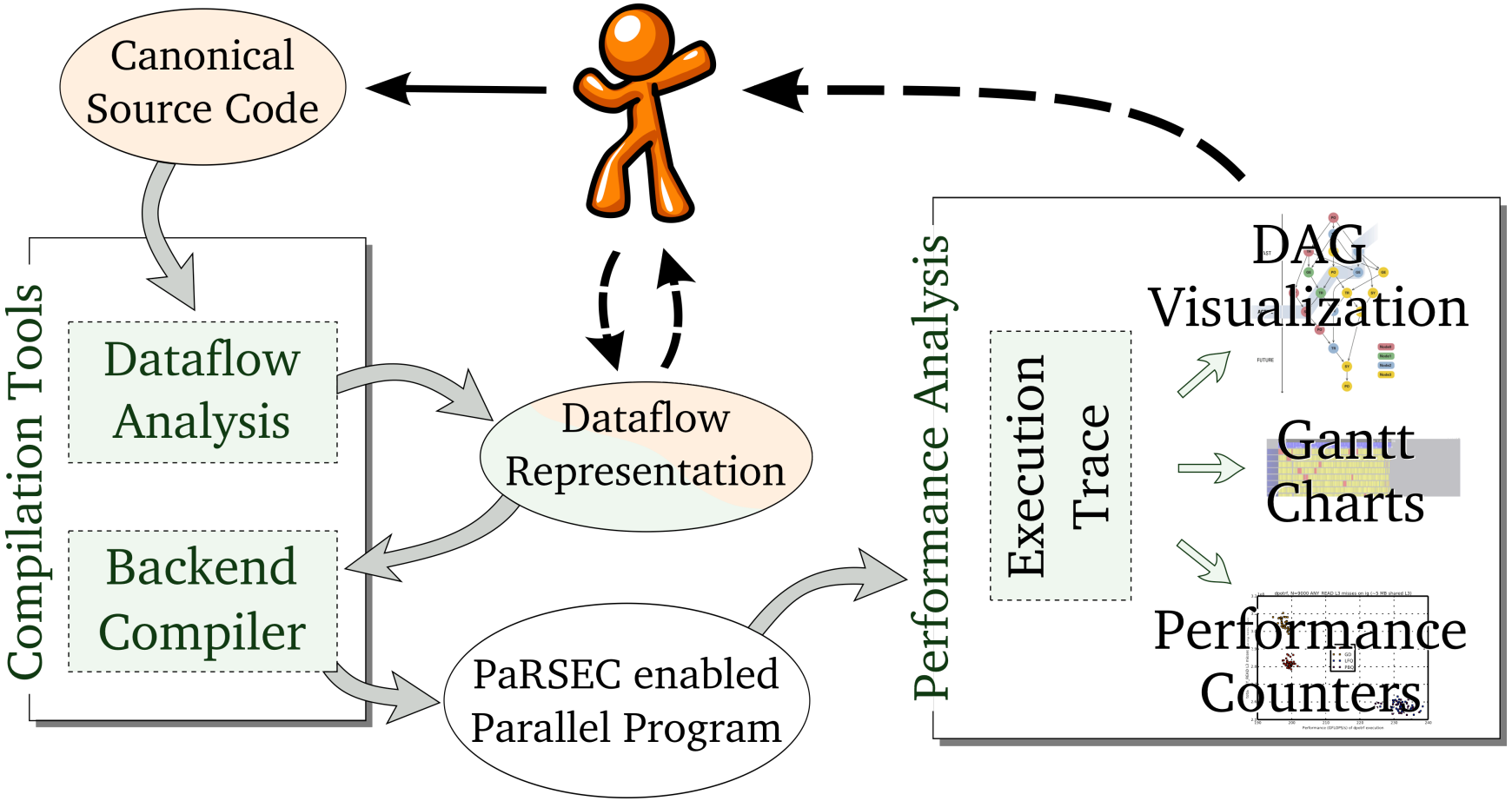
PARALLEL **R**UNTIME **S**CHEDULING AND **E**XECUTION **C**ONTROLLER

Dataflow-driven Programming Models



- Develop for a portability layer, not an architecture
- Let the runtime deal with the hardware characteristics
- **Task-scheduling: PaRSEC,** StarSS, StarPU, Swift, Parallex, Quark, Kaapi, DuctTeip

PaRSEC Features



PAPI and PaRSEC

Parallel Runtime Scheduling and Execution Controller

PaRSEC:

- Generic framework for architecture aware scheduling of micro-tasks on distributed many-core heterogeneous architectures

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

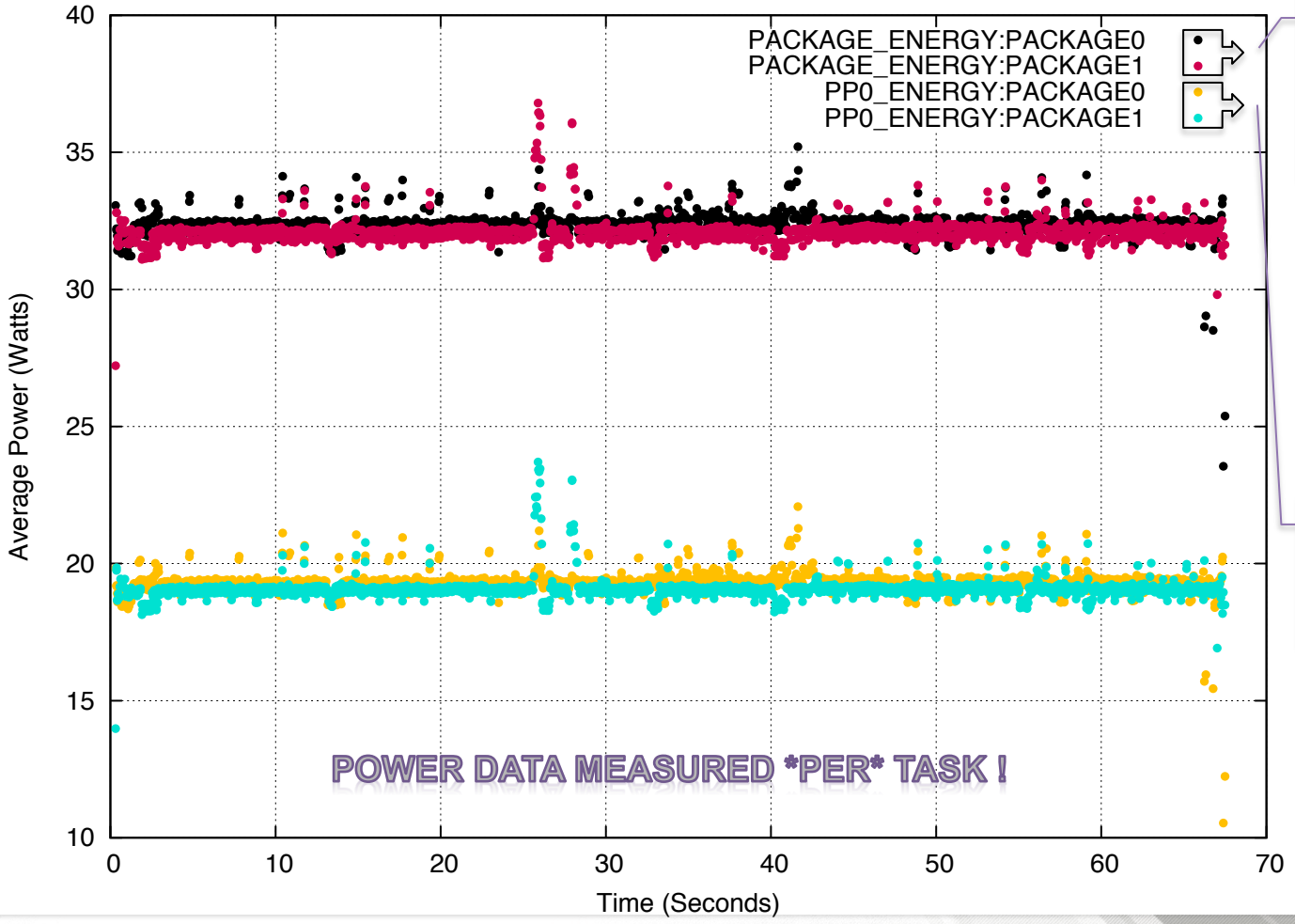
→ **Analysis features that show the connection of the dataflow and the execution profile/trace is extremely beneficial**

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

PAPI 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

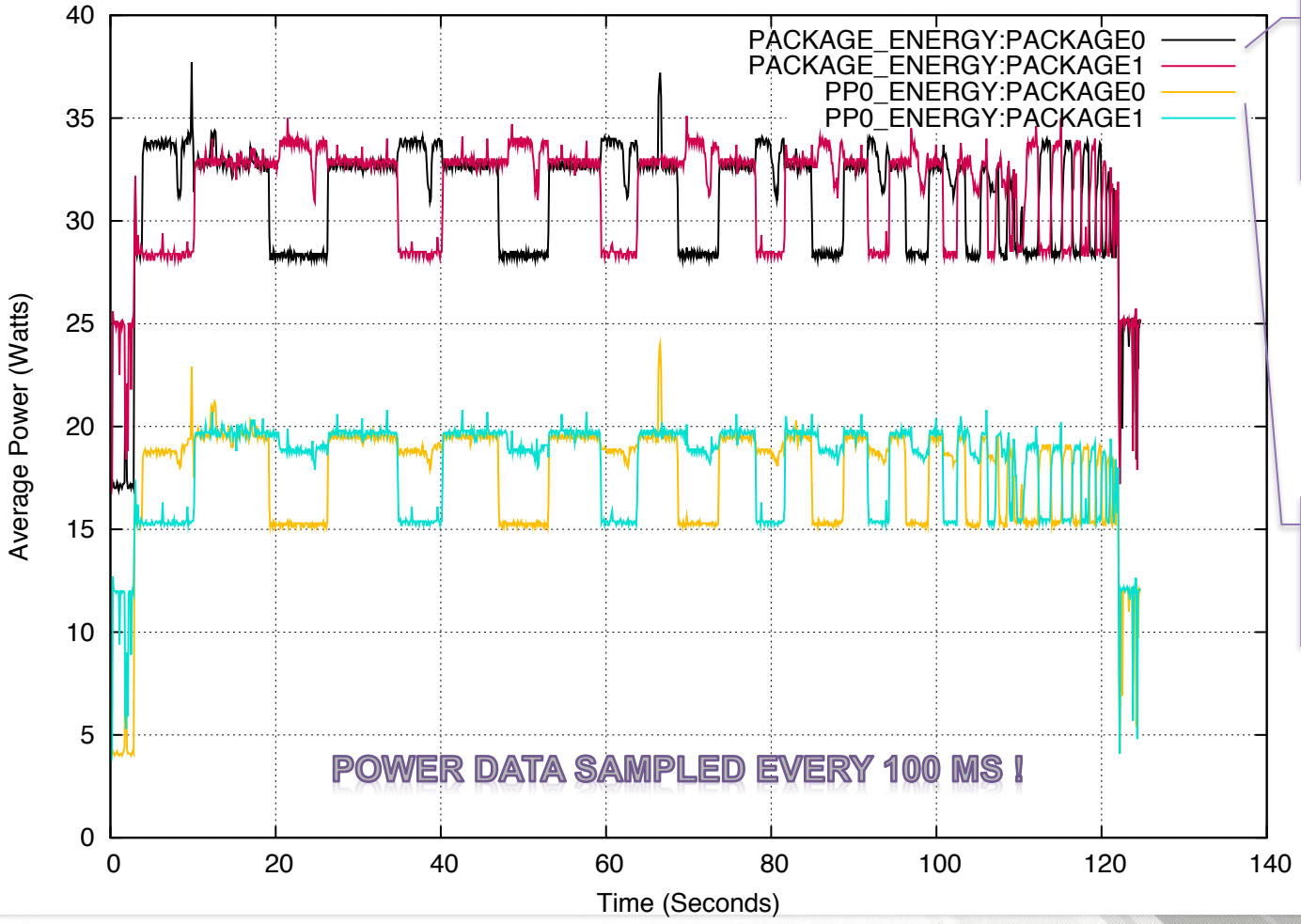


Total Power on Socket 0, 1 (includes everything)

Power of cores only on Socket 0, 1

PAPI 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

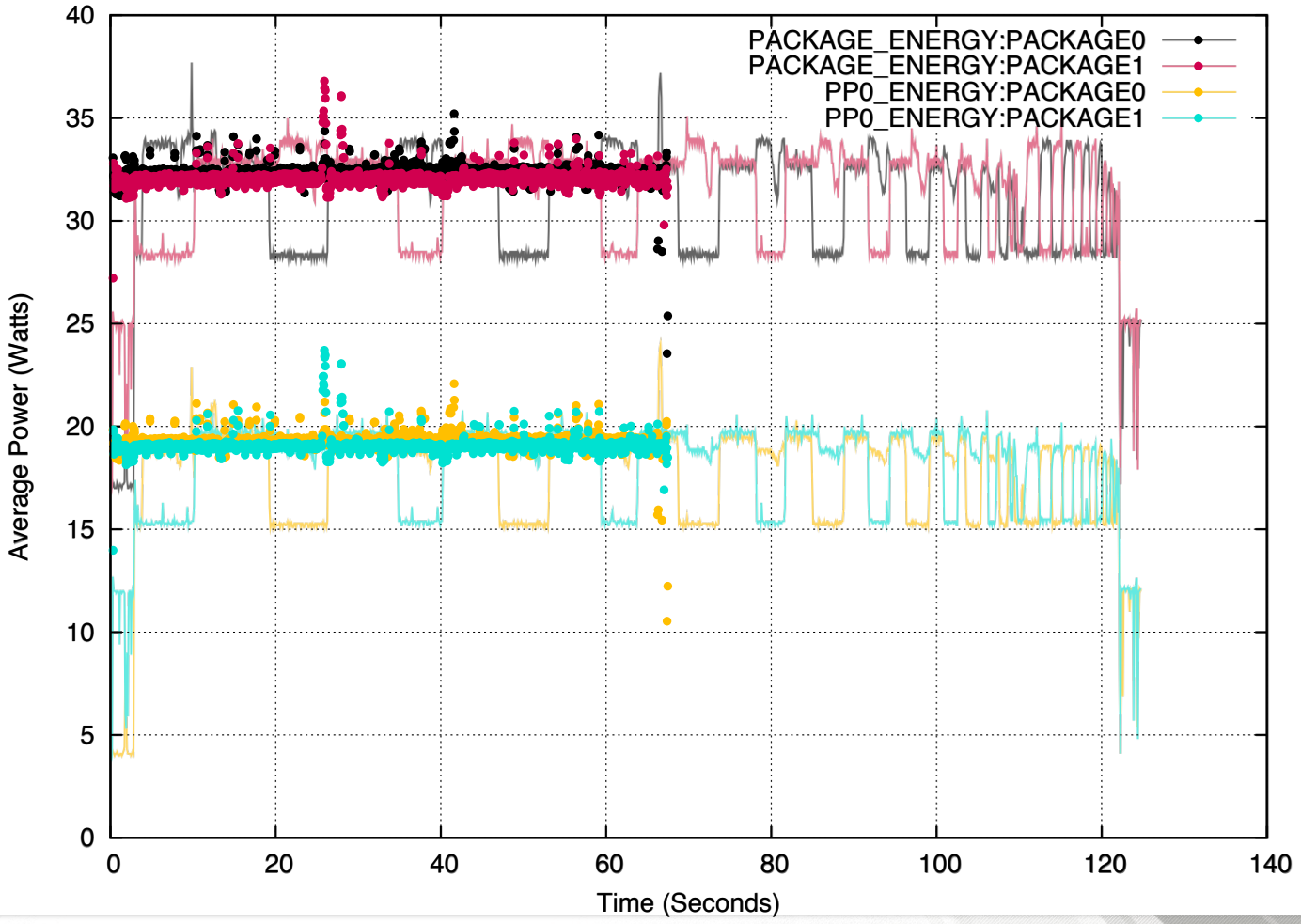


Total Power on Socket 0, 1 (includes everything)

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

PAPI POWER CONTROLLING

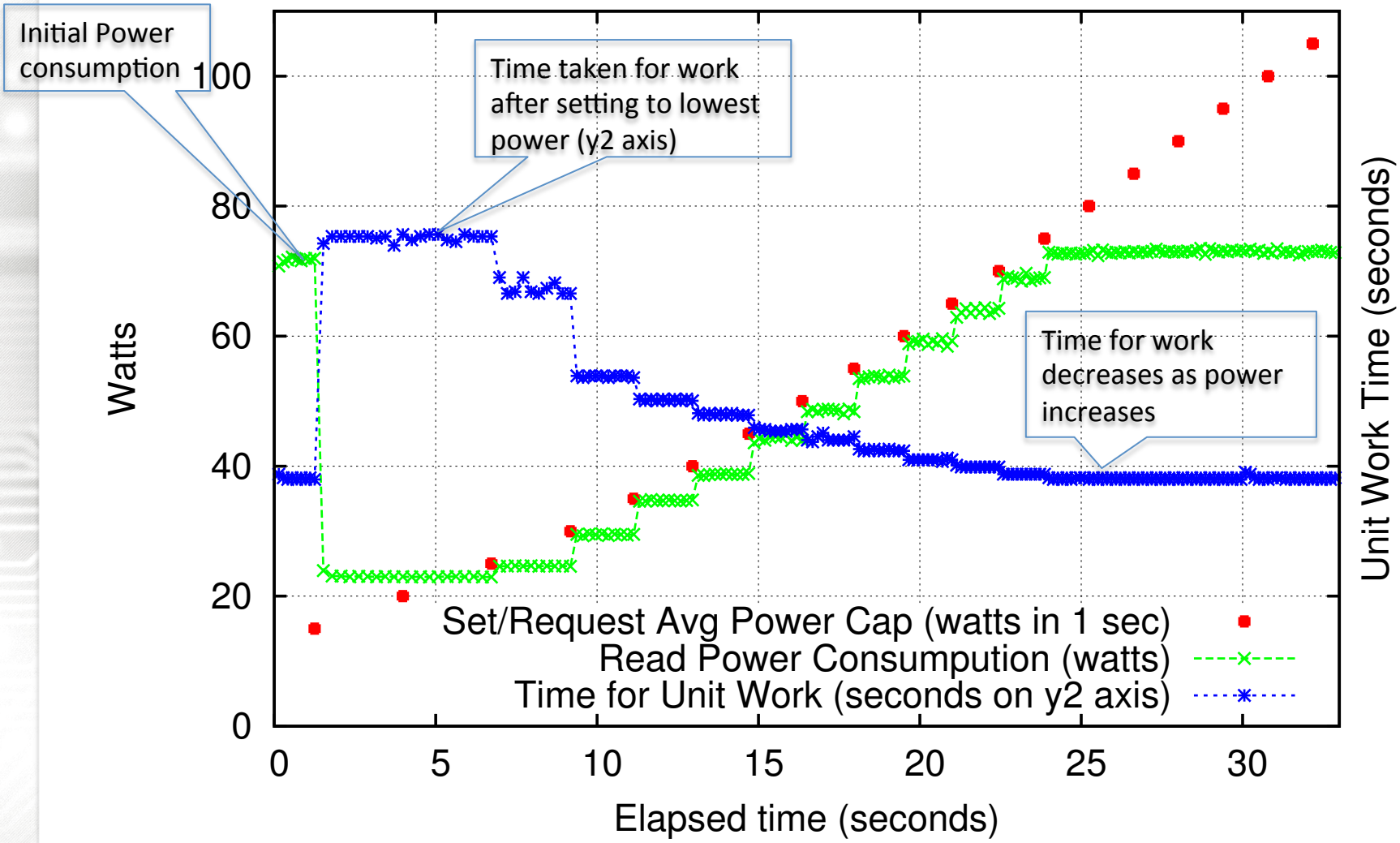
READING AND WRITING POWER

RAPL & msr-safe & libmsr

- RAPL – Running Average Power Limit
  - Intel Sandybridge or better
  - Models energy at package, DRAM controller, CPU core (PP0), graphics uncore (PP1)
- Providing write access to MSRs can be unsafe
  - Can have large effect on machine
  - To use, you need to make MSRs writeable, capability-executable, static-only executable, paranoid setting in kernel !!!
- msr-safe and libmsr from LLNL
  - msr-safe provides a safer whitelist controlled access to MSRs
    - kernel module provides /dev/cpu/*/safe_msr
  - libmsr is a library to simplify access to MSRs
- PAPI libmsr component in PAPI to WRITE values
  - Wraps the RAPL power calls in libmsr
  - Set RAPL power limits over a time-window (two windows)
    - set limit on socket, low, high, time-window
  - Collaboration with Barry Rountree (LLNL)

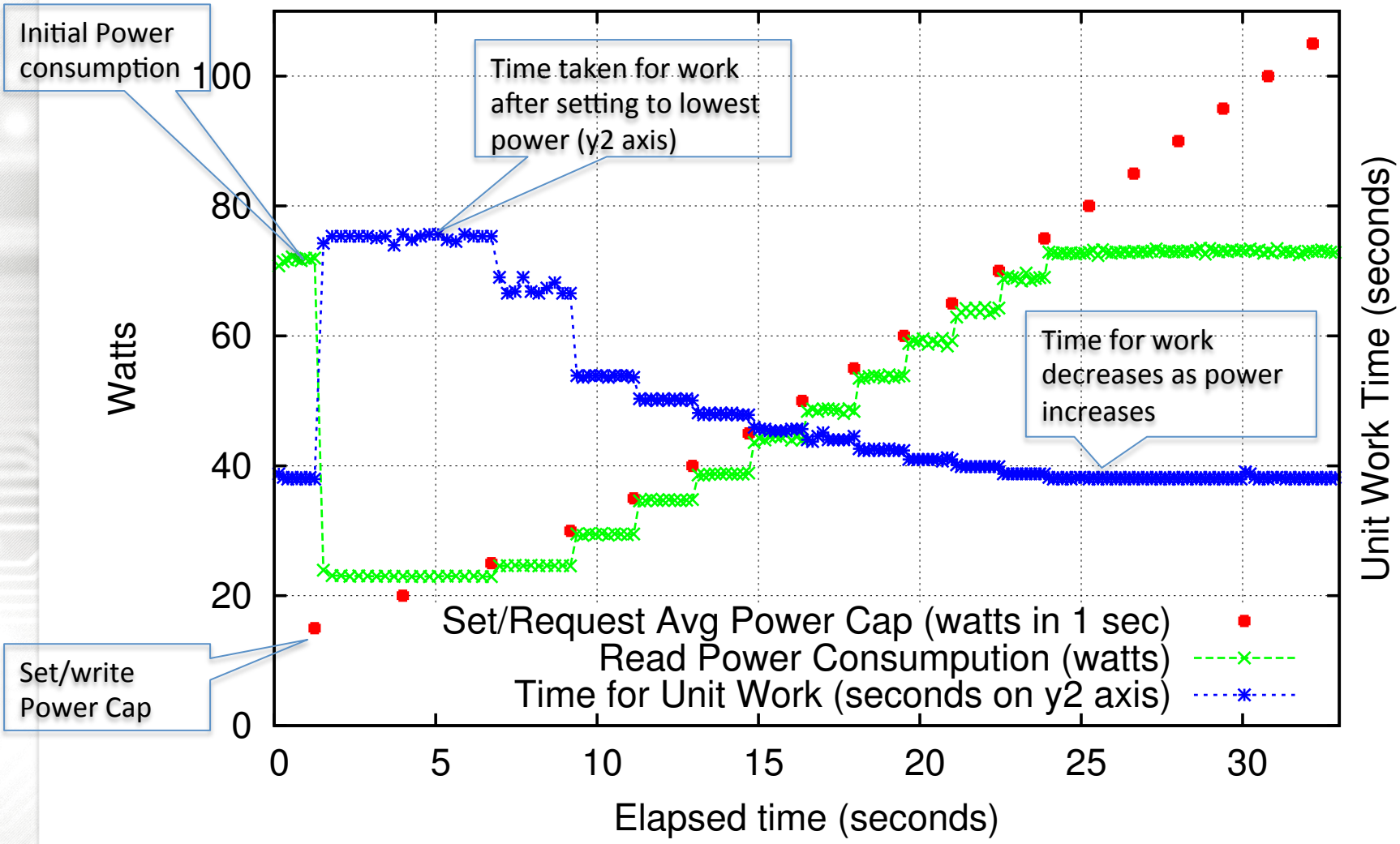
Controlling Power with PAPI-libmsr

Using PAPI libmsr component to read and set power caps
2x8 cores Xeon E5-2690 SandyBridge-EP at 2.9GHz



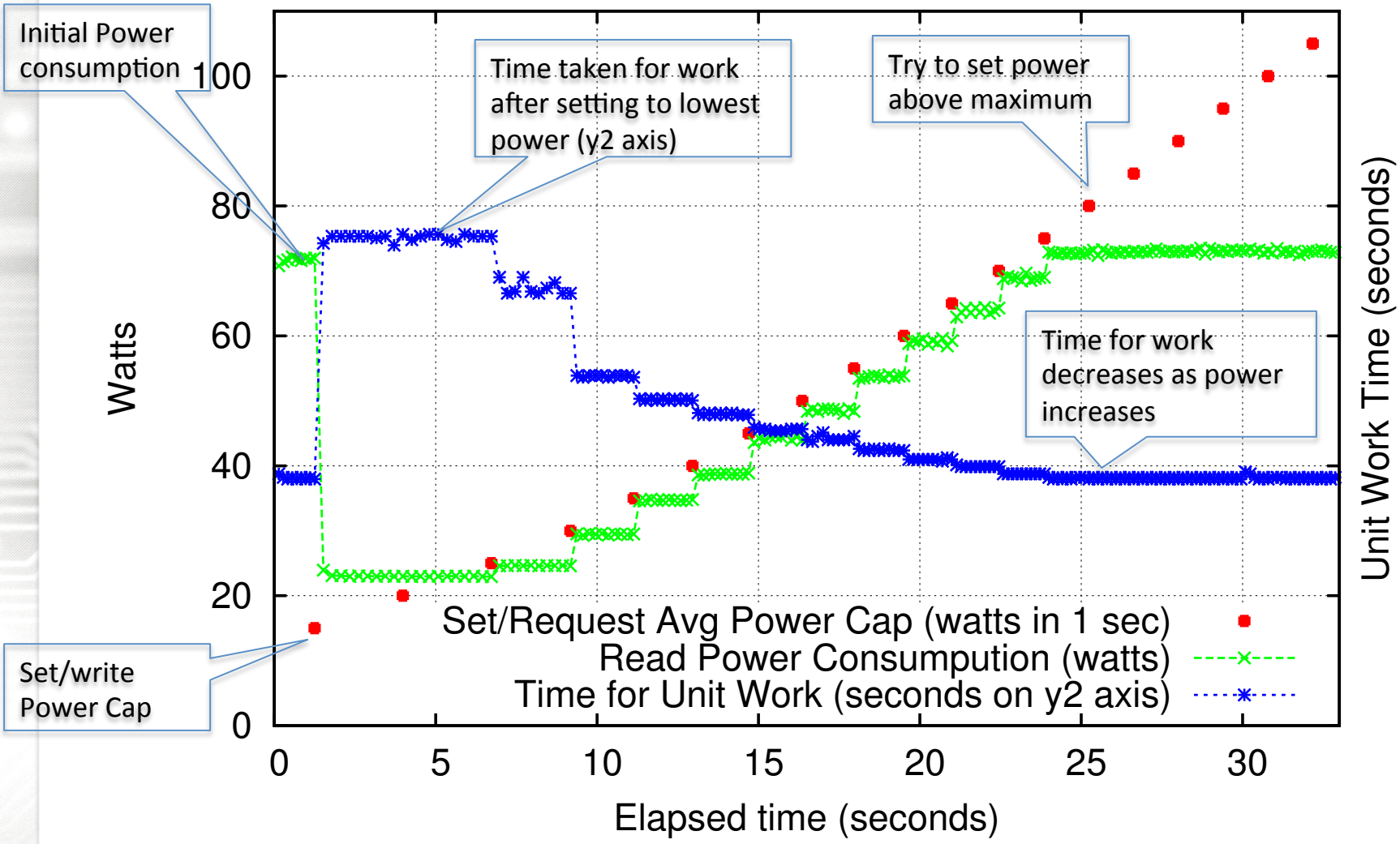
Controlling Power with PAPI-libmsr

Using PAPI libmsr component to read and set power caps
2x8 cores Xeon E5-2690 SandyBridge-EP at 2.9GHz



Controlling Power with PAPI-libmsr

Using PAPI libmsr component to read and set power caps
2x8 cores Xeon E5-2690 SandyBridge-EP at 2.9GHz

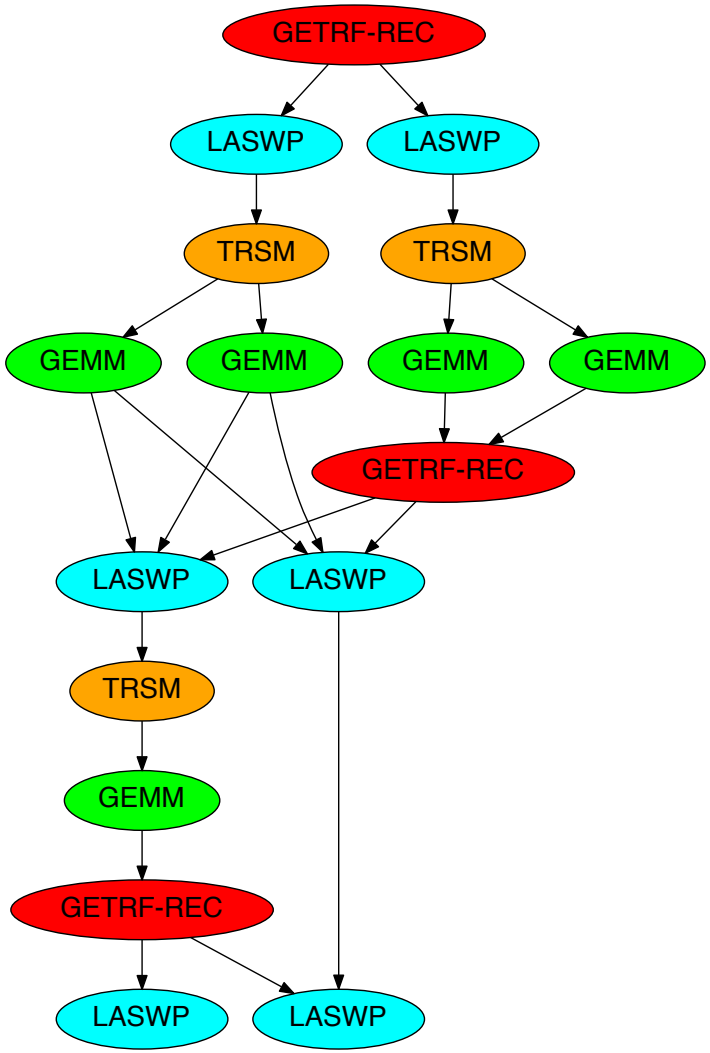


Usage Scenarios

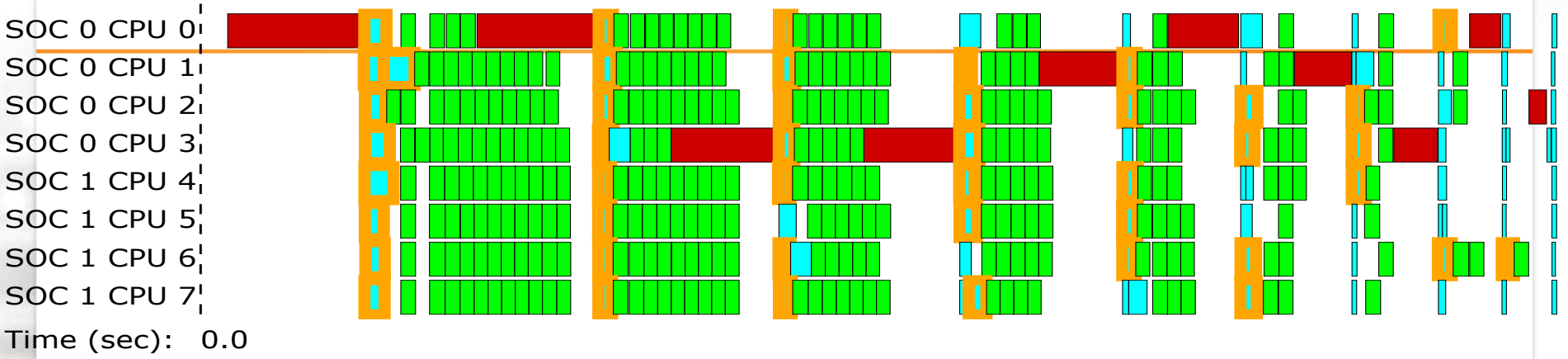
- Sample usage scenarios
  - If we know that computation requirements will decrease due to communication (I/O bound) and that the overall execution time will not suffer if the CPU power is capped temporarily.
  - We can schedule the critical path of the DAG on fast resources, decrease power consumption on sockets running the rest of the DAG.

LU factorization and DAG

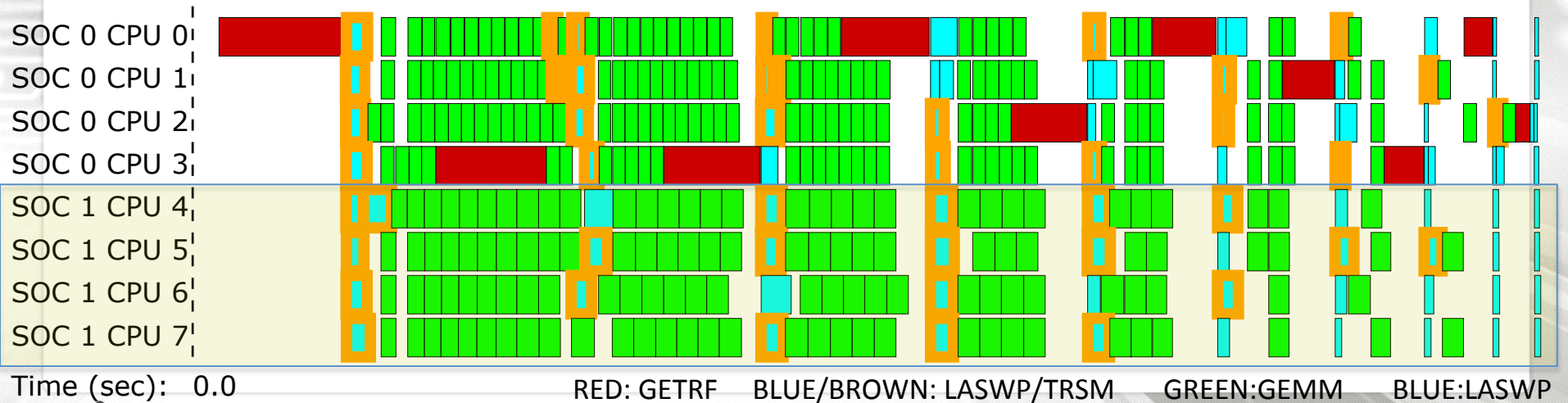
```
GETRF( A ) {
    for (k = 0; k < M; k++) {
        // panel factorization
        GETRF( A(k:M-1,k) );
        for (n = k+1; n < N; n++) {
            // row interchanges to the right
            LASWP ( A(k:M-1,n) );
            // triangular solve at the top
            TRSM ( A(k,n) );
            for (m = k+1; m < M; m++) {
                // matrix multiply to the righ
                gemm( A(m,k), A(k,n), A(m,n) )
            }
        }
    }
    for (k = 1; k < M; k++) {
        // row interchanges to the left
        for (n = 0; n < k-1; n++) {
            laswp( A(k:M-1,n) );
    }
}
```

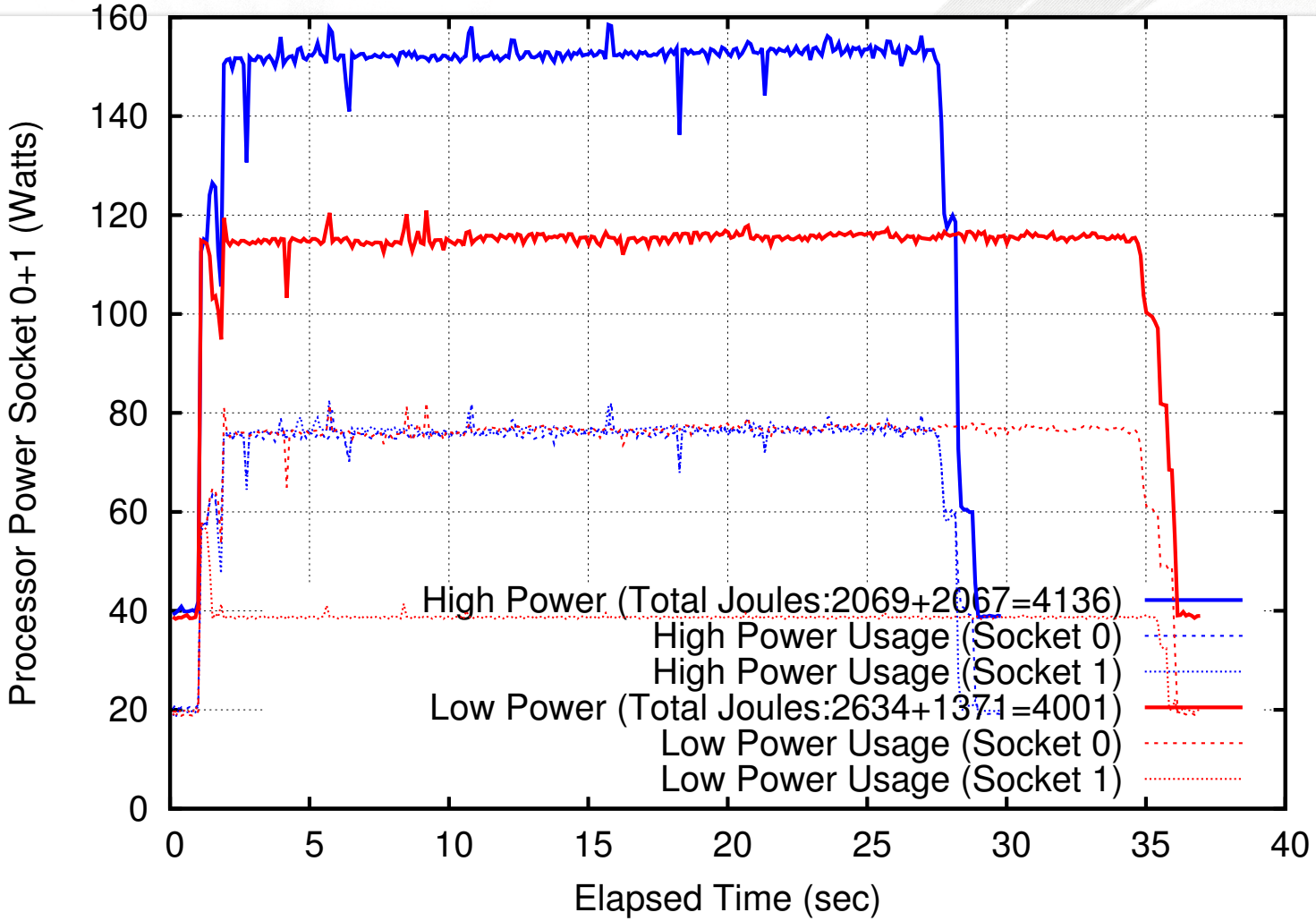


Both socket 0 and 1 running at full power. Note that the panel factorization (red) is long and on the critical path, so there is white space where no tasks can be run on socket 2.



Slow down socket 1 using RAPL and **lock critical path to socket 0**. The gemm tasks (green) take longer, filling out the white space on socket 2. This occurs without any overall loss in time for the full computation.



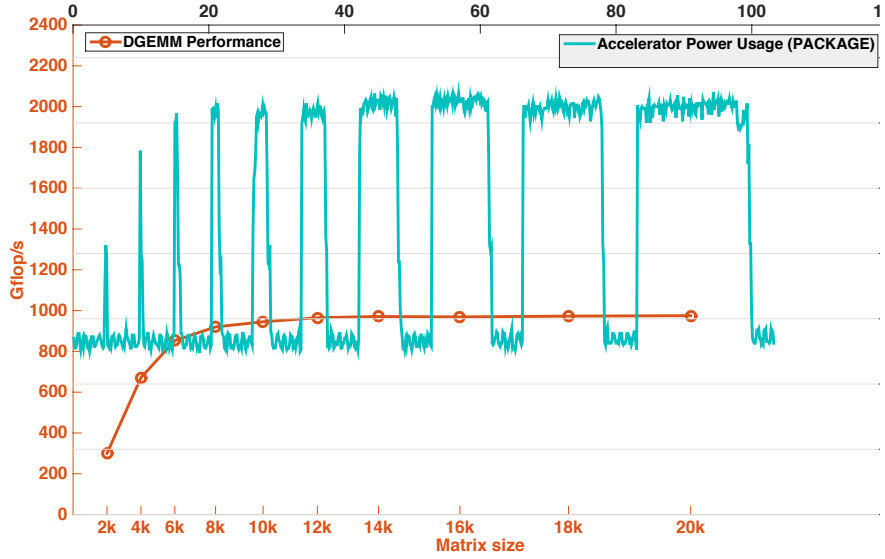


- Tiled LU (N=17920=224x80) using a SandyBridge EP (2 sockets, 4 cores/socket)
  - Demonstrating running the critical path at a higher speed than other tasks.
- High power: Both sockets running at full power.
- Low power: Second socket running a low power; critical path (panel) on first socket.
- Total energy used by processors in Joules: High 4136 Joules; Low: 4001 Joules

DGEMM Power & Performance: KNC vs. KNL

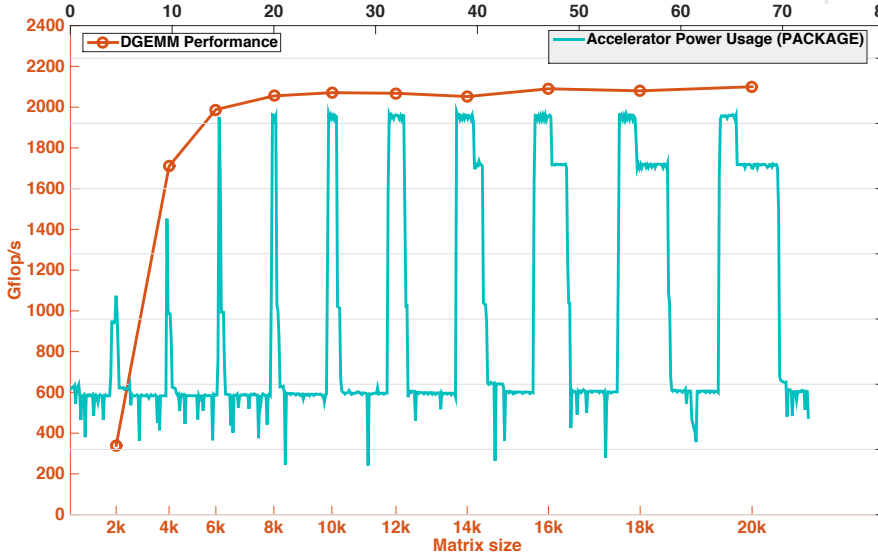
Knights Corner

60 cores (4 HW threads/core)



Knights Landing

68 cores (1 HW thread/core)



PAPI COUNTER INSPECTION TOOLKIT

Counter Inspection Toolkit

Define an "*accurate mapping*" between
high-level concepts of performance metrics and the underlying
low-level hardware events.

Benchmarks and analyses for:

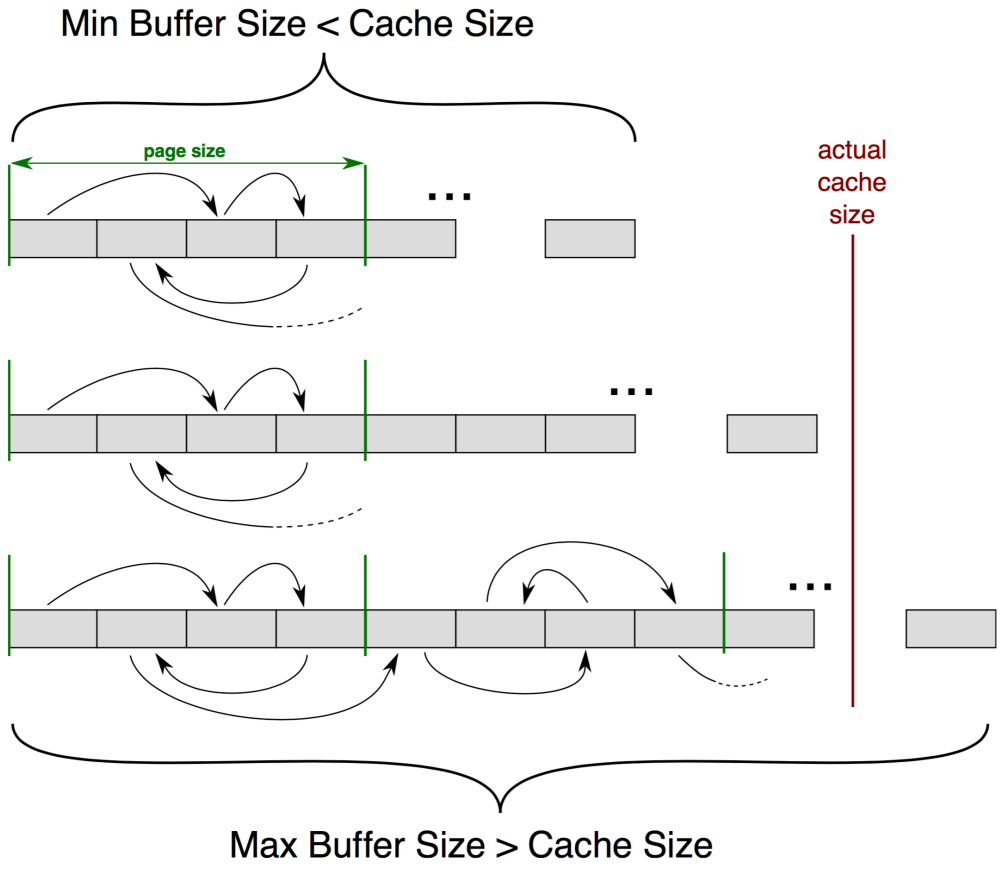
1. Validating native events

2. Defining high-level events ("pre-defined")

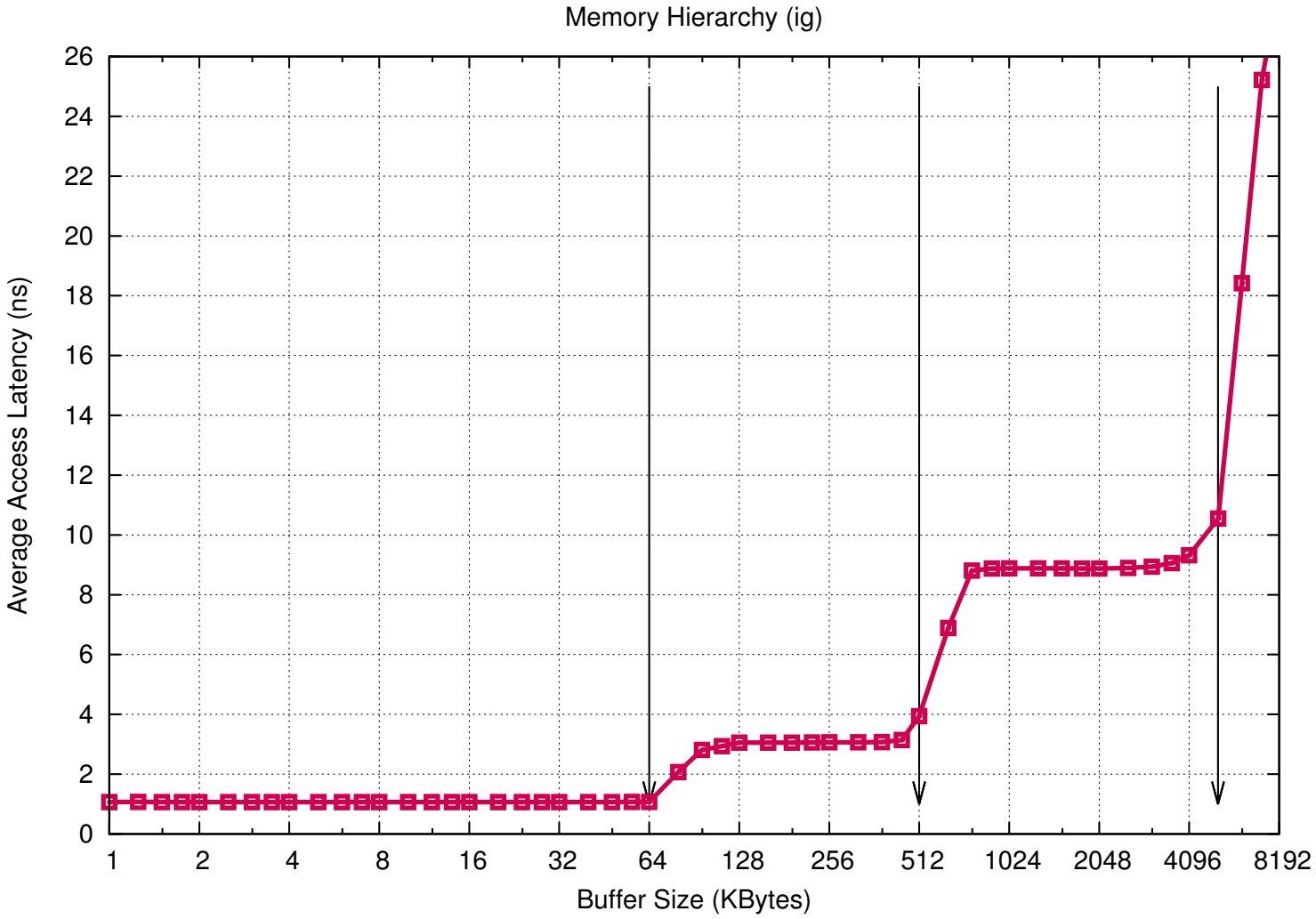
Random Pointer Chasing

```
SETUP( ) {
  p = (uintptr_t **) &array[0];
  for (i = random( ) ) {
     next = &array[i];
     *p = next;
     p = (uintptr_t **) next;
  }
}

MEASURE( ) {
  start_measurement();
  for (...) {
     p = (uintptr_t **) *p;
  }
  stop_measurement();
}
```



Benchmark timing



Defining high level events (Presets)

```
| LLC_MISSES                                                                  |
|            Alias for LAST_LEVEL_CACHE_MISSES                                |
-------------------------------------------------------------------------------
| LAST_LEVEL_CACHE_MISSES                                                     |
|            This is an alias for L3_LAT_CACHE:MISS                           |
-------------------------------------------------------------------------------
| L3_LAT_CACHE                                                                |
|            Core-originated cacheable demand requests to L3                  |
|     :MISS                                                                   |
|            Core-originated cacheable demand requests missed L3              |
|     :REFERENCE                                                              |
|            Core-originated cacheable demand requests that refer to L3       |
|     :e=0                                                                    |
|            edge level (may require counter-mask >= 1)                       |
|     :i=0                                                                    |
|            invert                                                           |
|     :c=0                                                                    |
|            counter-mask in range [0-255]                                    |
|     :t=0                                                                    |
|            measure any thread                                               |
|     :u=0                                                                    |
|            monitor at user level                                            |
|     :k=0                                                                    |
|            monitor at kernel level                                          |
```

PAPI 6 (a.k.a. PAPI-EX)

System-wide measurements:

- Shared hardware counter support is complex
  - limited vendor and kernel support

Counter inspection Toolkit:

- kernels that stress on-core + shared hardware features

Deeper Integration of PAPI for dataflow-based programming models

New Architectures:

- Xeon Phi Knights Landing, Cavium ThunderX, …

3rd Party Tools applying PAPI

- PaRSEC (UTK) http://icl.cs.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.cs.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