Hewlett Packard Enterprise

On Monitoring Energy Consumption of Frontier Applications

Ashesh Sharma, HPE
Bruno Villasenor, AMD

January 2025

Outline

- Cray power management counters
  - Workload manager plugins
  - Cray PAT
  - PAPI

- AMD's RAPL counters for CPUs

- AMD power monitoring tools for GPUs
  - ROCm-SMI
    - ROCm-SMI LIB
  - Omnitrace
  - Omnistat

- Power and Energy optimization

- Summary

Acknowledgements

- At HPE:
  - Steve Martin – Cray PM counters
  - Bill Homer – CrayPat
  - Marcus Wagner and Steve Abbott – COE experts on CrayPat/Apprentice
  - Anna Yue – RAPL counters

- At AMD:
  - Donald Cheung and Shuzhou (Bill) Liu - ROCm-SMI
  - Karl W. Schulz and Jorda Polo – Omnistat
  - Ian Chui – Firmware
  - Noah Wolfe and Paul Bauman - COE experts in power measurements
  - Nick Malaya – COE leadership

Resources

- SLURM plugin for Cray PM counters (https://slurm.schedmd.com/slurm.conf.html#OPT_AcctGatherEnergyType)

- CrayPat (https://cpe.ext.hpe.com/docs/latest/performance-tools/index.html#id1)
  - API for Cray PM counters (https://cpe.ext.hpe.com/docs/latest/performance-tools/man5/cray_pm.html)
  - Various components of CrayPat (https://cpe.ext.hpe.com/docs/latest/performance-tools/index.html#man-pages)

- PAPI (https://icl.utk.edu/projects/papi/files/documentation/PAPI_USER_GUIDE.pdf)

- ROCm links
  - ROCm-SMI (https://github.com/ROCm/rocm_smi_lib/tree/master/python_smi_tools#usage)
  - Omnitrace (https://rocm.github.io/omnitrace/)
    - Recent OLCF tutorial (https://www.olcf.ornl.gov/wp-content/uploads/Omnitrace_by_Example.pdf)
  - ROCm-SMI LIB (https://rocm.docs.amd.com/projects/rocm_smi_lib/en/latest/)

Cray Power Management (PM) Counters

Cray PM Counters: Overview

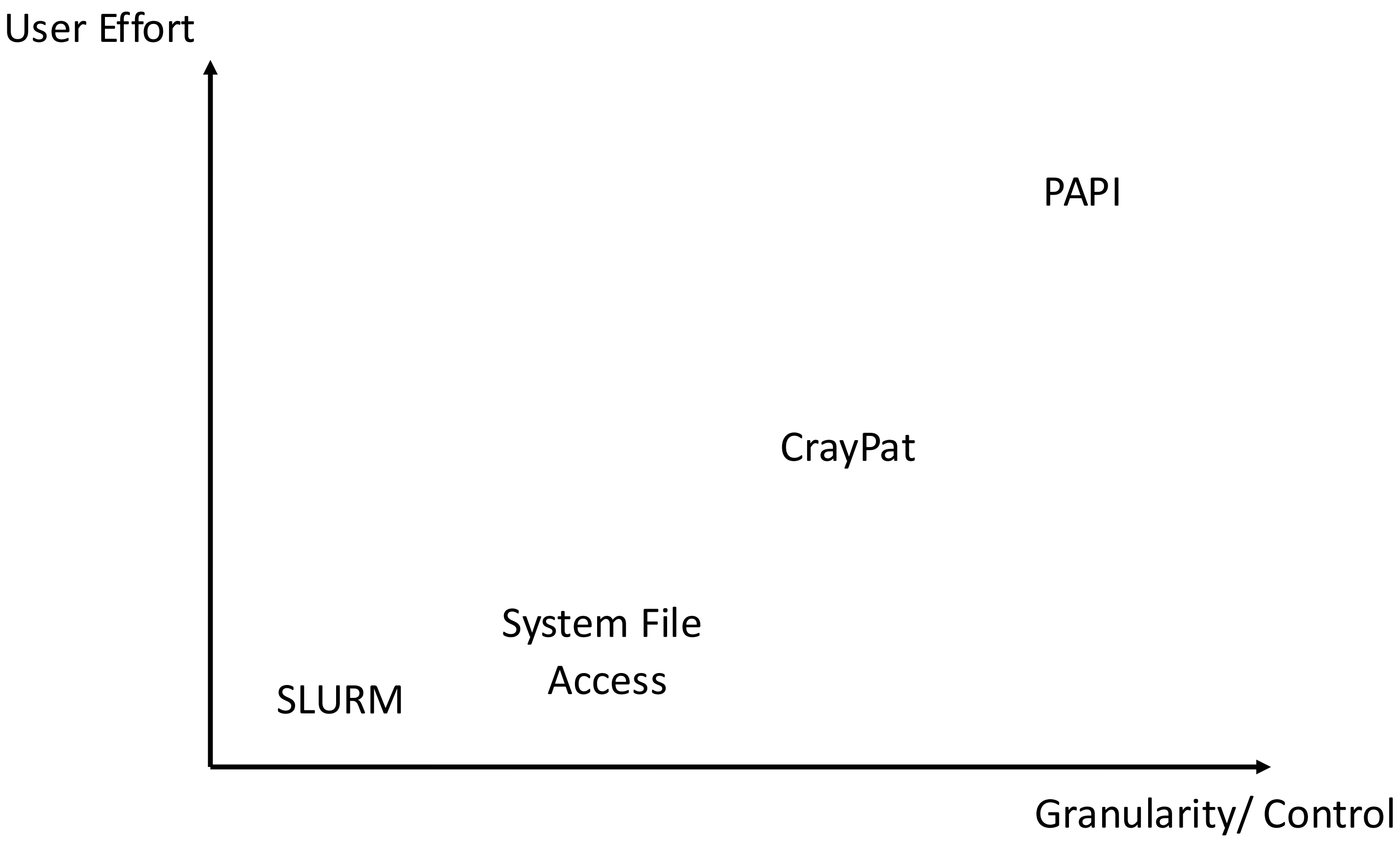
- Node-level power management counters first developed for Cray XC30 systems
  - Energy and power measurements from major components on the node
  - Reflect measurements from sensors on the node card/motherboard at the input end of the bus connecting the component
  - Include power caps for GPUs, modification of which requires elevated privileges
  - Measured values stored in files on compute nodes, in /sys/cray/pm_counters

- PM counter data is produced by the node controller
  - Written in the system files out-of-band, but reading the counter files is in-band
  - Written at 10 Hz update rate
  - Data is cached, so oversampling counters has minimal impact on performance

- Tested and validated
  - Accuracy for readings (on Frontier) are within 5%-10%

- Examples used
  - ECP application PeleC (https://github.com/AMReX-Combustion/PeleC)
  - OLCF multinode PyTorch script (https://docs.olcf.ornl.gov/software/python/pytorch_frontier.html#ex-code)

Cray PM Counters: Overview



Cray PM Counters: Monitoring Energy using Workload Managers

- Slurm energy plugin
  - Reports energy consumed by entire job
  - Requires that the jobacct_gather plugin be installed and operational; already configured on Frontier
    – In slurm.conf, set AcctGatherEnergyType=acct_gather_energy/pm_counters
  - Collect information during job execution sstat --jobs=<jobid>.batch --format=ConsumedEnergy
  - Upon job end, can read from the slurm database using sacct -j <JobId> -o ConsumedEnergy
  - Output gives values for each job step; output total energy across the job using --allsteps

User commands

Cray PM Counters: Files in /sys/cray/pm_counters

- 25 files on every compute node
- Node power/energy accounts for GPUs, CPUs, node memory, Cassini cards, node controller
- Freshness counter indicates validity of the data
  - Increments at 10 Hz
  - If not incrementing, all counters are invalid
- Energy counters are monotonically increasing, i.e., take a difference between two readings to get a meaningful value
- Power cap modifications require elevated privileges

File	Unit	Description
accel[0-3]_energy	Joules	GPU energy
accel[0-3]_power	Watts	GPU power, idle value of ≈ 90 W
accel[0-3]_power_cap	Watts	GPU power cap (modification requires admin privileges)
cpu0_temp	Celsius	CPU temperature, idle value of ≈ 35 °C
cpu_energy	Joules	CPU energy
cpu_power	Watts	CPU power, idle value of ≈ 35 W
memory_energy	Joules	Node memory energy
memory_power	Watts	Node memory power, idle value of ≈ 60 W
energy	Joules	Energy for the entire node
power	Watts	Power for the entire node, idle value of 620 W
power_cap	Watts	Power cap for entire node
raw_scan_hz	Hertz	Rate at which PM counters are updated, set at 10 Hz
freshness	-	Increments at raw_scan_hz (10 Hz)
generation	-	Increments when a power cap is changed
startup	-	Timestamp of counter subsystem
version	-	Revision of protocol

Cray PM Counters: Reading Files in /sys/cray/pm_counters

- Sample bash script for one compute node
  - Reading counters is in-band
  - Run on a core, if available, that is not running the application
  - For a multi-node job, accumulate readings from every compute node

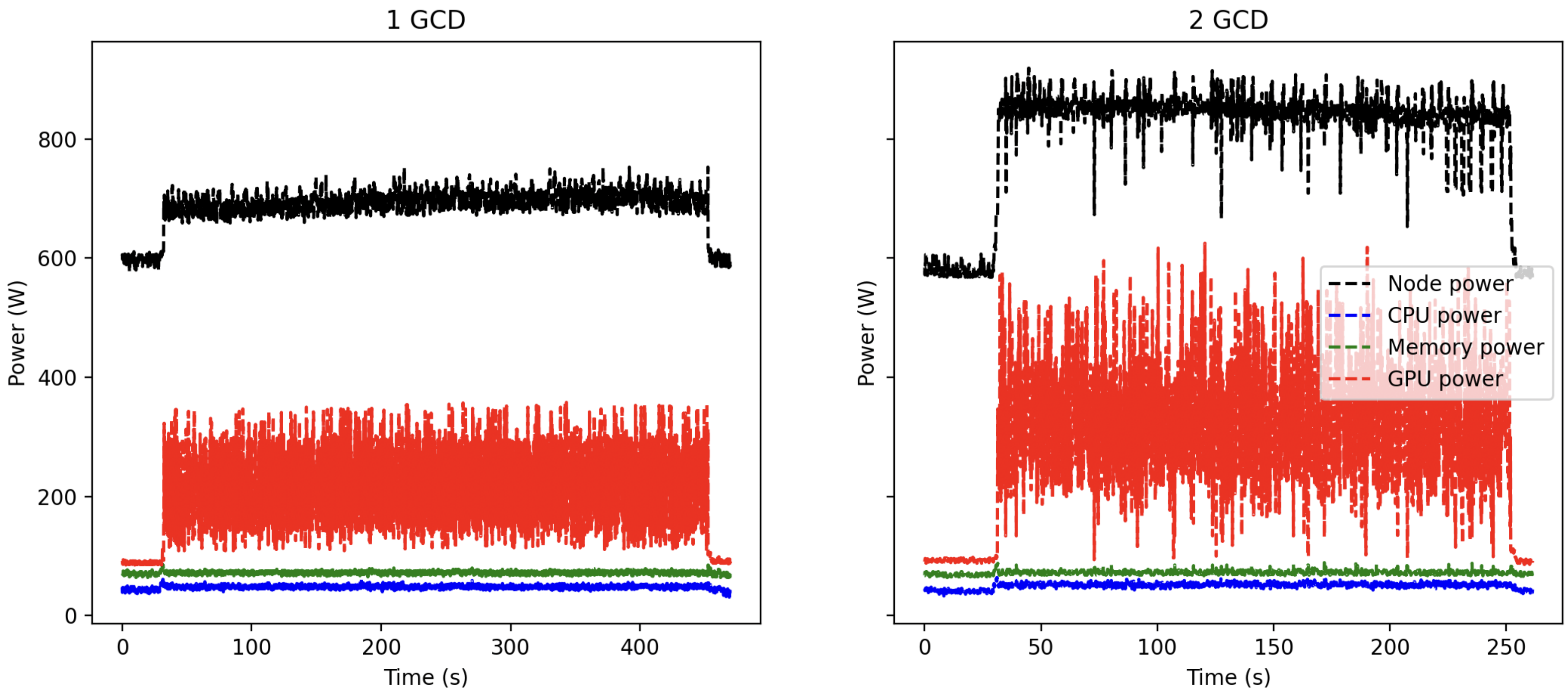
```
#/bin/bash
out_file="power_usage.txt"
PM=/sys/cray/pm_counters
NSAMPLES=1000

for i in $(seq 1 $NSAMPLES); do
  start_freshness=$(cat $PM/freshness)
  POWER=$(awk '{print $1}' "$PM/power")
  GPU_POWER=$(awk '{print $1}' "$PM/accel2_power")
  end_freshness=$(cat $PM/freshness)

if [ $end_freshness -eq $start_freshness ]; then
    echo >> $out_file
    echo "$i $(date +'%T.%6N')" >> $out_file
    echo "Total Power: $POWER" >> $out_file
    echo "GPU Power: $GPU_POWER" >> $out_file
    echo >> $out_file
  fi
done
```

Cray PM Counters: Reading Files in /sys/cray/pm_counters

- Monitoring power usage of 1 GCD vs 2 GCD for same problem size
  - Recorded power data using bash script from previous slide
  - Sleep for 30 seconds before and after the application to capture idle power of the node
  - Node Energy consumed was 322 kJ for 1 GCD run, and 210 kJ for 2 GCD run



Cray PM Counters: Using CrayPat to Monitor Power and Energy

- Refers to the perftools modules on Frontier

- Static application binaries can be instrumented using pat_build
  - Requires modules perftools-base and perftools
  - https://cpe.ext.hpe.com/docs/24.07/performance-tools/man1/pat_build.html

- The profiling report can be generated using pat_report
  - https://cpe.ext.hpe.com/docs/24.07/performance-tools/man1/pat_report.html

- A default output of pat_report is simulation energy consumption
  - Total energy is the default output; per-node energy accessible using pat_report -v -O program_energy -b ni <experiment output>

- Can collect specific power counters by setting environment variables, depending on pat_build experiment type
  – List available counters by running papi_native_avail -i cray_pm on a compute node

```
==============================================================
  nid.1
--------------------------------------------------------------
  PM Energy Node  1,003 W    233,717 J
  PM Energy Cpu      83 W     19,270 J
  PM Energy Memory   82 W     19,169 J
  PM Energy Acc0    159 W     36,943 J
  PM Energy Acc1    155 W     36,181 J
  PM Energy Acc2    159 W     37,047 J
  PM Energy Acc3    159 W     37,133 J
  Process Time             233.051299 secs
==============================================================
  nid.8
--------------------------------------------------------------
  PM Energy Node    939 W    218,871 J
  PM Energy Cpu      80 W     18,701 J
  PM Energy Memory   82 W     19,203 J
  PM Energy Acc0    159 W     37,013 J
  PM Energy Acc1    153 W     35,618 J
  PM Energy Acc2    161 W     37,469 J
  PM Energy Acc3    155 W     36,192 J
  Process Time             233.056491 secs
==============================================================
  nid.3
--------------------------------------------------------------
  PM Energy Node    904 W    210,587 J
  PM Energy Cpu      85 W     19,705 J
  PM Energy Memory   82 W     19,082 J
  PM Energy Acc0    154 W     35,817 J
  PM Energy Acc1    156 W     36,354 J
  PM Energy Acc2    158 W     36,852 J
  PM Energy Acc3    155 W     36,049 J
  Process Time             233.050952 secs
```

Cray PM Counters: Fine Grain Reports with CrayPat Tracing Experiments

- CrayPat's tracing experiments for collecting Cray PM counters
  - Instrument executable using pat_build -w -g mpi,omp,io,hip –o <instrumented executable> <target executable>
  - Counters are collected at entry and exit of functions; trace energy (not power) counters
    - Set the counters to sample : export PAT_RT_PERFCTR="<list of PM counters separated by comma, no spaces>"
    - Valid for summary runs: export PAT_RT_SUMMARY=0 ; ≈ 8% overhead for the examples presented

- The pat_report output includes tables for function groups that have significant time across ranks

```
=====================================================================
 MPI / MPI_Waitall
---------------------------------------------------------------------
 Time%                                    46.1%
 Time                               117.765059 secs
 Imb. Time                            9.772030 secs
 Imb. Time%                               7.8%
 Calls                 143.287  /sec   16,874.2 calls
 PM_ENERGY:NODE          1,039 W        122,324 J
 PM_ENERGY:ACC0            174 W         20,442 J
 PM_ENERGY:ACC1            172 W         20,249 J
 PM_ENERGY:ACC2            173 W         20,326 J
 PM_ENERGY:ACC3            171 W         20,136 J
 Average Time per Call                0.006979 secs
 CrayPat Overhead : Time  0.0%
```

```
=====================================================================
 HIP
---------------------------------------------------------------------
 Time%                                    39.8%
 Time                               101.465940 secs
 Imb. Time                                  -- secs
 Imb. Time%                                 --
 Calls                   0.020  /sec        2.0 calls
 PM_ENERGY:NODE            767 W         77,803 J
 PM_ENERGY:ACC0            128 W         12,993 J
 PM_ENERGY:ACC1            128 W         12,953 J
 PM_ENERGY:ACC2            129 W         13,097 J
 PM_ENERGY:ACC3            128 W         12,998 J
 Average Time per Call               50.732970 secs
 CrayPat Overhead : Time  0.0%
```

Cray PM Counters: Fine Grain Reports with CrayPat Tracing Experiments

- Explore pat_help to parse power usage statistics
  - E.g., pat_report -d PM_ENERGY:NODE%,ti,tr,P -b ca,pe=HIDE gives a bottom-up call stack ordered by functions with highest percentage of node energy

```
==============================================================================
  amrex::Gpu::Device::streamSynchronize / pele::physics::reactions::ReactorCvode::cF_RHS
------------------------------------------------------------------------------
  PM_ENERGY:NODE%                              18.1%
  Time                                     51.524670 secs
  Calls                                           -- calls
  PM_ENERGY:NODE              820 W           42,266 J
  PM_ENERGY:ACC0              140 W            7,193 J
  PM_ENERGY:ACC1              139 W            7,186 J
  PM_ENERGY:ACC2              142 W            7,292 J
  PM_ENERGY:ACC3              141 W            7,251 J
  Average Time per Call                           -- secs
  CrayPat Overhead : Time   0.0%
==============================================================================
  amrex::Gpu::Device::streamSynchronize / pele::physics::reactions::ReactorCvode::cF_RHS / cvLsDQJtimes
------------------------------------------------------------------------------
  PM_ENERGY:NODE%                              14.1%
  Time                                     39.332558 secs
  Calls                                           -- calls
  PM_ENERGY:NODE              837 W           32,926 J
  PM_ENERGY:ACC0              143 W            5,631 J
  PM_ENERGY:ACC1              143 W            5,627 J
  PM_ENERGY:ACC2              145 W            5,713 J
  PM_ENERGY:ACC3              144 W            5,681 J
  Average Time per Call                           -- secs
  CrayPat Overhead : Time   0.0%
```

Cray PM Counters: Fine Grain Reports with CrayPat APA Experiments

- CrayPat's automatic program analysis for sampling Cray PM counters
  - Instrument executable using pat_build –o <instrumented executable> <target executable>
  - Counters are collected at specific time intervals; sample power (not energy) counters; reports max power across nodes

    Sampling frequency

    – Tell the program to collect performance counters: export PAT_RT_SAMPLING_DATA=perfctr@10

    – Set the PM counters to sample : export PAT_RT_PERFCTR="<list of PM counters separated by comma, no spaces>"

    – Valid for summary runs: export PAT_RT_SUMMARY=0; ≈ 1% overhead for the examples presented

- Default pat_report output includes tables for function groups that have significant sample hits across ranks

```
==================================
 MPI / MPI_Allgather
----------------------------------
 Samp%                 4.9%
 Samp               1,162.8
 Imb. Samp            163.2
 Imb. Samp%           12.5%
 PM_POWER:NODE      1,304 W
 PM_POWER:ACC0        372 W
 PM_POWER:ACC1        358 W
 PM_POWER:ACC2        361 W
 PM_POWER:ACC3        367 W
```

```
==================================
 HIP / hipMemcpyAsync
----------------------------------
 Samp%                 3.8%
 Samp                 894.8
 Imb. Samp            196.2
 Imb. Samp%           18.2%
 PM_POWER:NODE      1,329 W
 PM_POWER:ACC0        466 W
 PM_POWER:ACC1        438 W
 PM_POWER:ACC2        451 W
 PM_POWER:ACC3        446 W
```

Cray PM Counters: Fine Grain Reports with CrayPat APA Experiments

- Explore pat_help to parse power usage statistics
  - E.g., pat_report -d PM_POWER:NODE%,ti,tr,P -b ca,pe=HIDE gives a bottom-up call stack ordered by functions with highest percentage of node power

```
 PM_POWER |      Samp | PM_POWER | PM_POWER | PM_POWER | PM_POWER | PM_POWER | Caller
   :NODE% |           |    :NODE |    :ACC0 |    :ACC1 |    :ACC2 |    :ACC3 |  PE=HIDE
          |           |      (W) |      (W) |      (W) |      (W) |      (W) |

   100.0% | 23,848.6 |    1,329 |      484 |      463 |      451 |      474 | Total
|-----------------------------------------------------------------------------
|   100.0% |     20.6 |    1,329 |      371 |      337 |      377 |      359 | amrex::launch<>
||----------------------------------------------------------------------------
||   100.0% |     11.9 |    1,329 |      371 |      337 |      337 |      359 | amrex::detail::ParallelFor_doit<>
|||---------------------------------------------------------------------------
3||   100.0% |      3.1 |    1,329 |      300 |      314 |      337 |      359 | amrex::FabArray<>::pack_send_buffer_gpu<>
||||--------------------------------------------------------------------------
4|||   100.0% |      2.9 |    1,329 |      300 |      314 |      337 |      359 | amrex::FabArray<>::ParallelCopy_nowait
|||||-------------------------------------------------------------------------
5||||   100.0% |      1.9 |    1,329 |      300 |      314 |      337 |      359 | amrex::FillPatchSingleLevel<>
||||||------------------------------------------------------------------------
6|||||   100.0% |      1.1 |    1,329 |      300 |      314 |      337 |      359 | amrex::FillPatcher<>::fill<>
7|||||          |          |          |          |          |          |         |  amrex::AmrLevel::FillPatcherFill
8|||||          |          |          |          |          |          |         |   PeleC::do_sdc_iteration
9|||||          |          |          |          |          |          |         |    PeleC::do_sdc_advance
10||||          |          |          |          |          |          |         |     PeleC::advance
11||||          |          |          |          |          |          |         |      amrex::Amr::timeStep
12||||          |          |          |          |          |          |         |       amrex::Amr::coarseTimeStep
13||||          |          |          |          |          |          |         |        main
6|||||    82.3% |      0.8 |    1,094 |      157 |      159 |      101 |      157 | amrex::FillPatchIterator::FillFromLevel0
7|||||          |          |          |          |          |          |         |  amrex::FillPatchIterator::Initialize
8|||||          |          |          |          |          |          |         |   amrex::FillPatchIterator::FillPatchIterator
9|||||          |          |          |          |          |          |         |    amrex::AmrLevel::FillPatch
||||||||||--------------------------------------------------------------------
```

Cray PM Counters: Monitoring Energy Consumption of Python Applications

- Most AI/ML applications are coded in Python which is dynamically typed
  - pat_run can be used for tracing and sampling experiments; arguments are similar to pat_build;
  - Requires modules perftools-base and perftools-preload; use perftools-base/24.11.0
  - https://cpe.ext.hpe.com/docs/24.07/performance-tools/man1/pat_run.html

  - Tracing the OLCF multimode PyTorch example
    – srun -N2 -n16 -c7 --gpus-per-task=1 --gpu-bind=closest pat_run -w -g <options> python3 -W ignore -u ./multinode_olcf.py 2000 10 --master_addr=$MASTER_ADDR --master_port=3442
    – Set the PM counters to sample : export PAT_RT_PERFCTR="<list of PM counters separated by comma, no spaces>"

- Default pat_report output includes tables for function groups that have significant time across ranks

```
==========================================================
  HIP
----------------------------------------------------------
  Time%                                      57.8%
  Time                                   71.033646 secs
  Imb. Time                                     -- secs
  Imb. Time%                                    --
  Calls                    0.007 M/sec   466,662.9 calls
  PM_ENERGY:NODE             848 W          60,215 J
  PM_ENERGY:ACC0             141 W          10,025 J
  PM_ENERGY:ACC1             148 W          10,479 J
  PM_ENERGY:ACC2             148 W          10,498 J
  PM_ENERGY:ACC3             138 W           9,836 J
  Average Time per Call                   0.000152 secs
  CrayPat Overhead : Time   0.5%
  Acc Util                                    0.4%
```

```
==========================================================
  HIP / hipMemcpyWithStream
----------------------------------------------------------
  Time%                                      56.3%
  Time                                   69.190204 secs
  Imb. Time                               0.943970 secs
  Imb. Time%                                  1.4%
  Calls                  232.302  /sec    16,073.0 calls
  PM_ENERGY:NODE             853 W          59,008 J
  PM_ENERGY:ACC0             142 W           9,829 J
  PM_ENERGY:ACC1             149 W          10,276 J
  PM_ENERGY:ACC2             149 W          10,296 J
  PM_ENERGY:ACC3             139 W           9,643 J
  Average Time per Call                   0.004305 secs
  CrayPat Overhead : Time   0.0%
  Acc Util                                    0.1%
```

Cray PM Counters: Monitoring Energy Consumption of Python Applications

- Explore pat_help to parse power usage statistics
  - E.g., pat_report -d PM_ENERGY:NODE%,ti,tr,P -b ca,pe=HIDE gives a bottom-up call stack ordered by functions with highest percentage of node energy

```
==============================================================================
  hipLaunchKernel / python.cross_entropy / python._run_batch / python._run_epoch
------------------------------------------------------------------------------
  PM_ENERGY:NODE%                                0.2%
  Time                                      0.405565 secs
  Calls                        0.099 M/sec  40,000.0 calls
  PM_ENERGY:NODE                 448 W           182 J
  PM_ENERGY:ACC0                  76 W            31 J
  PM_ENERGY:ACC1                  80 W            32 J
  PM_ENERGY:ACC2                  78 W            32 J
  PM_ENERGY:ACC3                  72 W            29 J
  Average Time per Call                     0.000010 secs
  CrayPat Overhead : Time   8.2%
  Acc Util                                       0.1%
==============================================================================
  hipLaunchKernel / python.cross_entropy / python._run_batch / python._run_epoch / python.train
------------------------------------------------------------------------------
  PM_ENERGY:NODE%                                0.2%
  Time                                      0.405565 secs
  Calls                        0.099 M/sec  40,000.0 calls
  PM_ENERGY:NODE                 448 W           182 J
  PM_ENERGY:ACC0                  76 W            31 J
  PM_ENERGY:ACC1                  80 W            32 J
  PM_ENERGY:ACC2                  78 W            32 J
  PM_ENERGY:ACC3                  72 W            29 J
  Average Time per Call                     0.000010 secs
  CrayPat Overhead : Time   8.2%
  Acc Util                                       0.1%
```

Cray PM Counters: Suggested PM Counters for CrayPat Experiments

- With the perftools-base modules loaded, users can list all available counters by running papi_native_avail -i cray_pm on a compute node

Suggested counters for Tracing	Suggested counters for APA (Sampling)
PM_ENERGY:NODE PM_ENERGY:CPU PM_ENERGY:ACC0 PM_ENERGY:ACC1 PM_ENERGY:ACC2 PM_ENERGY:ACC3 PM_ENERGY:MEMORY	PM_POWER:NODE PM_POWER:CPU PM_POWER:ACC0 PM_POWER:ACC1 PM_POWER:ACC2 PM_POWER:ACC3 PM_POWER:MEMORY

Cray PM Counters: Interfacing with Apprentice3

- GUI support through Apprentice3 to monitor power usage
  - Apprentice3 available since perftools-base/24.03.0

Text Report

calltree+hwpc

Enable Thresholds

Calltree View by Function with Counters

Samp%	Samp	PM_POWER:NODE (W)	PM_POWER:ACC0 (W)	PM_POWER:ACC1 (W)	PM_POWER:ACC2 (W)	PM_POWER:ACC3 (W)	Calltree
100.0%	23848.6	1318	361	406	371	375	Total
99.9%	23825.0	1318	361	385	371	375	main
98.7%	23537.9	--	--	--	--	--	amrex::Amr::coarseTimeStep
98.5%	23501.4	--	--	--	--	--	amrex::Amr::timeStep
88.5%	21108.8	1130	329	336	344	219	PeleC::advance
88.5%	21108.6	1130	329	336	344	219	PeleC::do_sdc_advance
88.5%	21106.7	--	--	--	--	--	PeleC::do_sdc_iteration
0.0%	1.3	1130	204	236	325	219	amrex::MultiFab::Copy
0.0%	0.4	875	329	336	344	205	amrex::FabArray<>::setVal<>
0.0%	0.1	--	--	--	--	--	amrex::StateData::allocOldData
0.0%	0.0	--	--	--	--	--	amrex::Print::~Print

Table Notes

```
Table options
  This table shows functions that have significant exclusive time,
    averaged across ranks, as the leaves in a calltree view. Time
    shown for a caller (non-leaf) is the sum of times below it in the
    tree.
    Processor HW counter data is also shown, if available
  For further explanation, use:  pat_report -v -O calltree+hwpc ...

  Table option:
    -O calltree+hwpc -s table.show_data="cols"
  Options implied by table option:
    -d ti%,ti,tr,P -b ct,pe=HIDE -s table.group=Calltree -s
    table.show_data=cols
  Other options:
    -s content='tables'
    -T
```

Cray PM Counters: Manual Code Instrumentation using PAPI

- Performance Application Programming Interface (PAPI) for monitoring power usage
  - Just need to module load papi, and #include <papi.h> for C or #include "f90papi.h" for Fortran
  - List available counters by running papi_native_avail -i cray_pm on a compute node
  - PAPI code modifications need to execute on only 1 rank per node

PAPI modifications for a C program

```
int eventset = PAPI_NULL;
int events[2] = {0};
long long count[2] = {0};
...
PAPI_library_init(PAPI_VER_CURRENT);  // initialize PAPI
PAPI_create_eventset(&eventset); // create an eventset
...
PAPI_event_name_to_code("cray_pm:::PM_POWER:NODE", & events[0]); // get PAPI code from event string
PAPI_event_name_to_code("cray_pm:::PM_POWER:ACC2", & events[1]);
PAPI_add_events(eventset, events, 2); // add events to the eventset
...
PAPI_reset(eventset); // reset counters
PAPI_start(eventset); // start counters
...
PAPI_read(eventset, count); // read counters
...
PAPI_stop(eventset, count); // stop counters
PAPI_cleanup_eventset(eventset); // cleanup PAPI data structures
PAPI_destroy_eventset(&eventset);
```

Cray PM Counters: Manual Code Instrumentation using PAPI

- Upsides of using PAPI
  - API is lightweight and straightforward to implement
  - If intimately familiar with code, PAPI_read calls can be placed strategically to capture PM counters during GPU kernel execution

- Complexities with using PAPI
  - Intrusive
    - Requires knowledge of code
    - Recompilation
  - Need additional code/logic to process and aggregate counter values from each node
  - Needs additional consideration to process counters from different counter groups simultaneously
    - Need one PAPI_create_eventset call per counter group
    - Examples of different counter groups include Cray PM counters, Linux perf_event counters, ROCm counters, etc.

Running Average Power Limit (RAPL) Counters

RAPL Counters: Overview

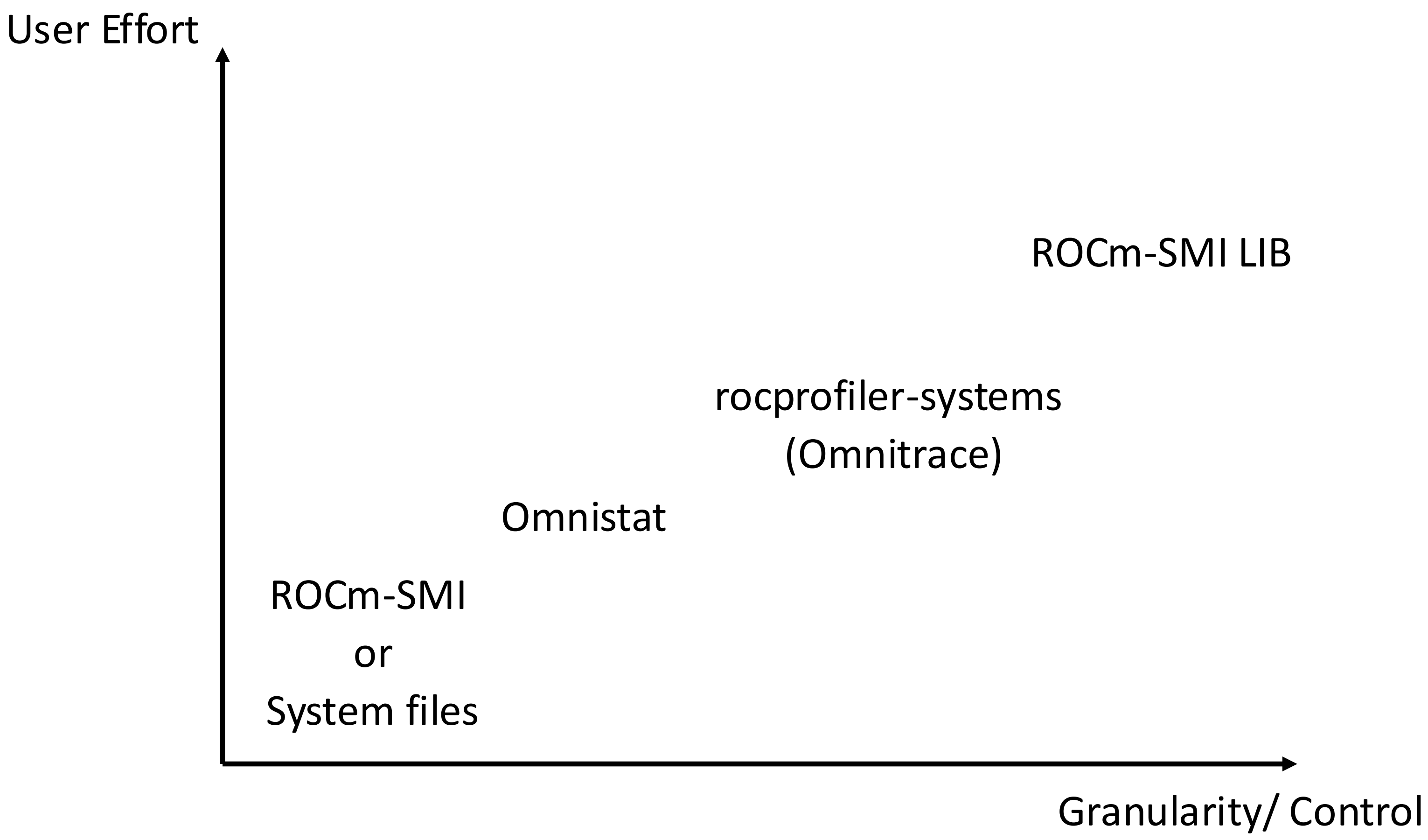
- Socket- and core-level power monitoring counters
  - PACKAGE_ENERGY reports energy used by CPU socket
  - PP0_ENERGY reports energy used by energy used by all cores and caches of a socket
    - Depending on architecture, will measure all cores, or only 1; through PAPI, it will measure all cores
    - Excludes uncore components

- Collected in model-specific registers (MSR)
  - 32-bit raw values that require scaling for conversion to SI units
  - Written at 1 kHz update rate
  - Wraparound time, which varies based on energy consumption; can be as low as 60 seconds
  - Direct access to registers requires elevated privileges

- Supported by CrayPat and PAPI
  - Obtaining counter values follows exact approach described for Cray PM counters

AMD Tools for Power Management of GPUs

AMD Power Management Overview



ROCm System Management Interface (ROCm-SMI)

ROCm-SMI: Overview

- GPU-level resources management framework
  - Shows information about GPU power, shader and memory frequency, temperature, GPU utilization, memory usage, etc.
  - Average power measurements from GPU, accounting for both GCDs on MI250X
    – Device-specific power can be parsed using rocm-smi -d <Device> --showpower
  - Include power caps, modification of which requires elevated privileges

```
========================================== ROCm System Management Interface ==========================================
==================================================== Concise Info ====================================================
Device  Node  IDs              Temp    Power  Partitions          SCLK    MCLK     Fan  Perf  PwrCap  VRAM%  GPU%
              (DID,     GUID)  (Edge)  (Avg)  (Mem, Compute, ID)
======================================================================================================================
0       4     0x7408,   29312  30.0°C  91.0W  N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  560.0W  0%     0%
1       5     0x7408,   27578  30.0°C  N/A    N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  0.0W    0%     0%
2       6     0x7408,   3292   23.0°C  91.0W  N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  560.0W  0%     0%
3       7     0x7408,   26097  23.0°C  N/A    N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  0.0W    0%     0%
4       8     0x7408,   7704   25.0°C  87.0W  N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  560.0W  0%     0%
5       9     0x7408,   30477  28.0°C  N/A    N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  0.0W    0%     0%
6       10    0x7408,   10324  23.0°C  80.0W  N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  560.0W  0%     0%
7       11    0x7408,   329    24.0°C  N/A    N/A, N/A, 0         800Mhz  1600Mhz  0%   auto  0.0W    0%     0%
======================================================================================================================
=============================================== End of ROCm SMI Log ================================================
```

rocm-smi output on an idle compute node

- Reflects measurements from the system management unit (SMU) and reported by the firmware
  - ROCm-SMI queries system files written by the firmware for resources monitoring
  - On Frontier compute nodes, these system files are in /sys/class/drm/cardX/device/hwmon/hwmonX/ where X is the device ID (0,2,4,6) for each of the 4 MI250X GPUs on a node
  - Written at up to 1 kHz update rate
  - Files are written out-of-band and read/queried in-band

ROCm-SMI: Instrumenting Applications using ROCm-SMI LIB

- ROCm-SMI lib is a C library for Linux that provides interface for applications to monitor and control GPU resources
  - Part of the ROCm software stack
  - https://rocm.docs.amd.com/projects/rocm_smi_lib/en/latest/
  - Example of ROCm-SMI lib API calls for power measurement

```
#include "rocm_smi/rocm_smi.h”
…
rsmi_status_t ret;
uint32_t num_devices;
uint16_t dev_id;
RSMI_POWER_TYPE power_type;
uint64_t power, rsmi_power_avrg, rsmi_power_socket;
…
rsmi_init(0);
rsmi_num_monitor_devices(&num_devices);
…
rsmi_dev_power_ave_get (<device index>, 0, &rsmi_power_avrg);
rsmi_dev_power_get( <device index>, &power, &power_type);
…
rsmi_shut_down();
```

Power-related ROCm-SMI LIB calls

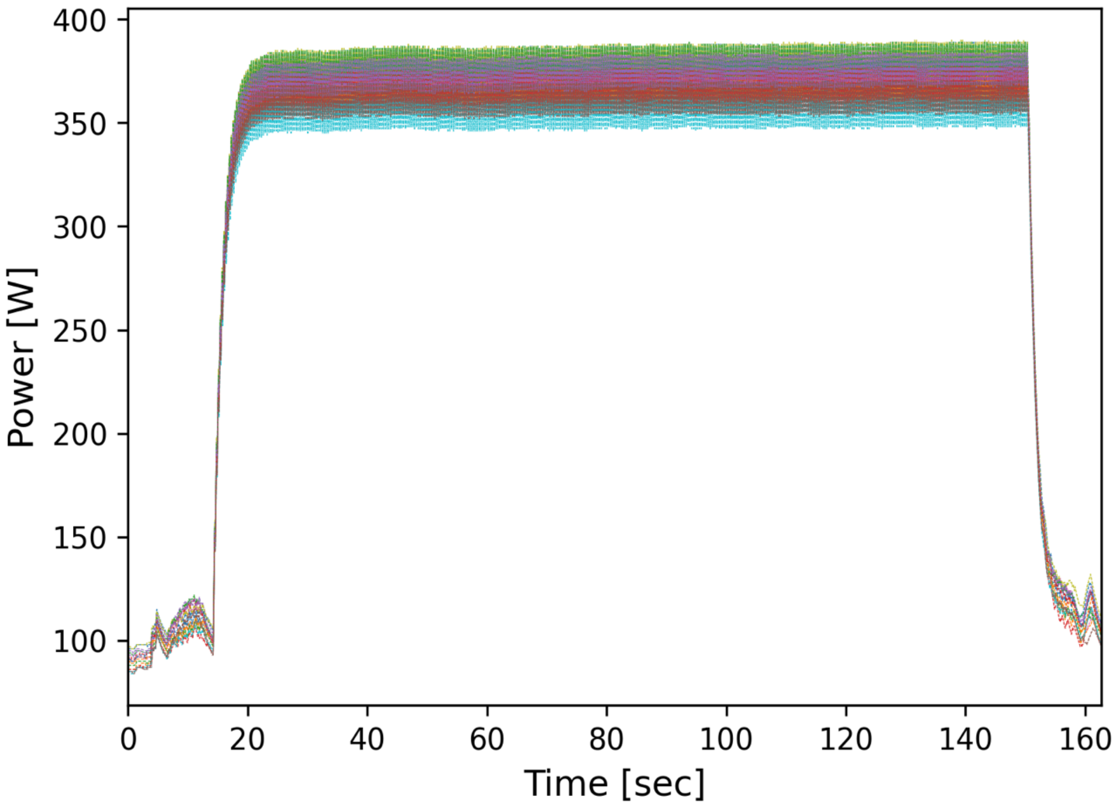
Returns average power on MI250X

ROCm-SMI: Power Profiling

- Power profile for the Cholla hydrodynamics app (https://github.com/cholla-hydro/cholla) running on 4 Frontier nodes
  - Sampling at 100 Hz
  - One background process per node queries power draw of the 4 GPUs
  - A section of the output for node 0:

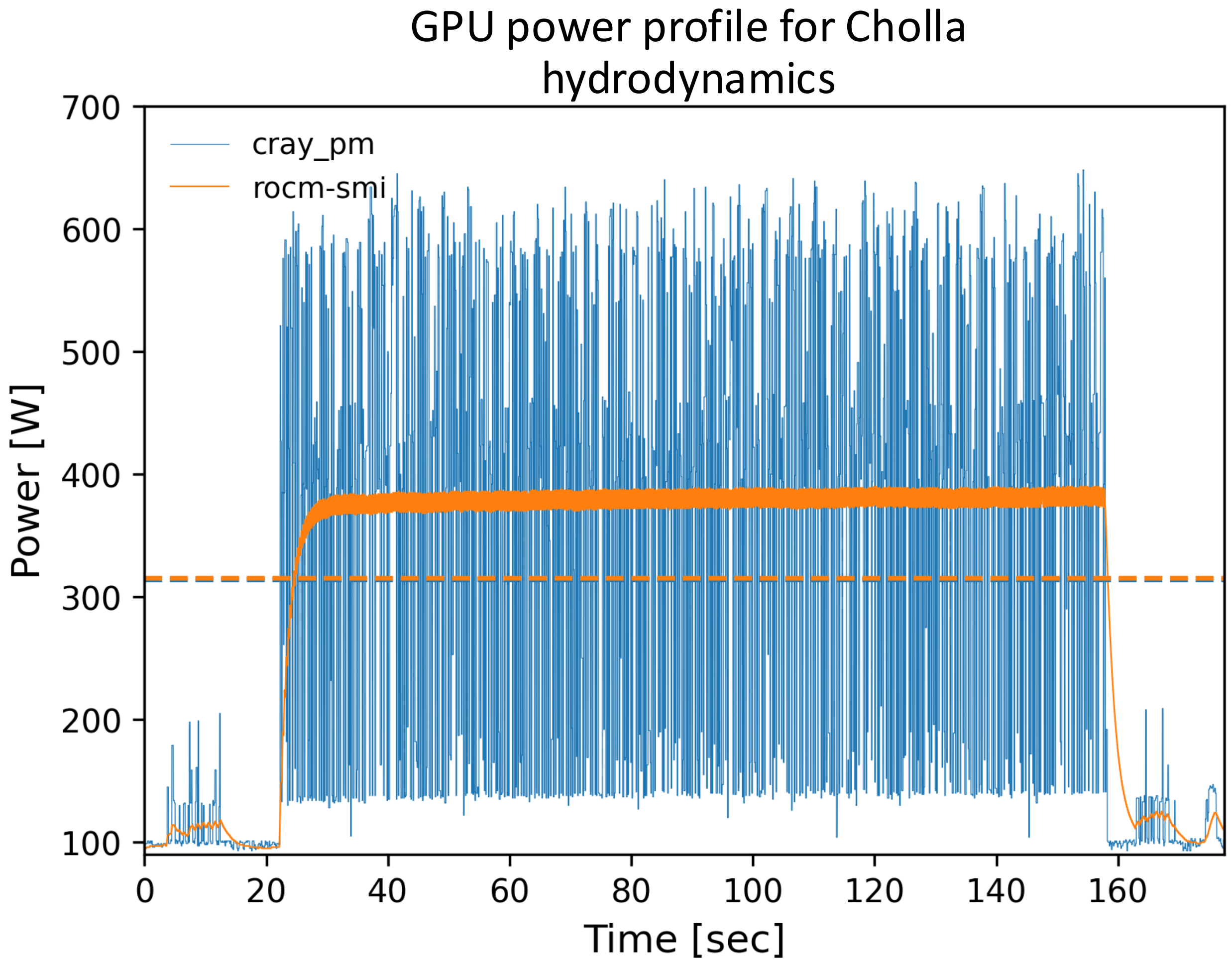
```
#date time pow-gpu0 pow-gpu1 pow-gpu2 pow-gpu3
2024-07-06 12:18:21.383780 379 373 369 365
2024-07-06 12:18:21.393852 377 371 367 363
2024-07-06 12:18:21.403927 376 370 366 362
2024-07-06 12:18:21.414002 374 368 364 360
2024-07-06 12:18:21.424077 373 367 363 359
2024-07-06 12:18:21.434152 371 366 361 357
2024-07-06 12:18:21.444227 370 364 360 356
2024-07-06 12:18:21.454302 370 364 360 356
2024-07-06 12:18:21.464373 370 364 360 356
2024-07-06 12:18:21.474447 371 365 361 357
2024-07-06 12:18:21.484896 372 367 362 359
2024-07-06 12:18:21.495376 374 368 364 360
2024-07-06 12:18:21.505824 374 369 364 361
```

GPU power profile
for all 16 MI250X



ROCm-SMI: Comparison with Cray PM Counters

- Currently, on MI250X GPUs, ROCm-SMI returns average power usage which differs from the instantaneous power reported by Cray PM counters
  - ROCm-SMI measurements are averaged over a time interval of ~6 milliseconds
  - Ongoing efforts to have the firmware report instantaneous power for MI250X
- The averaged measurements from Cray PM counters and ROCm-SMI over the entire application run are consistent (dashed horizontal lines)



rocprofiler-systems (Omnitrace)

rocprof-systems (Omnitrace): Overview

- High level view of the entire application run
- Holistic view of CPU, GPU, and system activity
  - Kernel-level power profile can be collected
  - Calls rocm-smi to collect power information
- Visualize in Perfetto (https://ui.perfetto.dev/); One Perfetto trace file for each MPI rank
- OLCF tutorial on Omnitrace by Gina Sitaraman: https://www.olcf.ornl.gov/wp-content/uploads/Omnitrace_by_Example.pdf

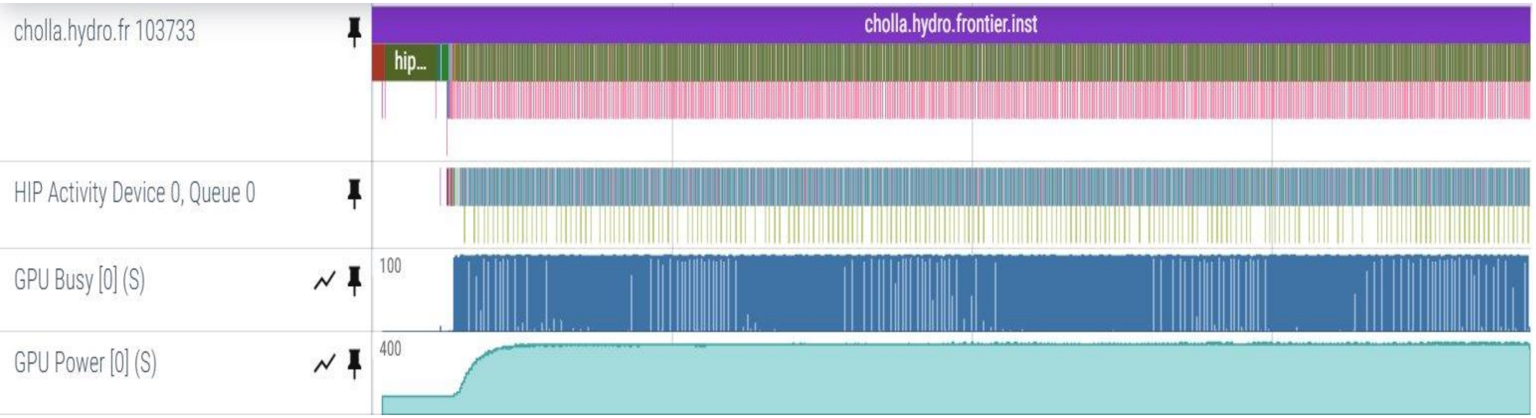
Omnitrace: How To Use

- Step 1: Load Omnitrace on Frontier
  module load rocm
  module load omnitrace/1.11.4

- Step 2: After building your application, instrument the executable
  omnitrace-instrument –o cholla.hydro.frontier.inst -- cholla.hydro.frontier

- Step 3: Run your application as usual, but calling omnitrace-run -- <instrumented_binary>
  srun -u -n 32 --ntasks-per-node=8 --gpus-per-node=8 --gpu-bind=closest \
  omnitrace-run -- cholla.hydro.frontier.inst parameter_file.txt

- Step 4: Copy the Omnitrace output to your local system and visualize the traces in Perffeto (https://ui.perfetto.dev/)



Omnitrace: Runtime Configuration

- Create an Omnitrace configuration file and edit it for your needs

```
omnitrace-avail -G  $PATH_TO_CONFIG/omnitrace.cfg
```

- Relevant configuration options for GPU power profiling

```
OMNITRACE_TRACE                   = true
OMNITRACE_PROFILE                 = true
OMNITRACE_USE_SAMPLING            = true
OMNITRACE_USE_PROCESS_SAMPLING    = true
OMNITRACE_USE_ROCM_SMI            = true
OMNITRACE_SAMPLING_FREQ           = 100
OMNITRACE_SAMPLING_GPUS           = $env:HIP_VISIBLE_DEVICES
OMNITRACE_SAMPLING_CPUS           = none
```

- Set your Omnitrace configuration file `export  OMNITRACE_CONFIG_FILE=$PATH_TO_CONFIG/omnitrace.cfg`

- Measured a 2-4% overhead when running without CPU sampling
- Measured a 25-30% overhead when sampling the CPUs

Omnitrace: Power Timelines

- Omnitrace profile for Cholla on 4 nodes showing GPU power usage.

Omnitrace: Power Statistics

- Omnitrace outputs several statistics for each MPI rank
  - GPU power statistics are available at the kernel level
  - Example GPU power statistics in file sampling_gpu_power-0.txt

POWER USAGE OF GPU(S)

LABEL	COUNT	DEPTH	METRIC	UNITS	MEAN	MIN	MAX	VAR	STDDEV	% SELF
CopyHostToDevice	166	0	sampling_gpu_power	watts	339.37	98.00	379.00	7963.46	89.24	100.0
CopyDeviceToHost	179	0	sampling_gpu_power	watts	372.23	100.00	381.00	526.97	22.96	100.0
Calc_dt_3D	118	0	sampling_gpu_power	watts	369.21	101.00	381.00	1582.34	39.78	100.0
FillBuffer	2	0	sampling_gpu_power	watts	103.50	102.00	105.00	4.50	2.12	100.0
PCM_Reconstruction_3D	1358	0	sampling_gpu_power	watts	364.47	108.00	372.00	590.49	24.30	100.0
Calculate_HLLC_Fluxes_CUDA	2781	0	sampling_gpu_power	watts	368.47	113.00	379.00	541.36	23.27	100.0
Update_Conserved_Variables_3D_half	1407	0	sampling_gpu_power	watts	368.66	118.00	375.00	536.20	23.16	100.0
Partial_Update_Advected_Internal_Energy_3D	261	0	sampling_gpu_power	watts	372.78	131.00	379.00	515.55	22.71	100.0
Update_Conserved_Variables_3D	1200	0	sampling_gpu_power	watts	373.75	133.00	380.00	468.30	21.64	100.0
Select_Internal_Energy_3D	169	0	sampling_gpu_power	watts	371.37	136.00	380.00	839.40	28.97	100.0
Sync_Energies_3D	141	0	sampling_gpu_power	watts	375.12	171.00	380.00	443.81	21.07	100.0
PackBuffers3DKernel	27	0	sampling_gpu_power	watts	371.33	225.00	381.00	884.77	29.75	100.0
Temperature_Floor_Kernel	120	0	sampling_gpu_power	watts	375.57	264.00	380.00	142.15	11.92	100.0
UnpackBuffers3DKernel	32	0	sampling_gpu_power	watts	372.91	322.00	380.00	97.44	9.87	100.0

rocprofiler-systems: How To Use (When available in Frontier)

- Load ROCm: module load rocm/6.3.x
- Instrument your binary: rocprof-sys-instrument –o cholla.hydro.frontier.inst -- cholla.hydro.frontier
- Run your application as: rocprof-sys-run -- <instrumented_binary> <application_options>
- Create a configuration file: rocprof-sys -G $PATH_TO_CONFIG/roctprof-sys.cfg
- Relevant configuration options for GPU power profiling

```
ROCPROFSYS_TRACE                   = true
ROCPROFSYS_PROFILE                 = true
ROCPROFSYS_USE_SAMPLING            = true
ROCPROFSYS_USE_PROCESS_SAMPLING    = true
ROCPROFSYS_USE_ROCM_SMI            = true
ROCPROFSYS_SAMPLING_FREQ           = 100
ROCPROFSYS_SAMPLING_GPUS           = $env:HIP_VISIBLE_DEVICES
ROCPROFSYS_SAMPLING_CPUS           = none
```

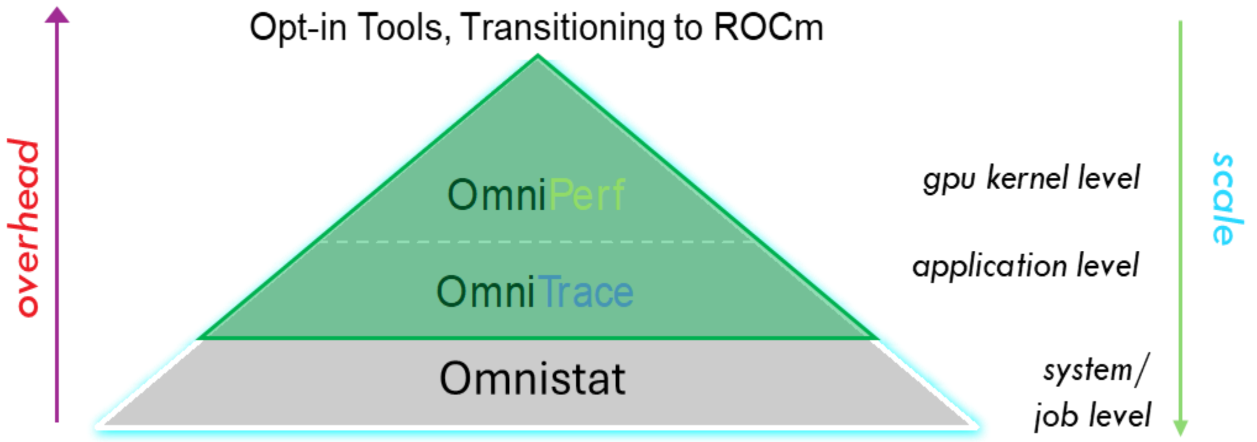
- Set your configuration file: export ROCPROFSYS_CONFIG_FILE=$PATH_TO_CONFIG/rocprof-sys.cfg

Omnistat

Omnistat: Overview

- AMD research tool designed to have lower overhead than profiling tools like Omniperf or Omnitrace, and monitor resource usage at the job level, but can scale up to system level monitoring
- https://github.com/AMDResearch/omnistat

- Provides monitoring of system resources usage, e.g.
  - GPU utilization
  - HBM high water mark
  - GPU power
  - Host memory utilization
  - Network inbound/outbound traffic
    – Slingshot is not yet supported



- Simple invocation by placing a pair of commands at the beginning and at the end of the SLURM script.

- Already used by a team at ORNL to measure energy consumption of the HydraGNN large language model running on 128 Frontier nodes

Scalable Training of Graph Foundation Models for Atomistic Materials Modeling: A Case Study with HydraGNN https://arxiv.org/abs/2406.12909

Omnistat: Summary Report

- Summary report at end of the job showing time profile of resource usage and relevant statistics
- Can help identifying potentially problematic hosts or GPUs
- Potential to provide summary usage statistics of all jobs on a cluster

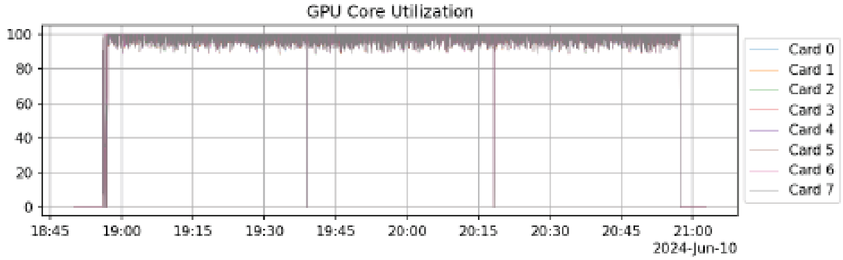
Report Card: JobID = 2012377
Start Time: 2024-06-10 13:50:00
End Time: 2024-06-10 16:03:00

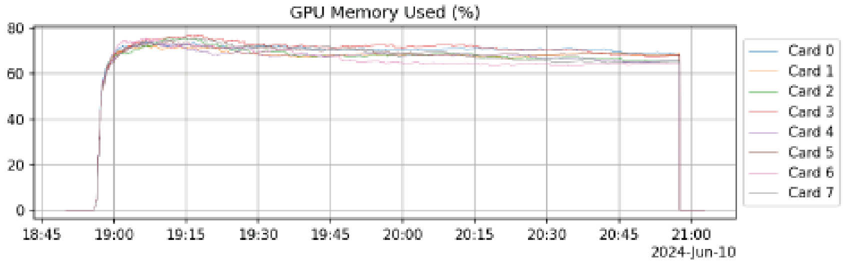
GPU Statistics

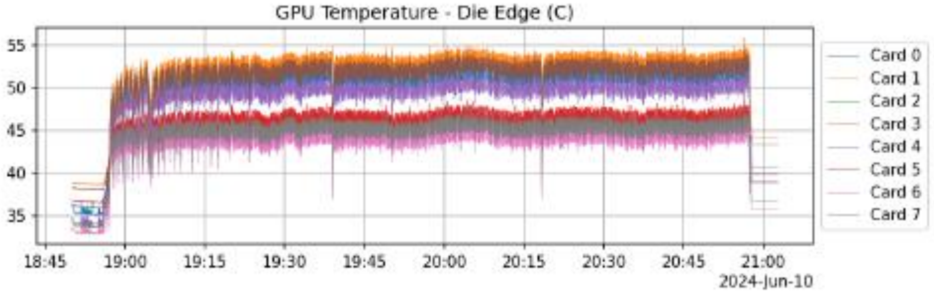
GPU	Utilization (%) Max	Utilization (%) Mean	Memory Use (%) Max	Memory Use (%) Mean	Temperature (C) Max	Temperature (C) Mean	Power (W) Max	Power (W) Mean	Energy (kWh) Total
0	100.00	88.85	99.98	64.33	64.00	49.77	514.00	326.46	92.08
1	100.00	88.85	99.98	62.39	68.00	51.64	0.00	0.00	0.00
2	100.00	88.84	99.98	62.28	59.00	44.24	512.00	324.26	91.51
3	100.00	88.86	99.98	65.02	57.00	45.50	0.00	0.00	0.00
4	100.00	88.84	99.97	61.90	61.00	48.34	522.00	326.06	91.97
5	100.00	88.85	99.98	63.29	65.00	50.79	0.00	0.00	0.00
6	100.00	88.85	99.99	60.91	56.00	43.09	522.00	322.45	91.00
7	100.00	88.85	99.98	62.85	57.00	44.41	0.00	0.00	0.00

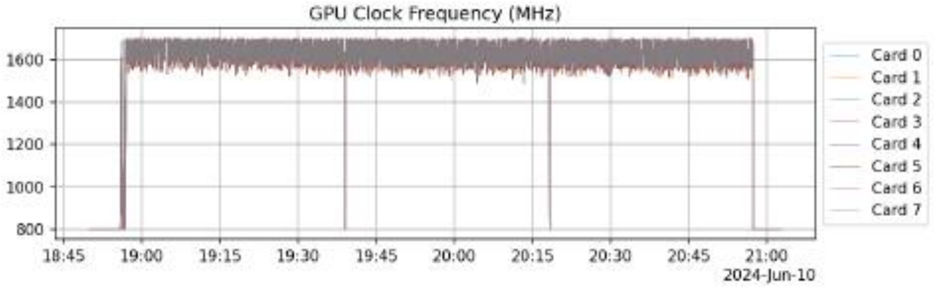
Total GPU Energy Consumed = 366.56 kWh

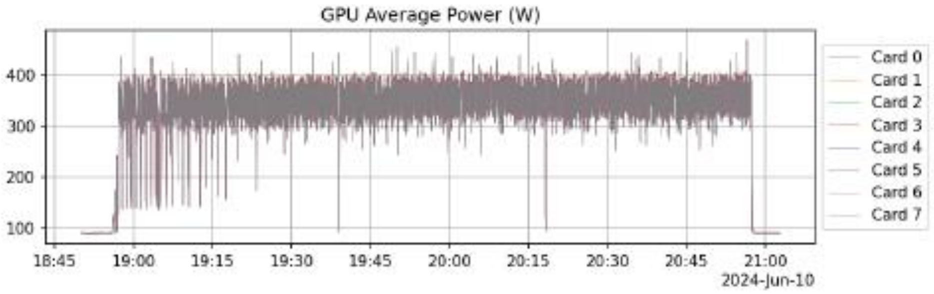
Time Series

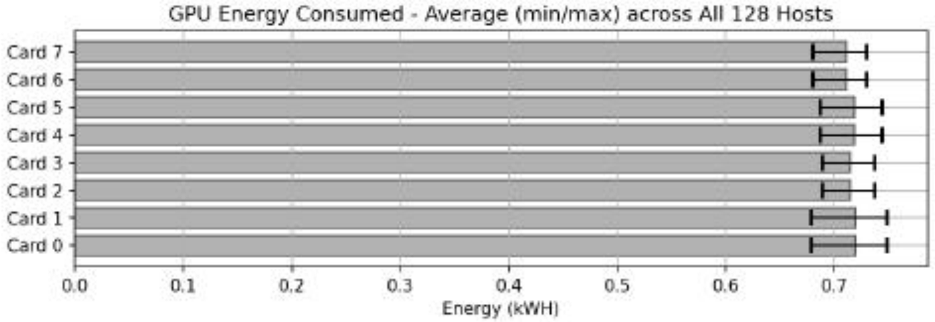












Omnistat: How to use in Frontier

- Example calling Omnistat on a SLURM script in Frontier:

```
# Set your environment (load other modules)
module load ...

# Load Omnistat
ml use /autofs/nccs-svm1_sw/crusher/amdsw/modules
ml omnistat-wrapper

# Start Omnistat – enable data collection
${OMNISTAT_WRAPPER} usermode --start --interval 1

# Run your application
srun ...

# Stop Omnistat – generate summary report and stop data collection
${OMNISTAT_WRAPPER} usermode --stopexporters
${OMNISTAT_WRAPPER} query --interval 1 --job ${SLURM_JOB_ID} --pdf omnistat.${SLURM_JOB_ID}.pdf
${OMNISTAT_WRAPPER} usermode --stopserver
mv /tmp/omnistat/${SLURM_JOB_ID} data_omnistat.${SLURM_JOB_ID}
```

Omnistat: Grafana Real-time Dashboards

- Omnistat output database is compatible with Grafana for interactive visualization
  - Copy the Omnistat output database to your local system and visualize with Grafana
  - Example visualization of six consecutive LINPACK runs on 32 nodes

Power and Energy Optimization

Energy Optimization Using GPU Power Caps

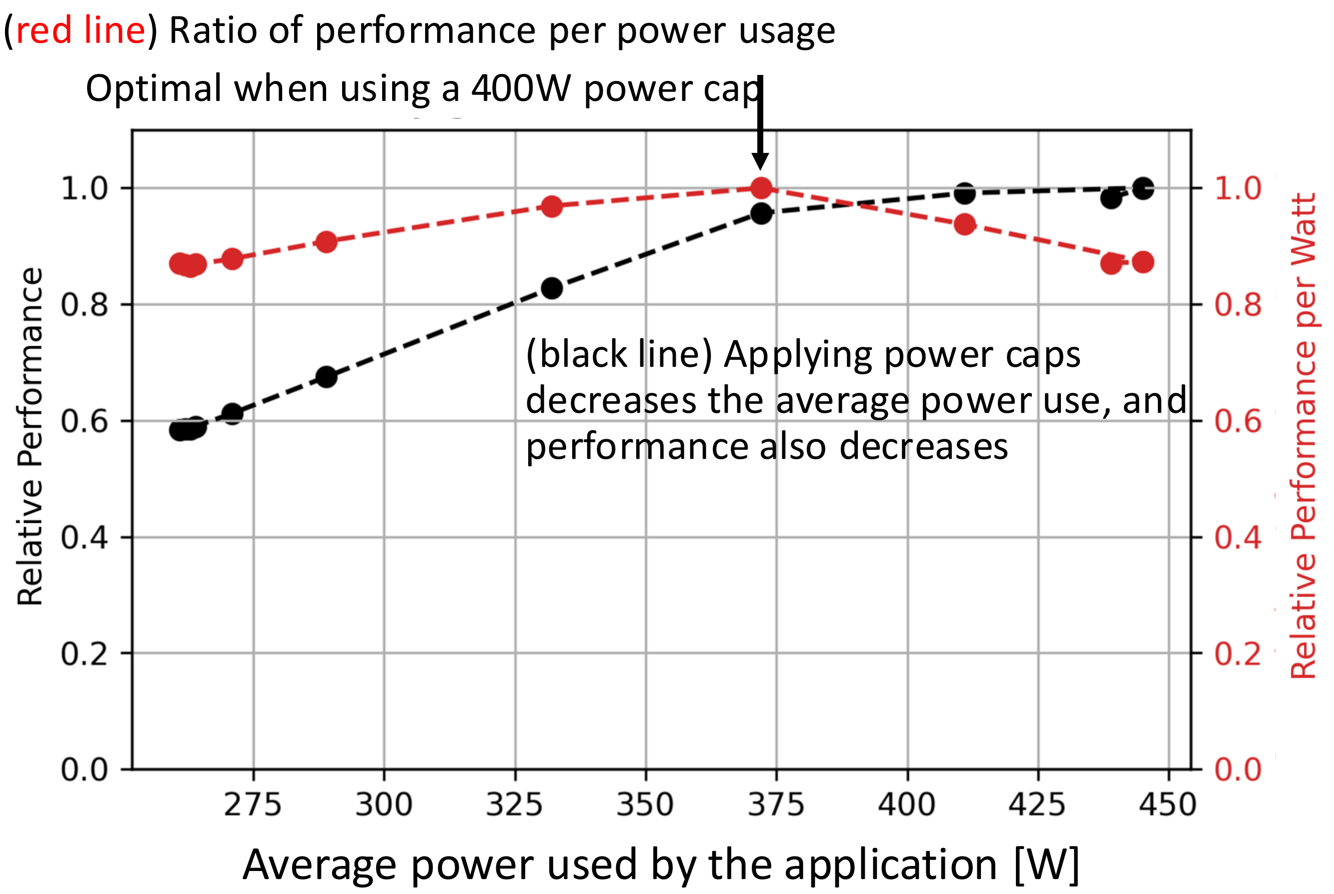
- The GPU power cap can be changed on Frontier through SLURM
  - #SBATCH --gpu-power-cap=<power_cap_in_watts>
  - Sets to power cap for all GPUs in the node, and all nodes for the job.
  - Default MI250X power cap is 560 W

- Iterate over several values for the power cap, measure the application performance and the average power draw

- The performance per watt (red) is optimal when the GPU power is capped

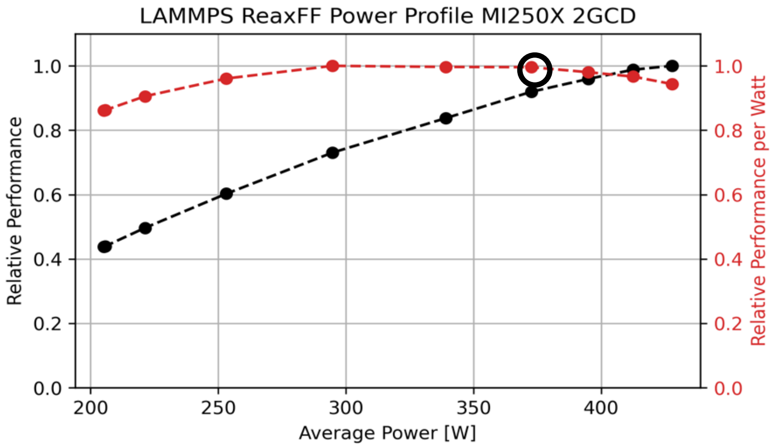
- The performance per watt ratio is proportional to the inverse of the energy used for the run

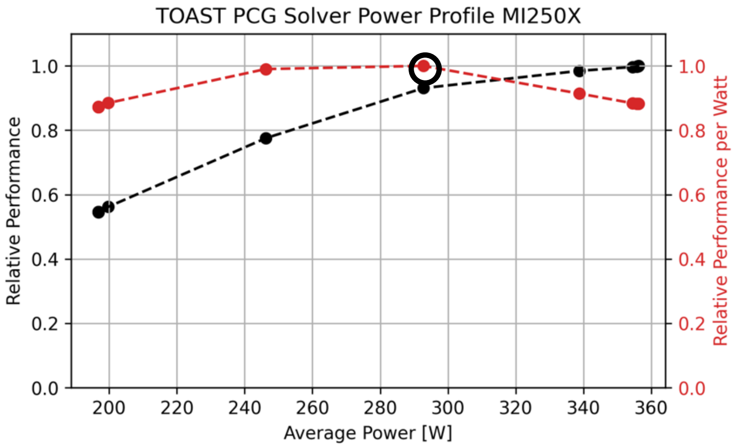
MILC Power Profile MI250X

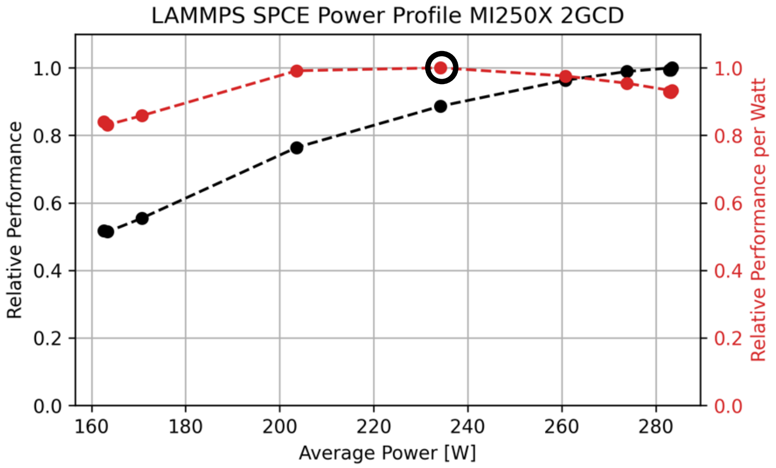


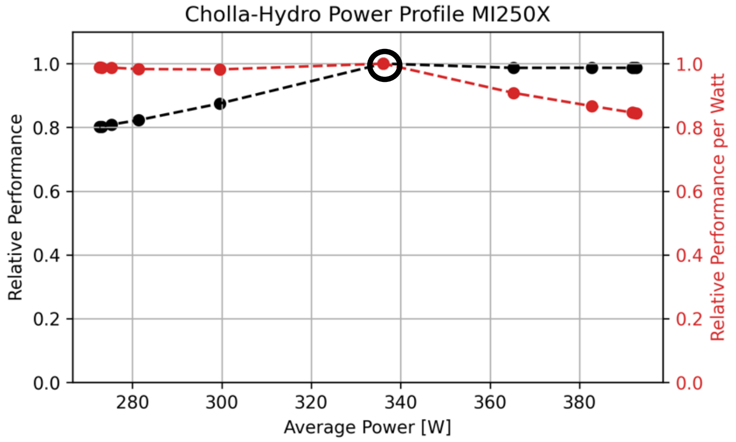
Energy Optimization Using GPU Power Caps

- We observe similar performance per watt curves for other HPC applications









Energy Efficiency: Other Considerations

- Applying power caps can improve the GPU energy efficiency but might not improve the system energy efficiency as other expenses need to be considered (e.g. cooling, network, etc.)

- An alternative is to set GPU frequency caps
  - Through SLURM

    `#SBATCH --gpu-gpu-srange=<min_gpu_clock> - <max_gpu_clock>`

    Set the GPU sclk range in MHz. Default is 500-1700
  - Through a script available on Frontier compute nodes

    `/usr/bin/set_gpu_max_sclk [-g gpu_id] <frequency_in_MHz>`

- Lowering the max GPU frequency for bandwidth-bound applications could lower the GPU energy footprint without significantly impacting performance

- Explore single/mixed precision operations
  - Less runtime is always better!

Summary

Overview of Power Monitoring on Frontier

- Cray PM counters provide node-level power information
  - Report power and energy usage from CPUs, GPUs, Memory; can also set power cap
  - Can query using SLURM, system files, CrayPat, PAPI
    - Updated out-of-band, at 10 Hz
    - SLURM accumulates energy across all nodes
    - System files, CrayPat, and PAPI report per-node information

- RAPL counters provide AMD CPU socket- and core-level power data
  - Can query using Cray PAT and PAPI

- ROCm power management counters provide GPU-level average power
  - Can also set power cap
  - Can query using system files, ROCm-SMI, Omnitrace, Omnistat
    - Updated out-of-band, at 1 kHz
    - Omnitrace can provide application and kernel-level power statistics
    - Omnistat can provide job/system-level statistics
    - Currently, Omnitrace and Omnistat use rocm-smi in the backend for power measurements

Thank you

Ashesh Sharma – Ashesh.Sharma@hpe.com
Bruno Villasenor Alvarez - Bruno.VillasenorAlvarez@amd.com