Introduction to Variorum

Module 1 of 2, ECP Lecture Series

6 August 2021 8:30AM-10:00AM PDT

20 August 2021 4:00PM-5:30PM PDT (Repeat)

Tapasya Patki, Aniruddha Marathe,
Stephanie Brink, and Barry Rountree

LLNL-PRES-825385

Lawrence Livermore National Laboratory

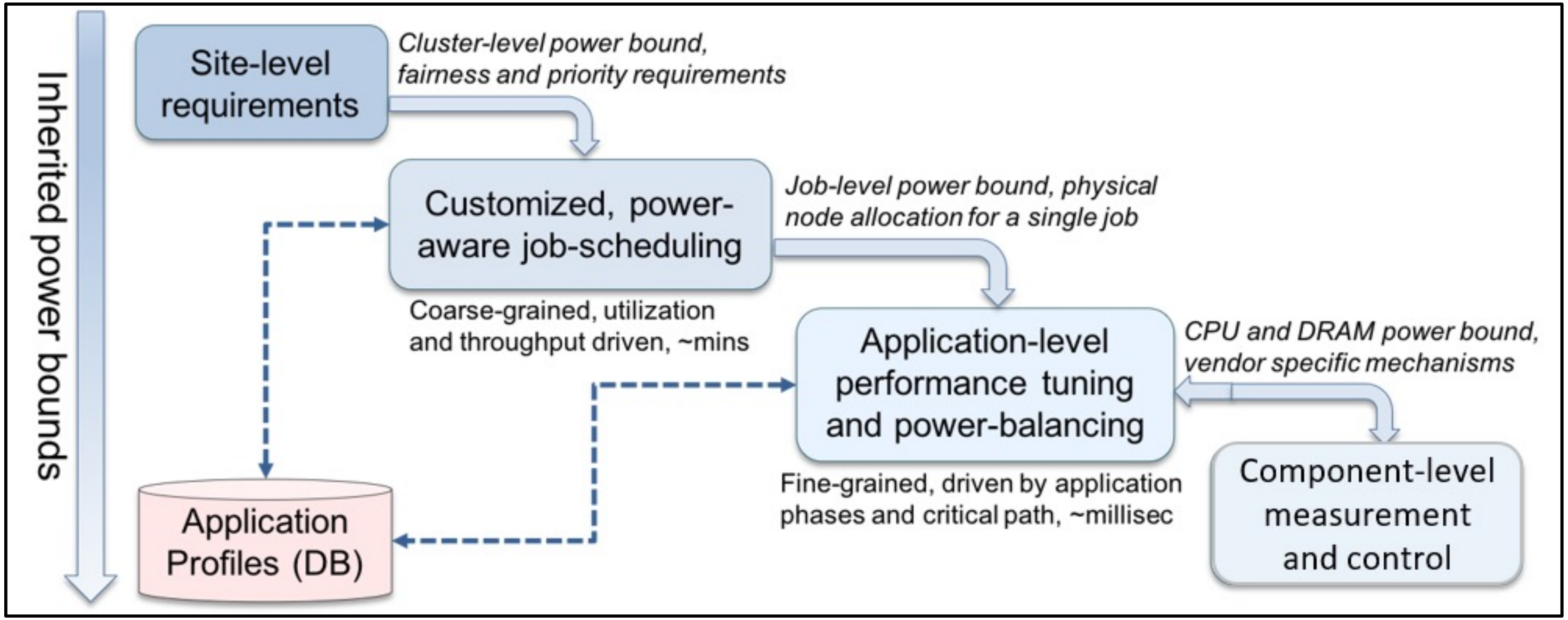
Module 1 Agenda

- Challenges in Power Management and the HPC Power Stack (20 min)
- Understanding Power Management Knobs on Intel, IBM, NVIDIA, ARM, and AMD Platforms (20 min)
- Variorum Library (40 min)
- Wrap Up (10 min)

Welcome: HPC Systems and Power Management

- Compute Nodes, I/O Nodes, Network
- Static and Dynamic Power Management
- DOE’s ongoing project: ECP Argo
  - Other synergistic projects: Flux, Caliper, Kokkos, GEOPM
  - Workflows: MuMMI, E3SM

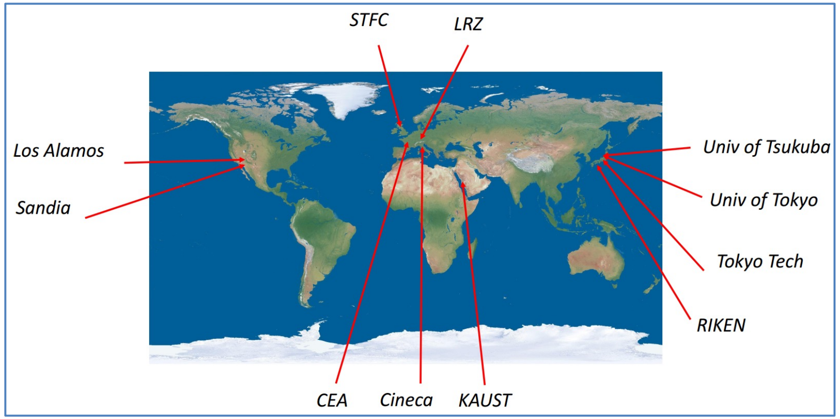
HPC PowerStack: Community Effort on System-wide, dynamic power management


Inherited power bounds
Site-level requirements
Cluster-level power bound, fairness and priority requirements
Customized, power-aware job-scheduling
Job-level power bound, physical node allocation for a single job
Coarse-grained, utilization and throughput driven, ~mins
Application-level performance tuning and power-balancing
CPU and DRAM power bound, vendor specific mechanisms
Fine-grained, driven by application phases and critical path, ~millisec
Component-level measurement and control
Application Profiles (DB)

https://hpcpowerstack.github.io/

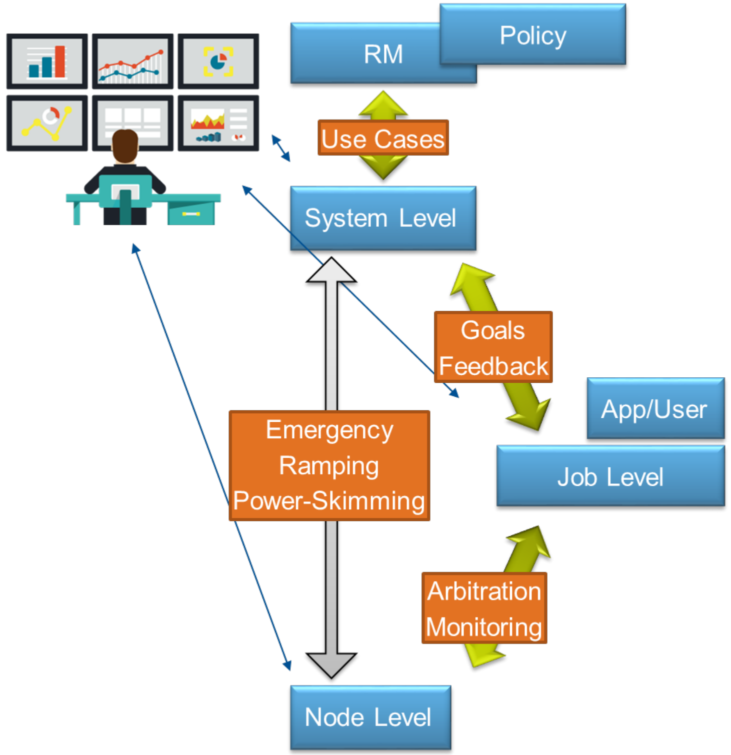
PowerStack: Stakeholders

- Current industry collaborators: Intel, IBM, AMD, ARM, NVIDIA, Cray/HPE, Fujitsu, Altair, ATOS/Bull, and PowerAPI community standard
- Multiple academic and research collaborators across Europe, Asia, US
- Three working groups established
- Dynamic power management at all levels, along with prioritization of the critical path, application performance and throughput
- One of the prototypes developed as part of ECP using SLURM, GEOPM, Variorum/msr-safe (close collaboration with Intel)
- Additional software with Flux and Variorum underway



EEHPC-WG's insight into sites investing in Energy- and Power-aware Job Scheduling and Resource Management (EPA-JSRM)

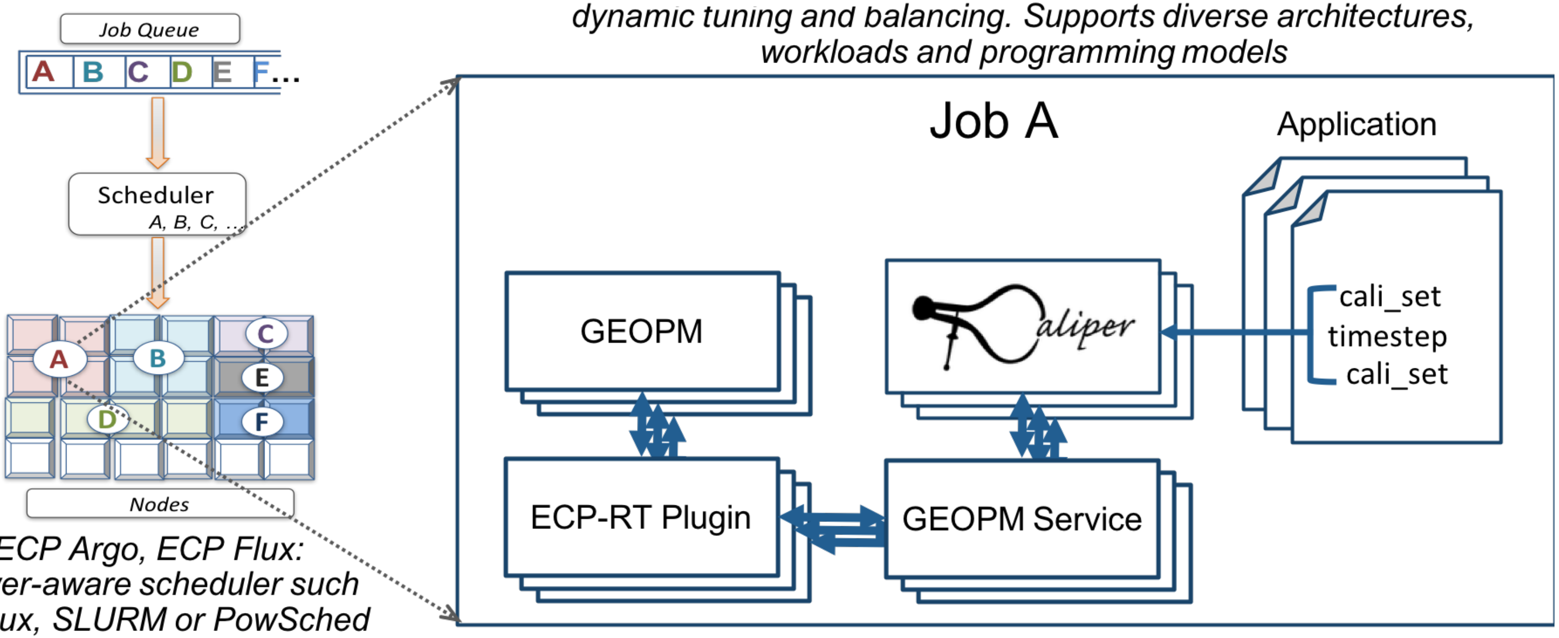
PowerStack: Layers



- Site-level
  - Policies, objectives
- System-level
  - Scheduling, runtimes, application-awareness
- Platform-level
  - Hardware interaction, PowerAPI
- Idle and Emergency Management

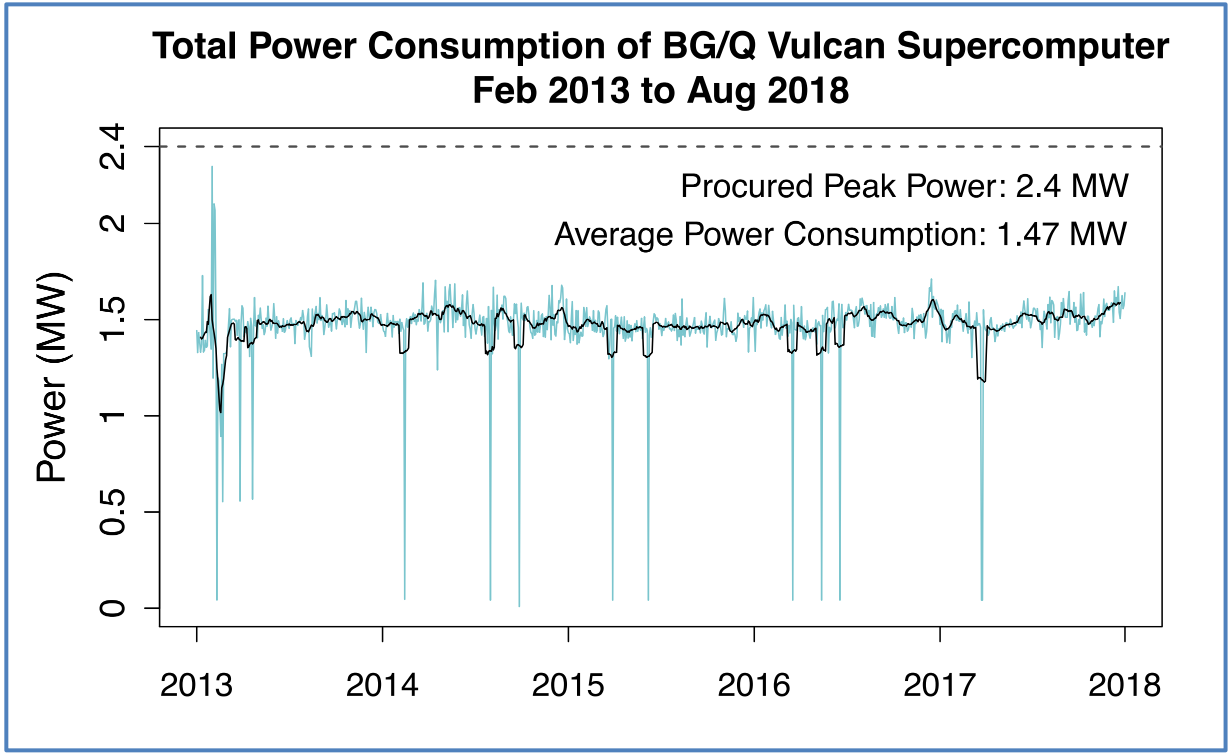
ECP Argo uses SLURM/Flux, GEOPM and Variorum as vehicles for power management at Exascale

ECP Argo: Job-level runtime system for fine-grained, dynamic tuning and balancing. Supports diverse architectures, workloads and programming models



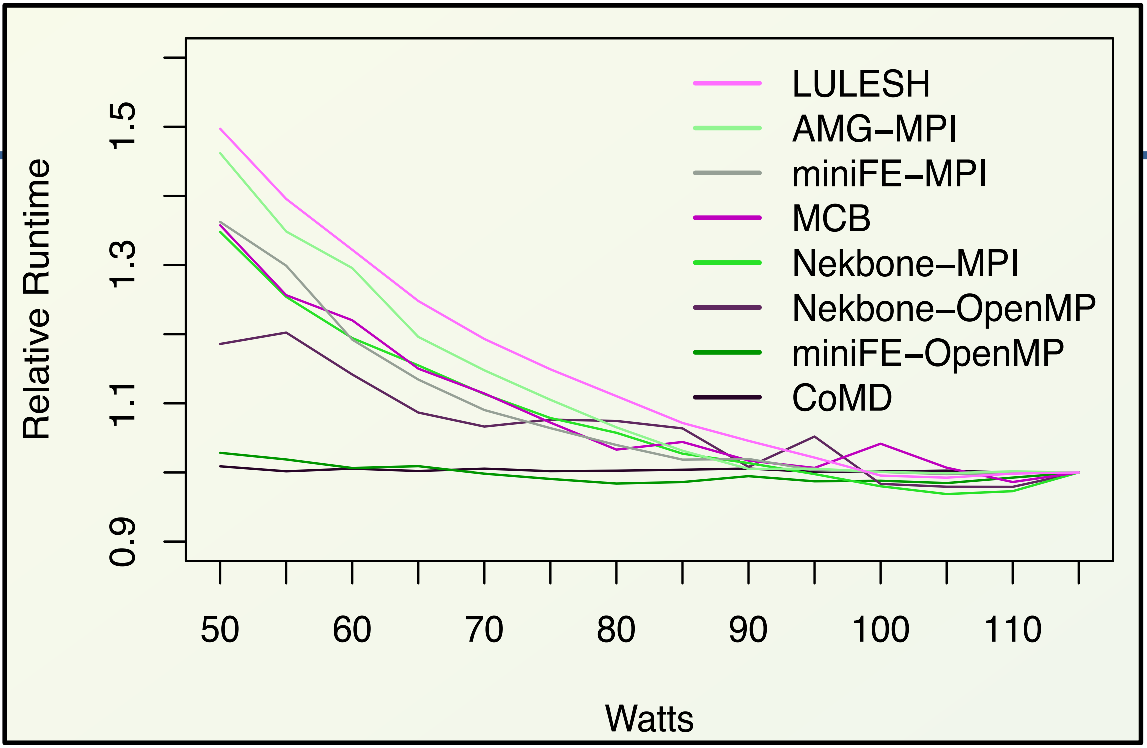
ECP Argo, ECP Flux: Power-aware scheduler such as Flux, SLURM or PowSched (low-overhead static decisions)

Unused Power: 40%



Why is it so?

- Applications have different memory, communication, I/O requirements and phase behaviors
- Applications don't utilize all of the allocated power, thus allocating more power doesn't always improve performance

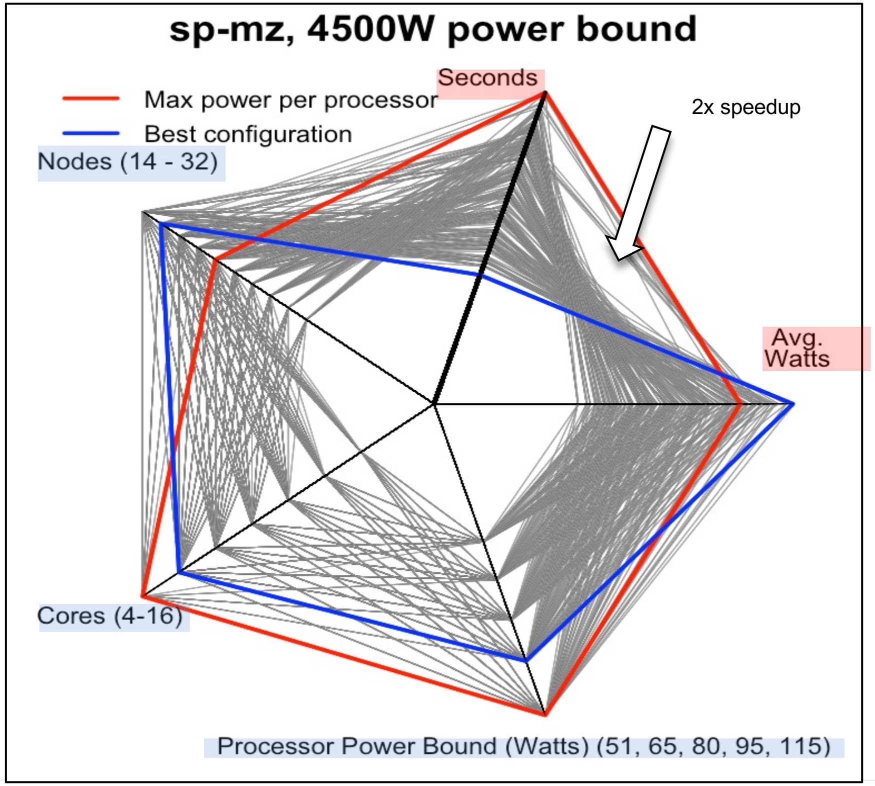


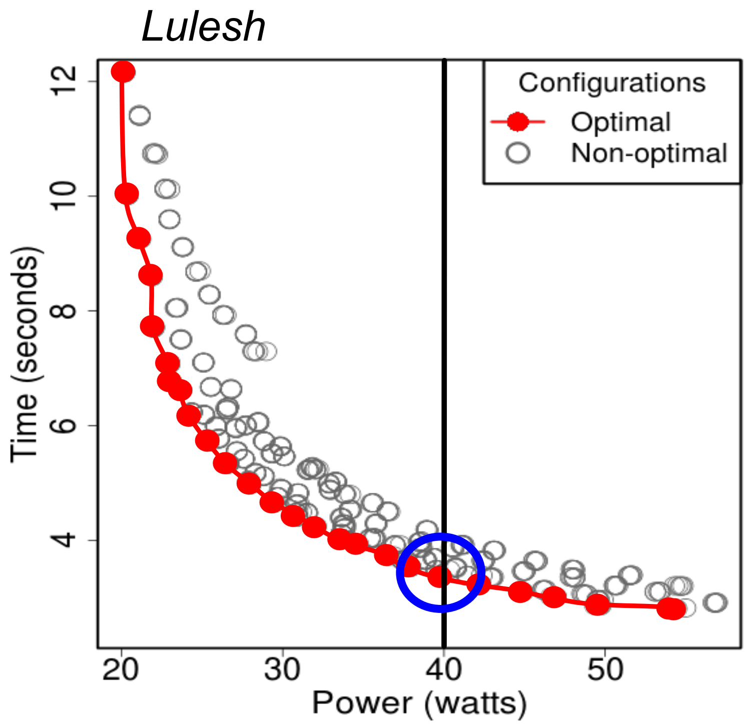
Intel Sandy Bridge, 8 nodes, (2 sockets, 8 cores)

Min: 51 W, Max: 115 W

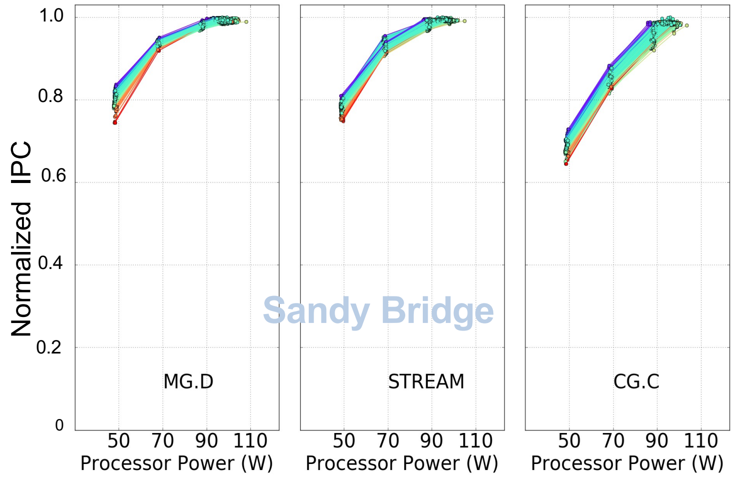
Graph Source: Daniel Ellsworth, Colorado College

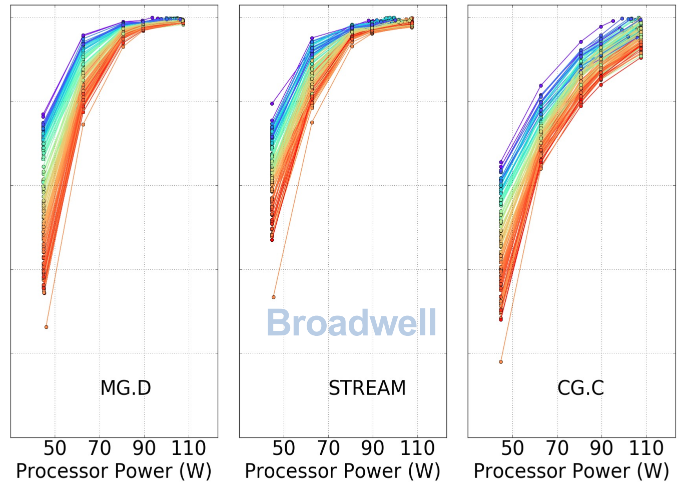
Automatic configuration selection is crucial for performance and energy efficiency





Significant performance variability can result from processor manufacturing process

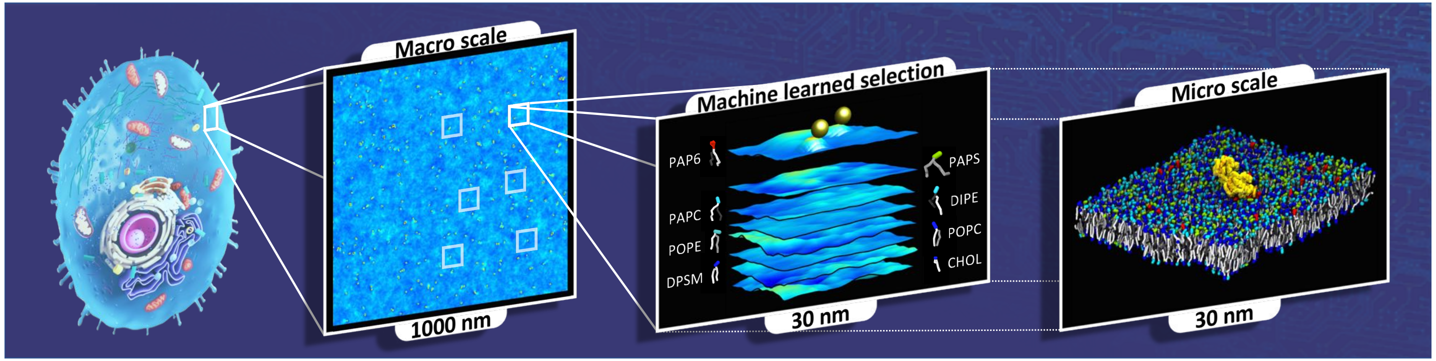




- Manufacturing variability in hardware continues to increase, and will worsen with heterogeneous systems
- 4x difference between two generations of Intel processors, needs advanced runtime options for mitigation

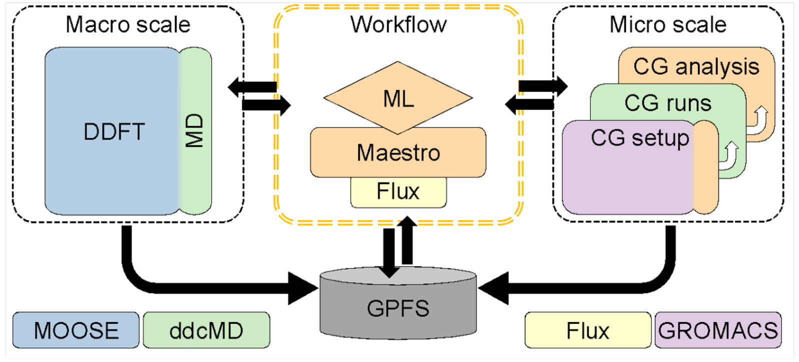
Demo of Application Performance and Processor Variability

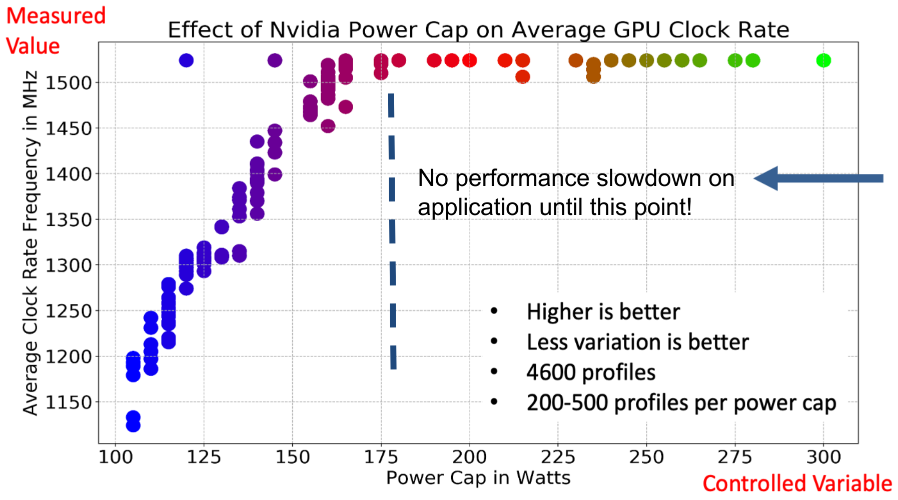
Power Management of workflows is necessary: Example of the Multiscale Machine-Learned Modeling Infrastructure (MuMMI)



- MuMMI is designed to understand cancer-signaling mechanisms using multi-scale machine learning.
- Based on the mutated RAS protein, which is responsible for **a third of all human cancers**
- Understanding the role of RAS requires exploration at multiple length- and time-scales
- The MuMMI workflow makes significant use of GPUs on the Lassen/Sierra supercomputers

Power Management Workflow Example: Significant Cost Savings with Active GPU Power Management on MuMMI Workflow

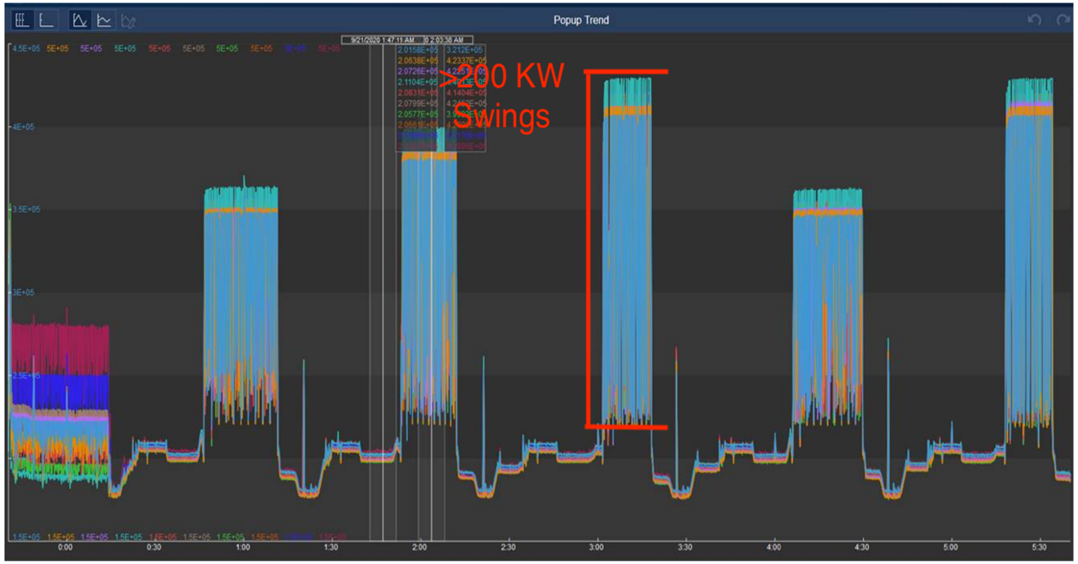




- ddcMD is one of the main compute and GPU-oriented components of MuMMI Workflow
- Cluster wide savings of 382kW with no performance slowdown with GPU power management!
- 254.6 kWh energy savings, $43k cost savings per day @ 7c per kWh

Patki et al., Comparing GPU Power and Frequency Capping: A Case Study with the MuMMI Workflow, WORKS'19 (SC)

Facilities Perspective: Mitigating Power Swings on Sierra/Lassen with in-depth application analysis with Variorum



*Example: LBANN on **Sierra** at full scale has significant fluctuations impacting LLNL's electrical grid -- workload swings expected to worsen at exascale*

- Livermore Big Artificial Neural Network toolkit (LBANN) -- infrastructure used for deep learning in HPC
- LBANN utilizes all 4 GPUs per node
- Data shows 3 minute samples over 6 hours on Sierra with >200 KW swings
- Other workflows have similar trends with power fluctuations at scale
- Mitigation of power fluctuations is required to avoid electrical supply disruption
- Variorum + Flux can dynamically analyze applications and prevent future fluctuations

So, how do we even manage power? (measurement vs control)

Two primary mechanisms in hardware for control:

- Dynamic voltage and frequency scaling (ARM)
- Power capping, such as RAPL (Intel, AMD) or OPAL (IBM) or NVML (NVIDIA)

- Automatic tuning through Intel Turbo Boost or IBM UltraTurbo

- ACPI defines P-states, which are voltage/frequency pairs
- DVFS: `cpufreq`, ‘`userspace`’ governor
- RAPL:
  - Low-level register programming, supported with `msr` kernel module along with `msr-tools`
  - LLNL’s `msr-safe` and `variorum` provide safe and clean access from user space across architectures
- OPAL:
  - Firmware layer providing in band monitoring with sensors and out of band power capping

Introduction to Intel’s RAPL

- Intel provides programmable, machine-specific registers for power, energy and thermal management (power capping)

- MSR Domains for server architectures:
  - Package – represents processor cores, caches, and other things on the socket
  - DRAM – represents memory components
  - Other uncore registers also exist

Intel SDM: Vol 3, Chapter 14.9, https://software.intel.com/en-us/articles/intel-sdm

Deep dive into the `MSR_PKG_POWER_LIMIT, (0x610h,rw)`

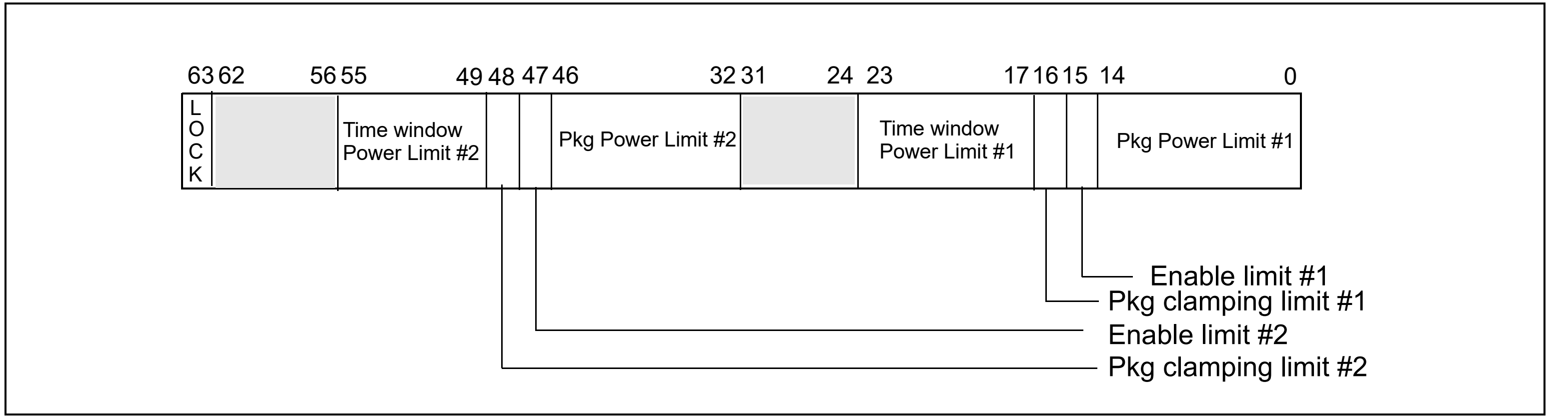


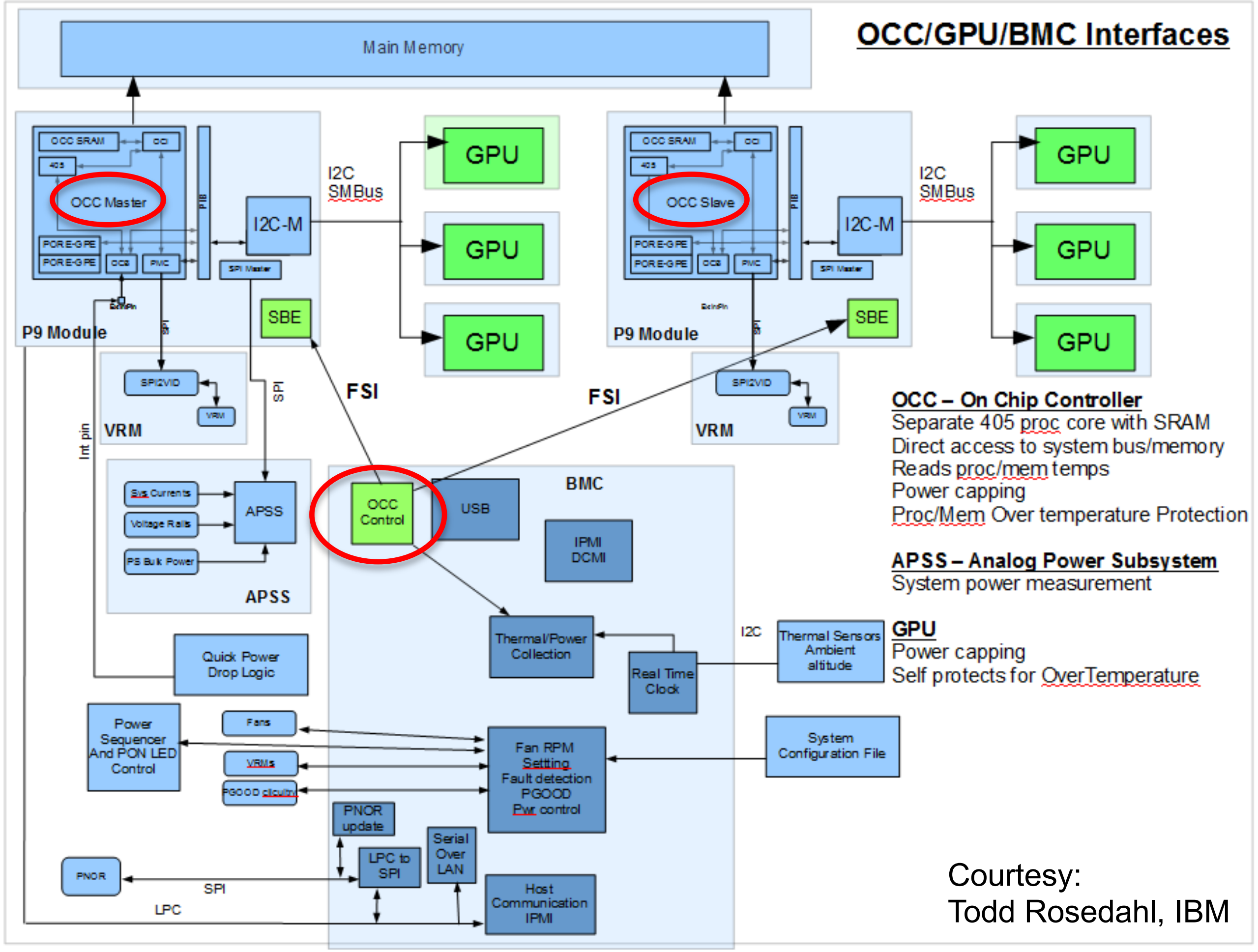
Figure 14-32. MSR_PKG_POWER_LIMIT Register

- Pkg Power Limit#1, bits 14:0, sets average power limit corresponding for window 1, unit 0.125 W
- Enable Limit#1: bit 15, 0 = disabled
- Pkg Clamping Limit#1, bit 16, allow going below OS requested P/T state
- Time Window Limit#1, bits 23:17, minimum is typically 1 millisec
- Pkg Limit #2 is typically not used

Demo of `msr-tools` and introduction to `msr-safe`, `libmsr`

Power9 OCCs have a Primary-Secondary Design

- OCC collects thermal data every 250us
- Power knobs are exposed with the Open Power Abstraction Layer (OPAL)
- When power button is pressed, BMC selects master chip and releases SBEs, OPAL firmware is loaded



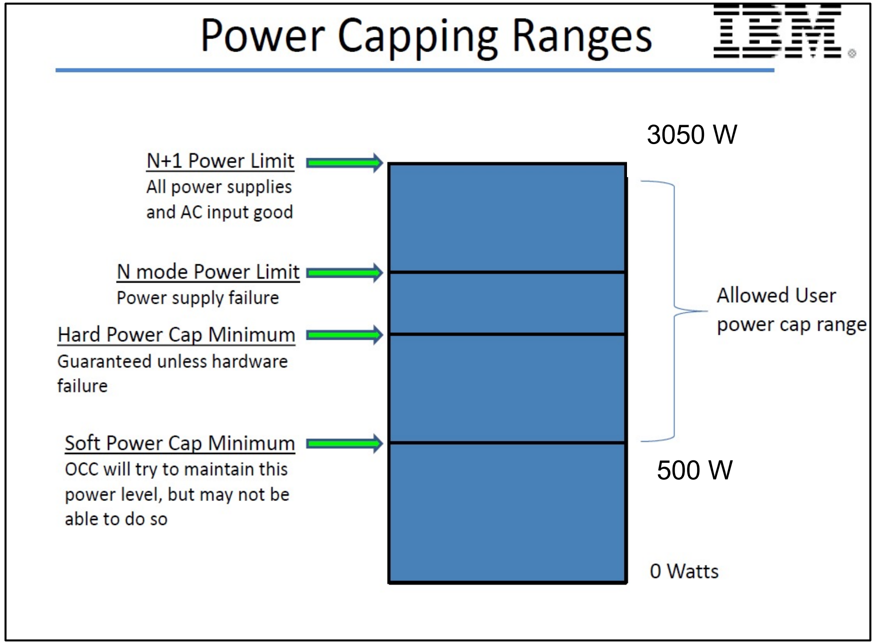
OCC – On Chip Controller
Separate 405 proc core with SRAM
Direct access to system bus/memory
Reads proc/mem temps
Power capping
Proc/Mem Over temperature Protection

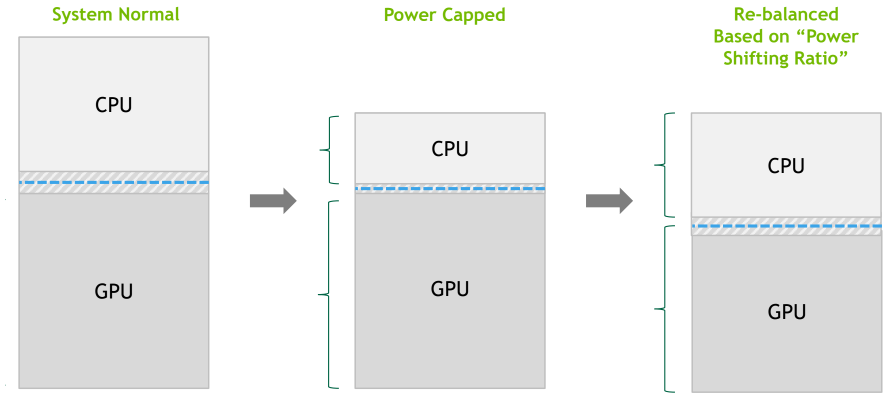
APSS – Analog Power Subsystem
System power measurement

GPU
Power capping
Self protects for OverTemperature

Courtesy:
Todd Rosedahl, IBM

Capping Node and GPU power


Power Capping Ranges
IBM
3050 W
N+1 Power Limit
All power supplies and AC input good
N mode Power Limit
Power supply failure
Hard Power Cap Minimum
Guaranteed unless hardware failure
Soft Power Cap Minimum
OCC will try to maintain this power level, but may not be able to do so
Allowed User power cap range
500 W
0 Watts



- Power-shifting ratio determines distribution between CPU and GPU
- Per-socket cap and memory cap cannot be specified, master OCC has control

OPAL interfaces for in-band sensors and power capping

Requires OPAL firmware v6.0.18+ – includes fix for a firmware bug we found based on soft power cap

Read-only access:

- `/sys/firmware/opal/exports/occ_inband_sensors`
  - Over 336 sensors reported on `alehouse`
  - Including power, temperature, frequencies for CPU, Memory, GPU (1ms granularity)

Read/write access:

- `/sys/firmware/opal/powercap/system-powercap/powercap-current`
- `/sys/firmware/opal/psr/cpu_to_gpu_*` (per socket)

AMD power management can be accessed with E-SMI Library, Energy Driver, and Host System Management Port (HSMP) module

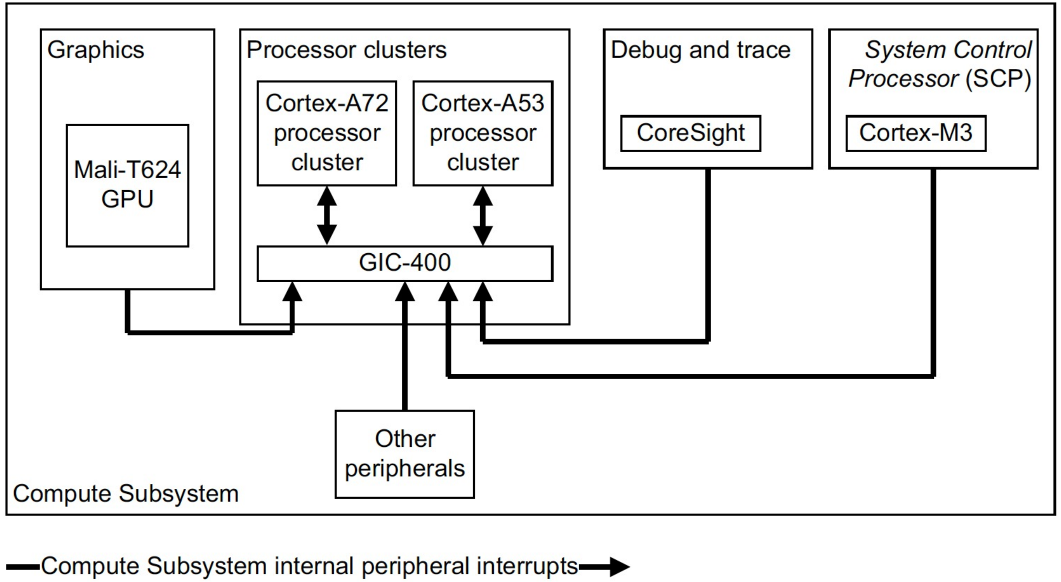
- EPYC™ System Management Interface In-band Library (E-SMI library) is available at: https://github.com/amd/esmi_ib_library
- AMD Energy Driver allows access for core and socket energy counters through MSRs and RAPL (with hwmon), available at: https://github.com/amd/amd_energy
  - Power, Energy and Time Units MSR_RAPL_POWER_UNIT/ C001_0299: shared with all cores in the socket
  - Energy consumed by each Core MSR_CORE_ENERGY_STATUS/ C001_029A: 32-bitRO, Accumulator, core-level power reporting
  - Energy consumed by Socket MSR_PACKAGE_ENERGY_STATUS/ C001_029B: 32-bitRO, Accumulator, socket-level power reporting, shared with all cores in socket
  - These registers are updated every 1 ms and cleared on reset of the system.
- HSMP driver for power metrics is available at: (https://github.com/amd/amd_hsmp)
  - Allows for power capping, setting of boost limits and PCIe access.
  - Internal mechanism uses a mailbox approach for register access
- Structure of SysFS interface (/sys/devices/system/cpu/) includes CPU, Socket and general system management (does not include GPU management, which is provided through `rocm-smi`)

AMD Power Control Knobs are exposed through the Host System Management Port (HSMP) kernel module

- SysFS structure:

amd_hsmp/cpuX/	Directory for each possible CPU
`boost_limit`	(RW) HSMP boost limit for the core in MHz
amd_hsmp/socketX/	Directory for each possible socket
`boost_limit`	(WO) Set HSMP boost limit for the socket in MHz
`c0_residency`	(RO) Average % all cores are in C0 state
`cclk_limit`	(RO) Most restrictive core clock (CCLK) limit in MHz
`fabric_clocks`	(RO) Data fabric (FCLK) and memory (MCLK) in MHz
`fabric_pstate`	(WO) Set data fabric P-state, -1 for autonomous
`power`	(RO) Average socket power in milliwatts
`power_limit`	(RW) Socket power limit in milliwatts
`power_limit_max`	(RO) Maximum possible value for power limit in mW
`proc_hot`	(RO) Socket PROC_HOT status (1 = active, 0 = inactive)
`tctl`	(RO) Thermal Control value (not temperature)

ARM Juno r2 : SoC Architecture


Graphics
Mali-T624 GPU
Processor clusters
Cortex-A72 processor cluster
Cortex-A53 processor cluster
GIC-400
Debug and trace
CoreSight
System Control Processor (SCP)
Cortex-M3
Other peripherals
Compute Subsystem
Compute Subsystem internal peripheral interrupts

- Cortex-A72 MP2 cluster (r0p0eac)
  - Dual cluster, SMP configuration
  - Overdrive 1.2GHz speed
- Cortex-A53 MP4 cluster (r0p3)
  - Quad cluster, SMP configuration
  - Overdrive 950MHz speed
- Quad Core MALI T624 r1p0
  - Nominal 600MHz operating speed
  - Caches: L2 128KB
- Control and telemetry
  - DVFS and power gating via SCP
  - 4 energy meters
  - Temperature, clocks telemetry

ARM Juno r2 : System Interface

- Monitoring and control through Sysfs interface[1]
  - Exposed through the Linux HWMon interface

Power telemetry (mW)
Location: /sys/class/hwmon/hwmon0/

SYS_POW_SYS : power1_input
SYS_POW_A72 : power2_input
SYS_POW_A53 : power3_input
SYS_POW_GPU : power4_input

Thermal telemetry (C)
Location: /sys/class/hwmon/hwmon0/

SoC temperature: temp1_input
big temperature: temp2_input
LITTLE temperature: temp3_input
GPU temperature: temp4_input

Clocks telemetry (kHz)
Location: /sys/devices/system/cpu/cpufreq/

big clocks: policy0/scaling_cur_freq
LITTLE clocks: policy1/scaling_cur_freq

Frequency control (kHz)
Location: /sys/devices/system/cpu/cpufreq/

big clocks: policy0/scaling_setspeed
LITTLE clocks: policy1/scaling_setspeed

1. https://community.arm.com/developer/tools-software/oss-platforms/w/docs/482/energy-monitoring-on-juno

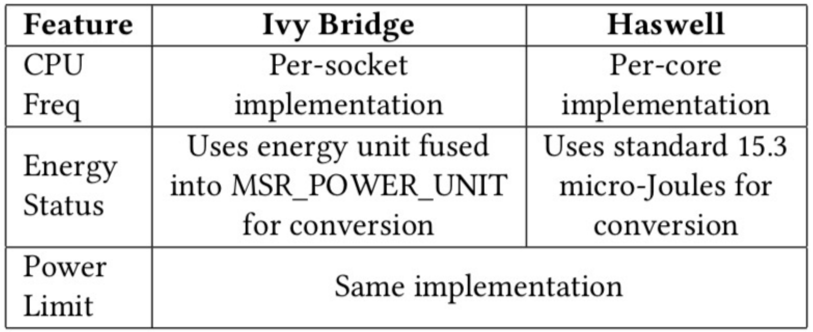
NVML[1]: Nvidia Measurement Interface

- Power usage: nvmlDeviceGetPowerUsage (device ID, &power)
  - Returns power usage of the target GPU device in watts
- Power limit: nvmlDeviceGetPowerManagementLimit(device ID, &powerlimit)
  - Retrieves the power management limit associated with this device in watts
- Temperature: nvmlDeviceGetTemperature (device ID, NVML_GPU, &temp)
  - Returns temperature of the target GPU device in degree Celsius
- Clocks: nvmlDeviceGetClock (device ID, NVML_CLOCK_SM, &clocks)
  - Returns clock speed of the target GPU device in MHz
- GPU utilization: nvmlDeviceGetUtilizationRates(device ID, &utilization)
  - Instantaneous utilization rate for the target GPU device

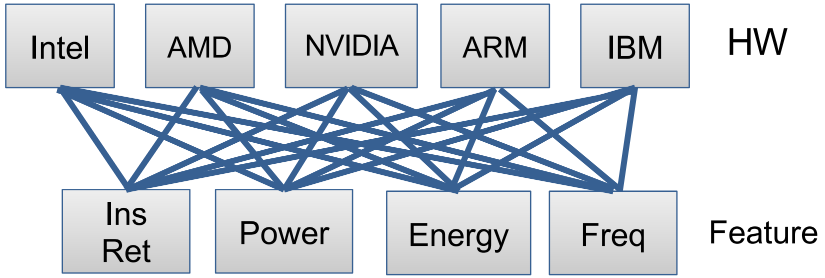
1. https://docs.nvidia.com/deploy/nvml-api/index.html

Power management capabilities differ across vendors

- libmsr (~2012) has been quite successful in the community
  - Provided simpler monitor/control interfaces translating bit fields into 64-bit MSRs
  - But, was Intel-specific
- Interfaces, domains, latency, capabilities
  - But, may also vary across generations within the same vendor
- Goals of variorum:
  - Target 95% of users (friendly APIs)
  - More devices
    - i.e., CPUs, Accelerators, IPMI, PCIe (CSRs, MMIO)
  - More platforms
    - i.e., Intel Skylake, IBM P9, NVIDIA Volta, AMD Epyc, ARM
  - More hardware knobs and controls



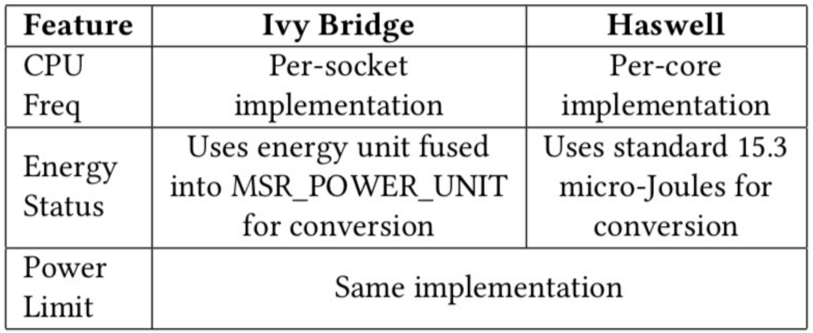
Feature	Ivy Bridge	Haswell
CPU Freq	Per-socket implementation	Per-core implementation
Energy Status	Uses energy unit fused into MSR_POWER_UNIT for conversion	Uses standard 15.3 micro-Joules for conversion
Power Limit	Same implementation	



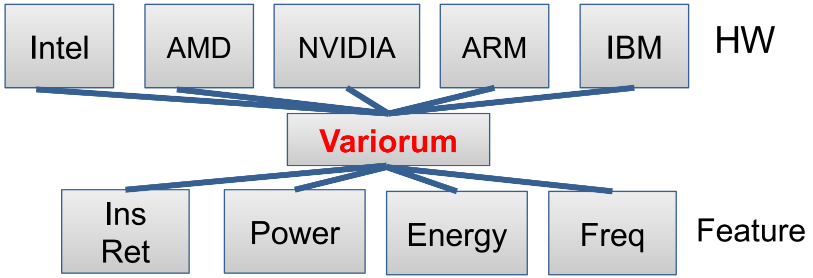
github.com/llnl/libmsr

Power management capabilities differ across vendors

- libmsr (~2012) has been quite successful in the community
  - Provided simpler monitor/control interfaces translating bit fields into 64-bit MSRs
  - But, was Intel-specific
- Interfaces, domains, latency, capabilities
  - But, may also vary across generations within the same vendor
- Goals of variorum:
  - Target 95% of users (friendly APIs)
  - More devices
    - i.e., CPUs, Accelerators, IPMI, PCIe (CSRs, MMIO)
  - More platforms
    - i.e., Intel Skylake, IBM P9, NVIDIA Volta, AMD Epyc, ARM
  - More hardware knobs and controls



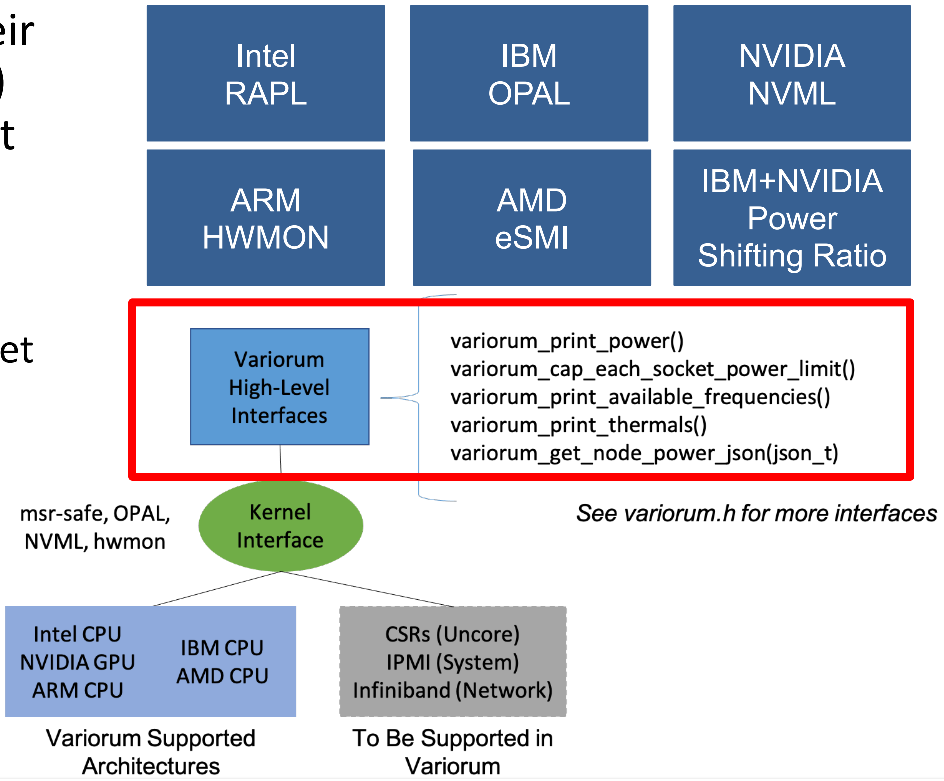
Feature	Ivy Bridge	Haswell
CPU Freq	Per-socket implementation	Per-core implementation
Energy Status	Uses energy unit fused into MSR_POWER_UNIT for conversion	Uses standard 15.3 micro-Joules for conversion
Power Limit	Same implementation	



github.com/llnl/libmsr

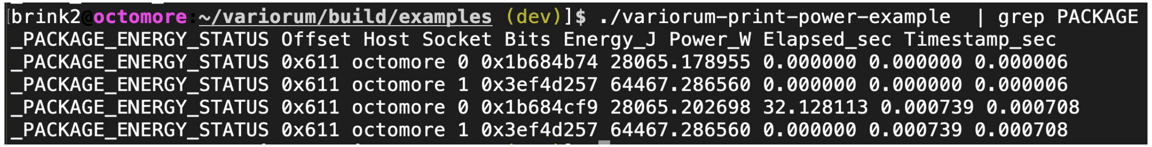
Variorum: Vendor-neutral user space library for power management

- Power management capabilities (and their interfaces, domains, latency, capabilities) widely differ from one vendor to the next
- Variorum: Platform-agnostic vendor-neutral, simple front-facing APIs
  - Evolved from *libmsr*, and designed to target several platforms and architectures
  - Abstract away tedious and chaotic details of low-level knobs
  - Implemented in C, with function pointers to specific target architecture
  - Integration with higher-level power management software through JSON

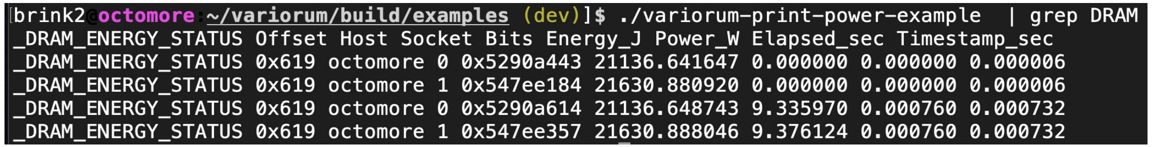

Intel RAPL
IBM OPAL
NVIDIA NVML
ARM HWMON
AMD eSMI
IBM+NVIDIA Power Shifting Ratio
Variorum High-Level Interfaces
variorum_print_power()
variorum_cap_each_socket_power_limit()
variorum_print_available_frequencies()
variorum_print_thermals()
variorum_get_node_power_json(json_t)
See variorum.h for more interfaces
msr-safe, OPAL, NVML, hwmon
Kernel Interface
Intel CPU
NVIDIA GPU
ARM CPU
IBM CPU
AMD CPU
CSRs (Uncore)
IPMI (System)
Infiniband (Network)
Variorum Supported Architectures
To Be Supported in Variorum

Demo of Variorum Build and APIs

Variorum Examples: Source code annotations as well as non-intrusive monitoring are supported

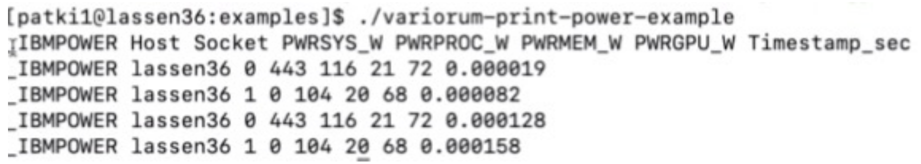


```
[brink2@octomore:~/variorum/build/examples (dev)]$ ./variorum-print-power-example  | grep PACKAGE
_PACKAGE_ENERGY_STATUS Offset Host Socket Bits Energy_J Power_W Elapsed_sec Timestamp_sec
_PACKAGE_ENERGY_STATUS 0x611 octomore 0 0x1b684b74 28065.178955 0.000000 0.000000 0.000006
_PACKAGE_ENERGY_STATUS 0x611 octomore 1 0x3ef4d257 64467.286560 0.000000 0.000000 0.000006
_PACKAGE_ENERGY_STATUS 0x611 octomore 0 0x1b684cf9 28065.202698 32.128113 0.000739 0.000708
_PACKAGE_ENERGY_STATUS 0x611 octomore 1 0x3ef4d257 64467.286560 0.000000 0.000739 0.000708
```



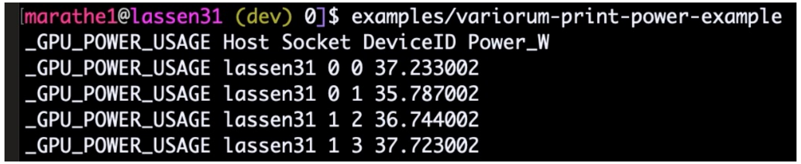
```
[brink2@octomore:~/variorum/build/examples (dev)]$ ./variorum-print-power-example  | grep DRAM
_DRAM_ENERGY_STATUS Offset Host Socket Bits Energy_J Power_W Elapsed_sec Timestamp_sec
_DRAM_ENERGY_STATUS 0x619 octomore 0 0x5290a443 21136.641647 0.000000 0.000000 0.000006
_DRAM_ENERGY_STATUS 0x619 octomore 1 0x547ee184 21630.880920 0.000000 0.000000 0.000006
_DRAM_ENERGY_STATUS 0x619 octomore 0 0x5290a614 21136.648743 9.335970 0.000760 0.000732
_DRAM_ENERGY_STATUS 0x619 octomore 1 0x547ee357 21630.888046 9.376124 0.000760 0.000732
```

Intel print power: Read energy register and convert to power



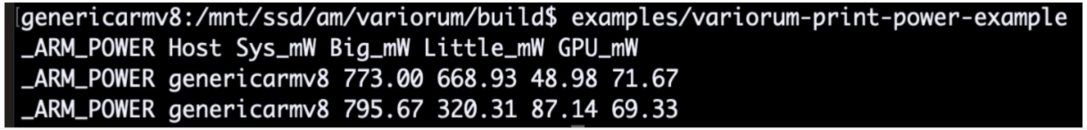
```
[patki1@lassen36:examples]$ ./variorum-print-power-example
_IBMPOWER Host Socket PWRSYS_W PWRPROC_W PWRMEM_W PWRGPU_W Timestamp_sec
_IBMPOWER lassen36 0 443 116 21 72 0.000019
_IBMPOWER lassen36 1 0 104 20 68 0.000082
_IBMPOWER lassen36 0 443 116 21 72 0.000128
_IBMPOWER lassen36 1 0 104 20 68 0.000158
```

IBM print power



```
[marathe1@lassen31 (dev) 0]$ examples/variorum-print-power-example
_GPU_POWER_USAGE Host Socket DeviceID Power_W
_GPU_POWER_USAGE lassen31 0 0 37.233002
_GPU_POWER_USAGE lassen31 0 1 35.787002
_GPU_POWER_USAGE lassen31 1 2 36.744002
_GPU_POWER_USAGE lassen31 1 3 37.723002
```

NVIDIA print power

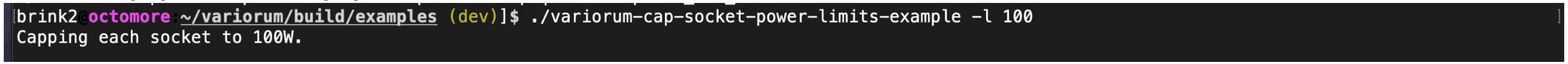


```
[genericarmv8:/mnt/ssd/am/variorum/build$ examples/variorum-print-power-example
_ARM_POWER Host Sys_mW Big_mW Little_mW GPU_mW
_ARM_POWER genericarmv8 773.00 668.93 48.98 71.67
_ARM_POWER genericarmv8 795.67 320.31 87.14 69.33
```

ARM print power

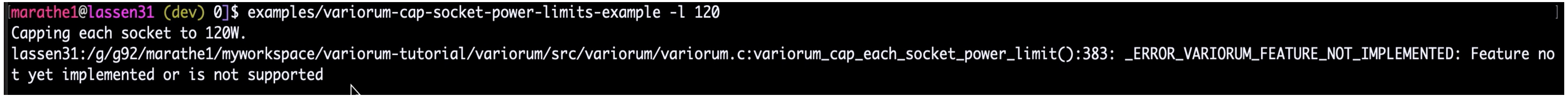
Variorum Examples: Source code annotations as well as non-intrusive monitoring are supported

Intel cap power: Cap power at the socket level



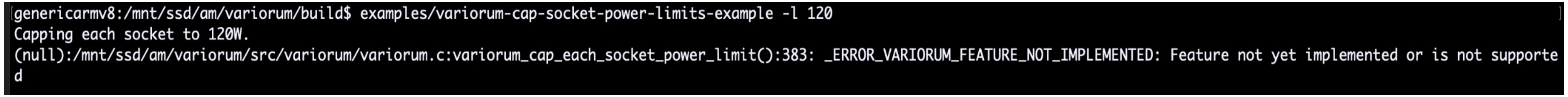
```
[brink2@octomore:~/variorum/build/examples (dev)]$ ./variorum-cap-socket-power-limits-example -l 100
Capping each socket to 100W.
```

NVIDIA cap power: Feature is available in NVML, but we haven't implemented yet



```
[marathe1@lassen31 (dev) 0]$ examples/variorum-cap-socket-power-limits-example -l 120
Capping each socket to 120W.
lassen31:/g/g92/marathe1/myworkspace/variorum-tutorial/variorum/src/variorum/variorum.c:variorum_cap_each_socket_power_limit():383: _ERROR_VARIORUM_FEATURE_NOT_IMPLEMENTED: Feature no
t yet implemented or is not supported
```

ARM cap power: Feature is not supported, only DVFS

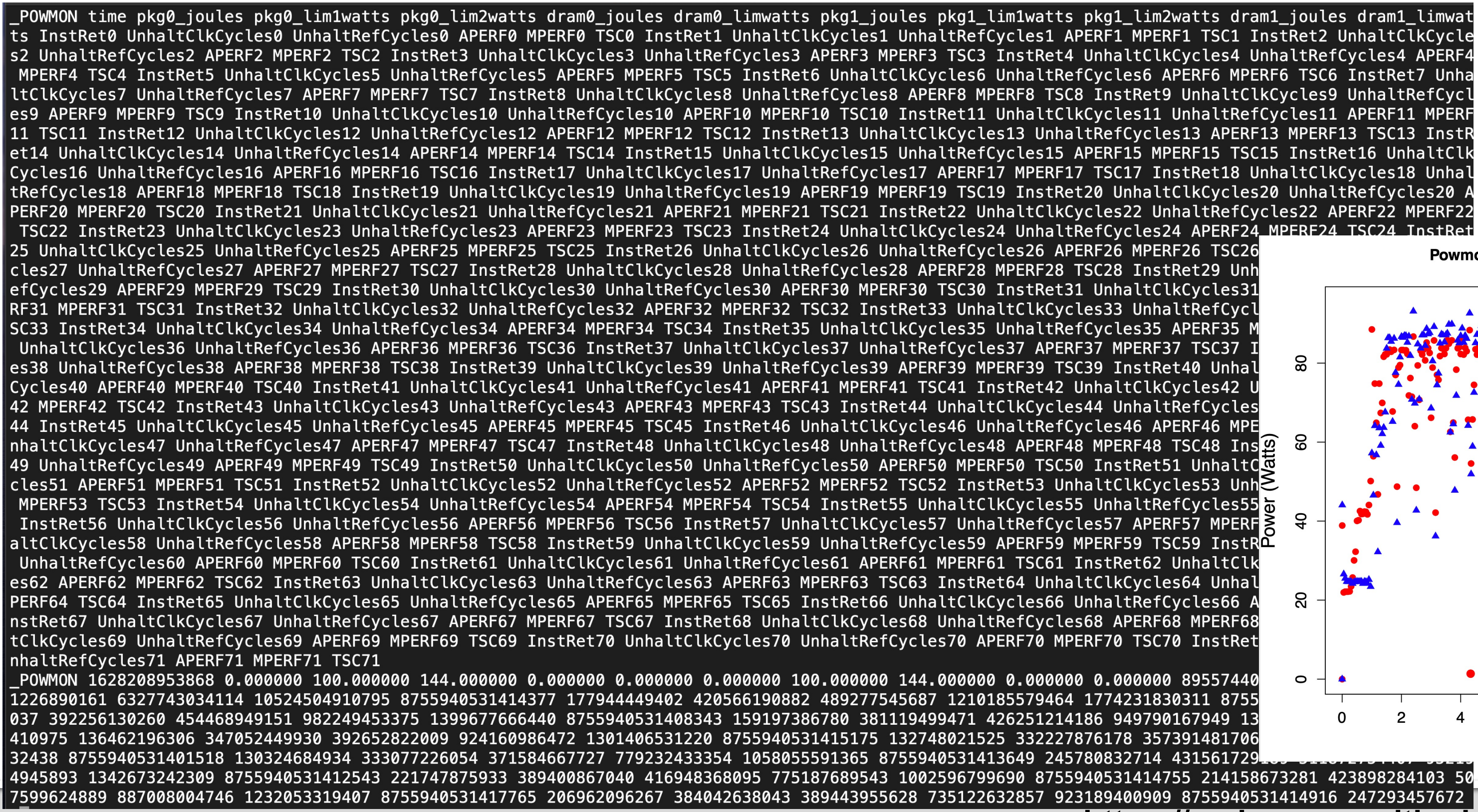


```
[genericarmv8:/mnt/ssd/am/variorum/build$ examples/variorum-cap-socket-power-limits-example -l 120
Capping each socket to 120W.
(null):/mnt/ssd/am/variorum/src/variorum/variorum.c:variorum_cap_each_socket_power_limit():383: _ERROR_VARIORUM_FEATURE_NOT_IMPLEMENTED: Feature not yet implemented or is not supporte
d
```

IBM cap power: Feature is not supported, only node-level

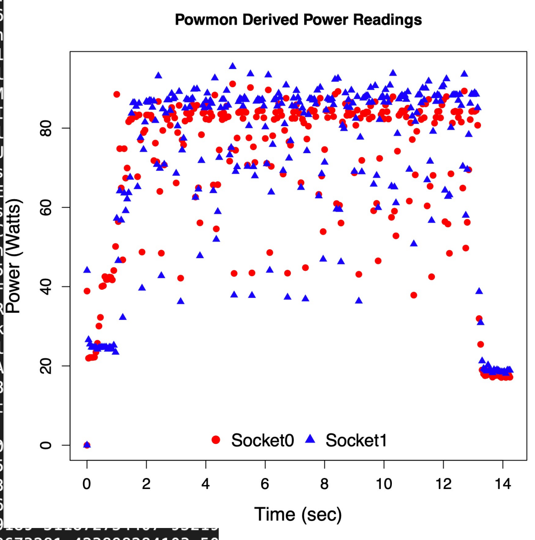
Demo of Variorum Non-Intrusive Monitoring: Powmon

Powmon: Non-intrusive monitoring tool across architectures



```
_POWMON time pkg0_joules pkg0_lim1watts pkg0_lim2watts dram0_joules dram0_limwatts pkg1_joules pkg1_lim1watts pkg1_lim2watts dram1_joules dram1_limwat
ts InstRet0 UnhaltClkCycles0 UnhaltRefCycles0 APERF0 MPERF0 TSC0 InstRet1 UnhaltClkCycles1 UnhaltRefCycles1 APERF1 MPERF1 TSC1 InstRet2 UnhaltClkCycle
s2 UnhaltRefCycles2 APERF2 MPERF2 TSC2 InstRet3 UnhaltClkCycles3 UnhaltRefCycles3 APERF3 MPERF3 TSC3 InstRet4 UnhaltClkCycles4 UnhaltRefCycles4 APERF4
 MPERF4 TSC4 InstRet5 UnhaltClkCycles5 UnhaltRefCycles5 APERF5 MPERF5 TSC5 InstRet6 UnhaltClkCycles6 UnhaltRefCycles6 APERF6 MPERF6 TSC6 InstRet7 Unha
ltClkCycles7 UnhaltRefCycles7 APERF7 MPERF7 TSC7 InstRet8 UnhaltClkCycles8 UnhaltRefCycles8 APERF8 MPERF8 TSC8 InstRet9 UnhaltClkCycles9 UnhaltRefCycl
es9 APERF9 MPERF9 TSC9 InstRet10 UnhaltClkCycles10 UnhaltRefCycles10 APERF10 MPERF10 TSC10 InstRet11 UnhaltClkCycles11 UnhaltRefCycles11 APERF11 MPERF
11 TSC11 InstRet12 UnhaltClkCycles12 UnhaltRefCycles12 APERF12 MPERF12 TSC12 InstRet13 UnhaltClkCycles13 UnhaltRefCycles13 APERF13 MPERF13 TSC13 InstR
et14 UnhaltClkCycles14 UnhaltRefCycles14 APERF14 MPERF14 TSC14 InstRet15 UnhaltClkCycles15 UnhaltRefCycles15 APERF15 MPERF15 TSC15 InstRet16 UnhaltClk
Cycles16 UnhaltRefCycles16 APERF16 MPERF16 TSC16 InstRet17 UnhaltClkCycles17 UnhaltRefCycles17 APERF17 MPERF17 TSC17 InstRet18 UnhaltClkCycles18 Unhal
tRefCycles18 APERF18 MPERF18 TSC18 InstRet19 UnhaltClkCycles19 UnhaltRefCycles19 APERF19 MPERF19 TSC19 InstRet20 UnhaltClkCycles20 UnhaltRefCycles20 A
PERF20 MPERF20 TSC20 InstRet21 UnhaltClkCycles21 UnhaltRefCycles21 APERF21 MPERF21 TSC21 InstRet22 UnhaltClkCycles22 UnhaltRefCycles22 APERF22 MPERF22
 TSC22 InstRet23 UnhaltClkCycles23 UnhaltRefCycles23 APERF23 MPERF23 TSC23 InstRet24 UnhaltClkCycles24 UnhaltRefCycles24 APERF24 MPERF24 TSC24 InstRet
25 UnhaltClkCycles25 UnhaltRefCycles25 APERF25 MPERF25 TSC25 InstRet26 UnhaltClkCycles26 UnhaltRefCycles26 APERF26 MPERF26 TSC26
cles27 UnhaltRefCycles27 APERF27 MPERF27 TSC27 InstRet28 UnhaltClkCycles28 UnhaltRefCycles28 APERF28 MPERF28 TSC28 InstRet29 Unh
efCycles29 APERF29 MPERF29 TSC29 InstRet30 UnhaltClkCycles30 UnhaltRefCycles30 APERF30 MPERF30 TSC30 InstRet31 UnhaltClkCycles31
RF31 MPERF31 TSC31 InstRet32 UnhaltClkCycles32 UnhaltRefCycles32 APERF32 MPERF32 TSC32 InstRet33 UnhaltClkCycles33 UnhaltRefCycl
SC33 InstRet34 UnhaltClkCycles34 UnhaltRefCycles34 APERF34 MPERF34 TSC34 InstRet35 UnhaltClkCycles35 UnhaltRefCycles35 APERF35 M
 UnhaltClkCycles36 UnhaltRefCycles36 APERF36 MPERF36 TSC36 InstRet37 UnhaltClkCycles37 UnhaltRefCycles37 APERF37 MPERF37 TSC37 I
es38 UnhaltRefCycles38 APERF38 MPERF38 TSC38 InstRet39 UnhaltClkCycles39 UnhaltRefCycles39 APERF39 MPERF39 TSC39 InstRet40 Unhal
Cycles40 APERF40 MPERF40 TSC40 InstRet41 UnhaltClkCycles41 UnhaltRefCycles41 APERF41 MPERF41 TSC41 InstRet42 UnhaltClkCycles42 U
42 MPERF42 TSC42 InstRet43 UnhaltClkCycles43 UnhaltRefCycles43 APERF43 MPERF43 TSC43 InstRet44 UnhaltClkCycles44 UnhaltRefCycles
44 InstRet45 UnhaltClkCycles45 UnhaltRefCycles45 APERF45 MPERF45 TSC45 InstRet46 UnhaltClkCycles46 UnhaltRefCycles46 APERF46 MPE
nhaltClkCycles47 UnhaltRefCycles47 APERF47 MPERF47 TSC47 InstRet48 UnhaltClkCycles48 UnhaltRefCycles48 APERF48 MPERF48 TSC48 Ins
49 UnhaltRefCycles49 APERF49 MPERF49 TSC49 InstRet50 UnhaltClkCycles50 UnhaltRefCycles50 APERF50 MPERF50 TSC50 InstRet51 UnhaltC
cles51 APERF51 MPERF51 TSC51 InstRet52 UnhaltClkCycles52 UnhaltRefCycles52 APERF52 MPERF52 TSC52 InstRet53 UnhaltClkCycles53 Unh
 MPERF53 TSC53 InstRet54 UnhaltClkCycles54 UnhaltRefCycles54 APERF54 MPERF54 TSC54 InstRet55 UnhaltClkCycles55 UnhaltRefCycles55
 InstRet56 UnhaltClkCycles56 UnhaltRefCycles56 APERF56 MPERF56 TSC56 InstRet57 UnhaltClkCycles57 UnhaltRefCycles57 APERF57 MPERF
altClkCycles58 UnhaltRefCycles58 APERF58 MPERF58 TSC58 InstRet59 UnhaltClkCycles59 UnhaltRefCycles59 APERF59 MPERF59 TSC59 InstR
 UnhaltRefCycles60 APERF60 MPERF60 TSC60 InstRet61 UnhaltClkCycles61 UnhaltRefCycles61 APERF61 MPERF61 TSC61 InstRet62 UnhaltClk
es62 APERF62 MPERF62 TSC62 InstRet63 UnhaltClkCycles63 UnhaltRefCycles63 APERF63 MPERF63 TSC63 InstRet64 UnhaltClkCycles64 Unhal
PERF64 TSC64 InstRet65 UnhaltClkCycles65 UnhaltRefCycles65 APERF65 MPERF65 TSC65 InstRet66 UnhaltClkCycles66 UnhaltRefCycles66 A
nstRet67 UnhaltClkCycles67 UnhaltRefCycles67 APERF67 MPERF67 TSC67 InstRet68 UnhaltClkCycles68 UnhaltRefCycles68 APERF68 MPERF68
tClkCycles69 UnhaltRefCycles69 APERF69 MPERF69 TSC69 InstRet70 UnhaltClkCycles70 UnhaltRefCycles70 APERF70 MPERF70 TSC70 InstRet
nhaltRefCycles71 APERF71 MPERF71 TSC71
_POWMON 1628208953868 0.000000 100.000000 144.000000 0.000000 0.000000 0.000000 100.000000 144.000000 0.000000 0.000000 89557440
1226890161 6327743034114 10524504910795 8755940531414377 177944449402 420566190882 489277545687 1210185579464 1774231830311 8755
037 392256130260 454468949151 982249453375 1399677666440 8755940531408343 159197386780 381119499471 426251214186 949790167949 13
410975 136462196306 347052449930 392652822009 924160986472 1301406531220 8755940531415175 132748021525 332227876178 357391481706
32438 8755940531401518 130324684934 333077226054 371584667727 779232433354 1058055591365 8755940531413649 245780832714 431561729
4945893 1342673242309 8755940531412543 221747875933 389400867040 416948368095 775187689543 1002596799690 8755940531414755 214158673281 423898284103 50
7599624889 887008004746 1232053319407 8755940531417765 206962096267 384042638043 389443955628 735122632857 923189400909 8755940531414916 247293457672
```

Intel



https://variorum.readthedocs.io/

Variorum Current Support (as of v0.4.1)

- Initial v0.1.0 released Nov 11, 2019
  - Platforms and microarchitectures supported:
    - Intel: Kaby Lake, Skylake, Broadwell, Haswell, Ivy Bridge, Sandy Bridge
    - IBM: Power9
- Current release (April 2021), v0.4.1:
  - Platforms and microarchitectures supported:
    - Nvidia: Volta
    - ARM: Juno
    - AMD (under review)
  - JSON API to integrate with external tools (e.g., Kokkos, Caliper, GEOPM, Flux)

https://github.com/llnl/variorum

Adding a vendor-neutral JSON interface

- Many of Variorum's APIs are printing output to stdout for user to parse
  - While nice for providing a friendly interface to understanding the hardware-level metrics, this limits ability for Variorum to provide these metrics to an external tool

- Added int variorum_get_node_power_json(json_t *) to integrate variorum with other tools (*e.g.*, Flux, Caliper, Kokkos, GEOPM)

  - { "hostname": (string),
  - "timestamp": (int),
  - "power_node": (int),
  - "power_cpu_socket_<id>": (int)
  - "power_mem_socket_<id>": (int)
  - "power_gpu_socket_<id>": (int) }

$$\left.\vphantom{\begin{matrix}a\\b\\c\\d\\e\\f\end{matrix}}\right\}$$ JSON object keys

- Example: Reporting end-to-end power usage for Kokkos loops

- Example: Provide power-awareness to Flux scheduling model enabling resources to be assigned based on available power

Implementing a vendor-neutral node-level power cap interface

- What if the device does not provide an explicit power knob for the node?

- IBM Power9 provides a node-level power knob through OPAL (part of IBM firmware)
  - But, not all platforms provide this same interface
  - Intel provides a socket-level power knob

- Intel provides CPU- and DRAM-level power knobs through MSRs, while IBM provides a power shifting ratio to determine the power limit of the CPU and GPU components

- Variorum's current implementation:

```
int variorum_cap_best_effort_node_power_limit(int power_lim)
```

  - IBM: caps the power limit to the node as expected
  - Intel: uniformly distribute the power limit across all sockets in the node (ignore memory and uncore power)

Join Us for Module 2 on Aug 27, 4:00PM-5:30PM PDT

“Integrating Variorum with System Software and Tools”

- The HPC Power Stack revisited: need for power management at various levels
- GEOPM: job-level power management
- Flux and SLURM (Research Extensions): system-level power management
- Kokkos and Caliper: application and workflow power management
- Upcoming Features in Variorum
- The HPC Power Stack Roadmap

Join our mailing list: variorum-users@llnl.gov

Questions? variorum-maintainers@llnl.gov

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory