OMNISTAT

Scale-out cluster telemetry for AMD Instinct GPUs

https://rocm.github.io/omnistat/

Jordà Polo, Karl W. Schulz, Bruno Villasenor, Ashwin Aji, Keith Lowery
AMD Research and Advanced Development
OLCF User Conference Call - September 2025

Outline

- What is this Omnistat thing....?
  - Underlying architecture
  - Support utilities
  - Demo usage on ORNL/Frontier

- Highlight some newer features:
  - CSV exports
  - CU occupancy
  - Hardware counter sampling
  - Additional user-supplied data

Acknowledgements

Omnistat development has been motivated and guided tremendously by multiple ORNL collaborations over the last two years.

Many, many thanks to ORNL personnel (and other internal AMD users)

Massimiliano Lupo Pasini
Jong Youl Choi
Kshitij Mehta
Pei Zhang
David Rogers
Jonghyun Bae
Khaled Z Ibrahim
Prasanna Balaprakash
Woong Shin
Trey White
Arthur F. Lorenzon
Matthias Maiterth
Aditya Kashi
Hao Lu
Nicholson Koukpaizan
Antigoni Georgiadou
Matthew Norman
Wael Elwasif
Michael Matheson
Feiyi Wang
Nicholas Frontiere
Sarp Oral
Thomas Beck
Bronson Messer

"Scalable training of trustworthy and energy-efficient predictive graph foundation models for atomistic materials modeling: a case study with HydraGNN", *Journal of Supercomputing,* 2025.

"Bridging the Gap: User-Centric Energy Monitoring for Policy-Driven Application Optimization in HPC Data Centers", *Sustainable Supercomputing Workshop*, SC25.

Omnistat – what is it?

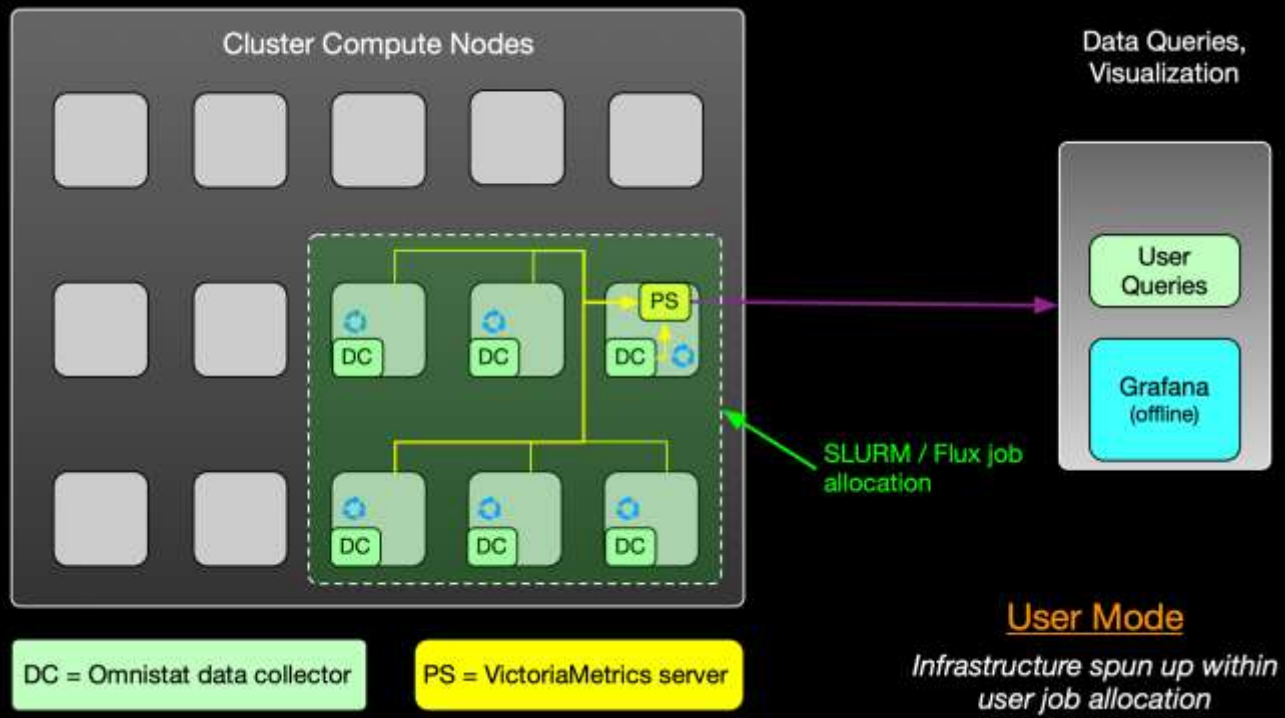
Omnistat is a set of Python utilities and data collectors to support scale-out cluster telemetry targeting AMD Instinct GPUs/APUs:

- Tracks metrics available via AMD SMI interface(s)
  - GPU utilization
  - High-bandwidth memory (HBM) usage
  - GPU power, clock frequencies, thermals, power-caps
  - RAS error counts, throttling events
- Additional metrics/features:
  - application-supplied figures of merit
  - network traffic
  - hardware counters
  - low overhead (target 1% or less)
  - resource manager job tracking (SLURM, Flux, PBSPro)
- Targets two primary use cases
  - system-mode
  - user-mode

Open-source: https://github.com/ROCm/omnistat

Scale-out Infrastructure - User Mode

- User-mode designed entirely for user-space within a single job (e.g SLURM)
  - Targets usage on external supercomputing systems by application teams
  - Omnistat provides plumbing to spin up monitoring infrastructure on all assigned computes
  - Time-series data can be downloaded locally for use with Omnistat Grafana container



- User-mode adopts a "push" model to VictoriaMetrics on a single compute node
- Default push occurs every 5 minutes
- Multi-threaded to support push while sampling new data
- Flask-based REST API for convenience to tear-down (which flushes cached data)

Runtime Configuration File

Default: omnistat/config/omnistat.default
Override via env variable: OMNISTAT_CONFIG

```
[omnistat.collectors]
port = 8001
enable_rocm_smi        = True
enable_amd_smi         = False
enable_ras_ecc         = True
enable_rms             = True
enable_network         = True
enable_vendor_counters = True
enable_rocprofiler     = False

# Path to local ROCM install to access SMI library
rocm_path = /opt/rocm-6.4.1


# List of IPs allowed to query the collector (comma separated).
allowed_ips = 127.0.0.1

[omnistat.collectors.rms]
host_skip = "login.*"
enable_annotations = True
job_detection_mode = file-based
job_detection_file = /tmp/omni_rmsjobinfo

...
[omnistat.usermode]
# settings when spinning up a local victoria-metrics instance within a job (enabled via external_victoria = False)
victoria_binary  = /path-to-victoria-metrics-install/victoria-metrics-prod
victoria_datadir = data_prom
victoria_logfile = vic_server.log

# settings for pushing data to external victoria-metrics instance (enabled via external_victoria = True)
external_victoria = False
external_victoria_endpoint = victoria.endpoint.org
external_victoria_port = 8440
# external_proxy=http://proxy.myorg.org:3128/
```

3. Download a **single-node** VictoriaMetrics server. Assuming a `victoria-metrics` server is not already present on the system, download and extract a precompiled binary from upstream. This binary can generally be stored in any directory accessible by the user, but the path to the binary will need to be known during the next section when configuring user-mode execution. Note that VictoriaMetrics provides a larger number binary releases and we typically use the `victoria-metrics-linux-amd64` variant on x86_64 clusters.

MI250 - Example gauge metrics exported via Omnistat on Frontier

Node-level metrics

```
rocm_num_gpus 8.0
omnistat_network_rx_bytes{device_class="net",interface="hsn0"} 3.442352842e+09
omnistat_network_tx_bytes{device_class="net",interface="hsn0"} 4.136354034e+09
...
omnistat_vendor_energy_joules{vendor="cray"} 2.682251788e+09
omnistat_vendor_memory_energy_joules{vendor="cray"} 3.60993687e+08
omnistat_vendor_cpu_energy_joules{vendor="cray"} 2.50639981e+08
rmsjob_info{batchflag="0",jobid="456322",jobstep="-1",nodes="1",partition="batch",type="slurm",user="karls"} 1.0
```

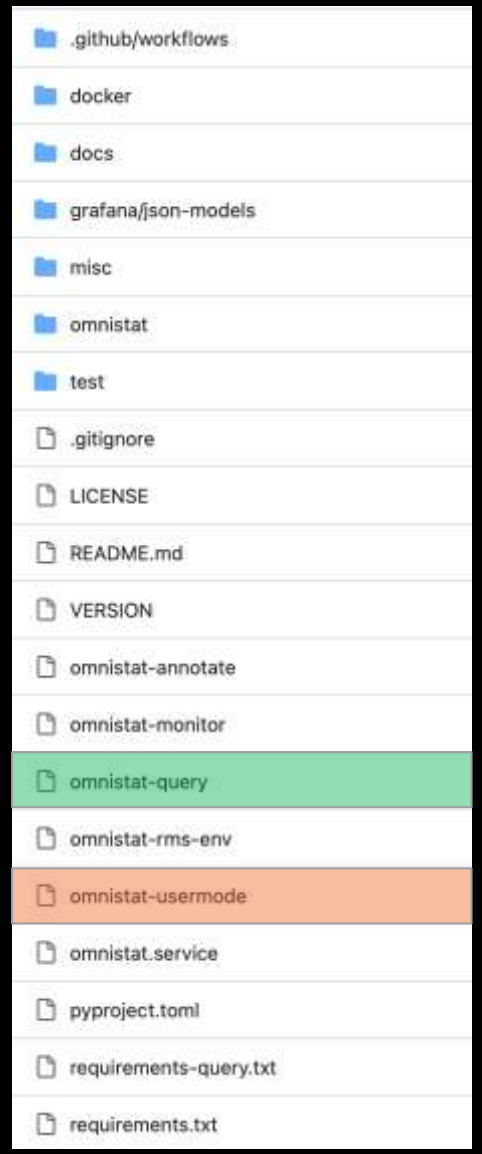
GPU metrics (Card 0)

```
rocm_version_info{card="0",driver_ver="6.12.12",schema="1.0",type="AMD Instinct MI250X / MI250",vbios="113-D65210V-073"} 1.
rocm_temperature_celsius{card="0",location="edge"} 34.0
rocm_temperature_memory_celsius{card="0",location="vram"} 49.0
rocm_average_socket_power_watts{card="0"} 90.0
rocm_sclk_clock_mhz{card="0"} 800.0
rocm_mclk_clock_mhz{card="0"} 1600.0
rocm_vram_total_bytes{card="0"} 6.870269952e+010
rocm_vram_used_percentage{card="0"} 0.016
rocm_vram_busy_percentage{card="0"} 0.0
rocm_utilization_percentage{card="0"} 0.0
rocm_ras_sdma_correctable_count{card="0"} 0.0
rocm_ras_sdma_uncorrectable_count{card="0"} 0.0
rocm_ras_gfx_correctable_count{card="0"} 0.0
rocm_ras_gfx_uncorrectable_count{card="0"} 0.0
rocm_ras_mmhub_correctable_count{card="0"} 0.0
rocm_ras_mmhub_uncorrectable_count{card="0"} 0.0

omnistat_vendor_accel_energy_joules{card="0",vendor="cray"} 5.82187477e+08
omnistat_vendor_accel_power_watts{card="0",vendor="cray"} 107.0
```

...repeat for 8 GPUs...

- *resulting `rocm_*` GPU metrics are the same whether using rocm-smi or amd-smi*
- *GPU card index maps to what you expect with `HIP_VISIBLE_DEVICES`*

Omnistat - Primary CLI Tools for User Mode



- To enable data collection in User Mode, a user needs to add commands in their job script to spin up data collectors on each assigned compute node
- an `omnistat-usermode` utility is provided to automate this process (queries local resource manager environment within running job)

```
omnistat-usermode --start --interval 1
omnistat-usermode --stop
```

Example setting up and tearing down data collection at 1 sec intervals

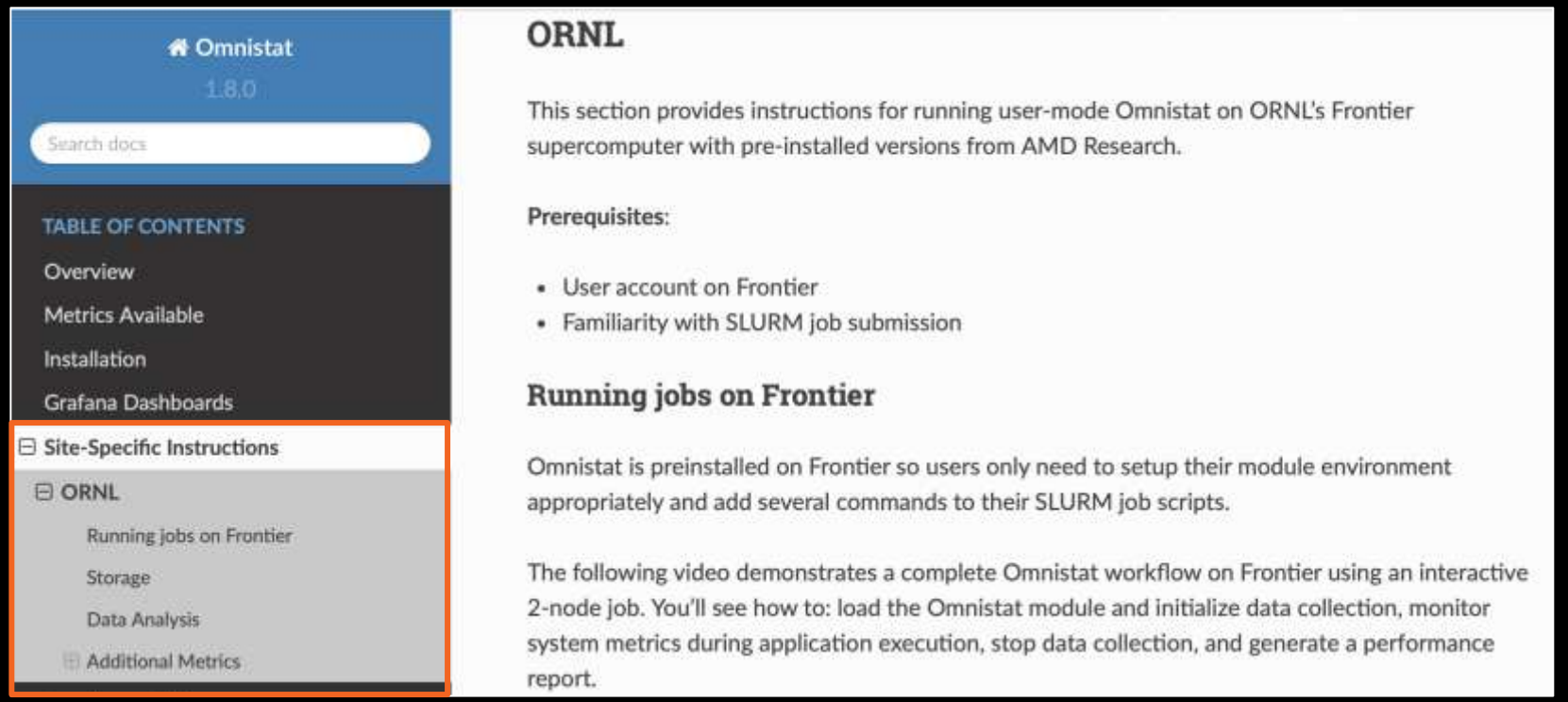
- an `omnistat-query` utility can be used to interrogate data collected at end of a job

```
omnistat-query --job ${SLURM_JOB_ID} --interval 5
```

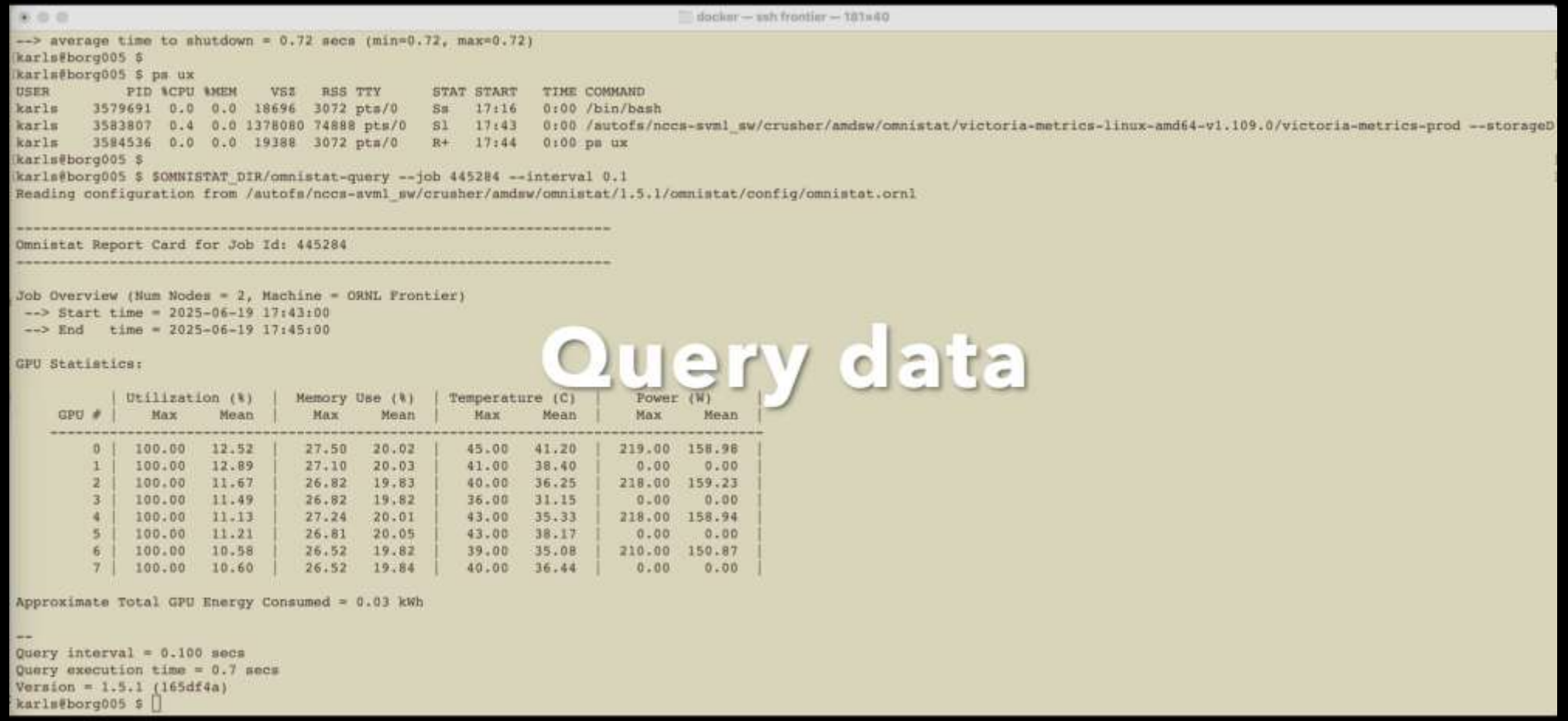
DEMOS

- *Note that Frontier-specific usage is also available in online docs*

https://rocm.github.io/omnistat/sites.html#ornl



ORNL Demo #1 - Omnistat hello world



https://github.com/ROCm/omnistat/wiki/Hello-World

Note: current Frontier module path for latest Omnistat: ml use /sw/frontier/amdsw/modulefiles

Time-series database storage - ORNL/Frontier

For concurrency management, chosen storage location requires flock support

ORNL Storage Location(s)
✓ /tmp
✓ Lustre ($MEMBERWORK, $PROJWORK)
✗ $HOME / NFS *(but can move here after the fact)*

Default is Lustre: /lustre/orion/$(SLURM_JOB_ACCOUNT)/scratch/$(USER)/omnistat/$(SLURM_JOB_ID)

In last demo, we overrode via env variable: `export OMNISTAT_VICTORIA_DATADIR=/tmp/omnistat/${SLURM_JOB_ID}`

```
karls@borg005 $ ls  /tmp/omnistat/445284/
cache  data  flock.lock  indexdb  metadata  snapshots  tmp

karls@borg005 $ du -sh   /tmp/omnistat/445284/
1.5M /tmp/omnistat/445284/
```

this holds all your time-series data; want to keep for post-processing

Omnistat - application impact

Generally, recommend 0.1 sec sampling or higher

- "push" method allows us to achieve higher sampling frequencies

```
[t008-001: 13:22:22] Received shutdown request
[t008-001: 13:22:22] Initiating final data push...
[t008-001: 13:22:22] Pushing local node telemetry to VictoriaMetrics endpoint -> http://borg019:909

[t008-001: 13:22:22] Metrics pushed successfully!
[t008-001: 13:22:22]
[t008-001: 13:22:22] --> Sampling interval          = 0.1000 (secs)
[t008-001: 13:22:22] --> Total # of samples         = 6234
[t008-001: 13:22:22] --> Average time/sample        = 0.0024 (secs)
[t008-001: 13:22:22] --> Total data pushes          = 2
[t008-001: 13:22:22] --> Average push duration      = 0.0031 (secs)
[t008-001: 13:22:22] --> Data collection duration   = 10.6492 (mins)
[t008-001: 13:22:22] --> Base memory use at start   = 34.234 MB
[t008-001: 13:22:22] --> Memory growth at stop      = 60.895 MB
```

Linpack impact example using GPUs and **all** host CPU cores

		user-space omnistat polling enabled				
	No collector	10 secs	1 secs	0.1 secs	0.05 secs	0.01 secs
Run #	GFLop/s	GFLop/s	GFLop/s	GFLop/s	GFLop/s	GFLop/s
1	3697200	3703700	3690700	3694900	3681600	3665900
2	3707400	3703000	3695900	3684900	3684800	3661800
Average	3702300	3703350	3693300	3689900	3683200	3663850
Impact -->	--	0.028%	-0.243%	-0.335%	-0.516%	-1.039%

ORNL Borg: 256 GPUs

Frontier Convenience Wrapper

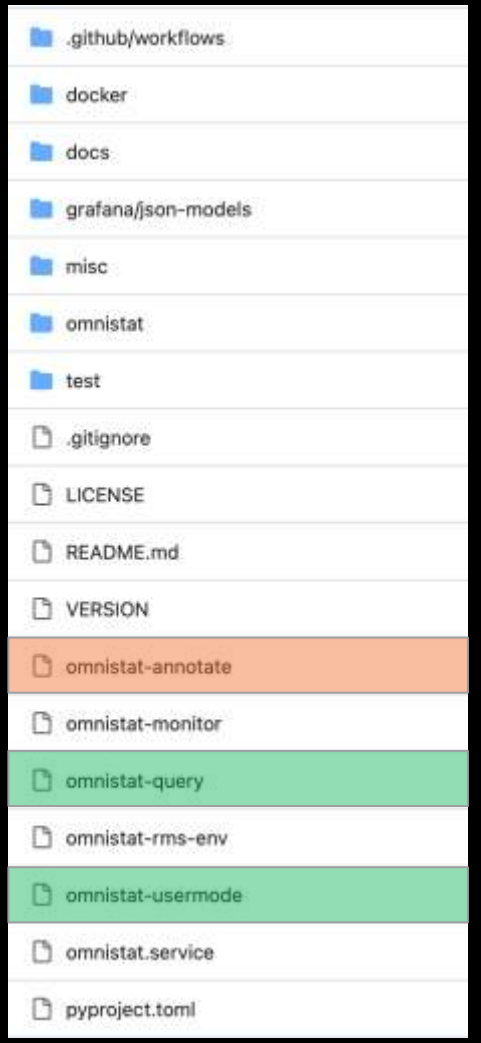
- Omnistat module at ORNL relies on the cray-python module to run commands highlighted in previous demo
- For teams running their own Python stack who may wish to avoid any pollution with an application's Python environment, a wrapper utility is also available for use on Frontier

```
$ omnistat-ornl  -h

Wrapper tool to setup python environment and execute Omnistat utilities.

Usage: omnistat-ornl [-h,--help] <mode> [arguments...]

Options:
  -h, --help          Show usage info and exit

Modes:
  usermode             Run omnistat-usermode
  query                Run omnistat-query
  rms                  Run omnistat-rms-env (used for job steps)

Arguments:
   [arguments]         Additional command line args to pass to desired mode
```

Users can run this wrapper script or access the utilities directly.

ORNL Demo #2 - Multiple apps and local data exploration

```
#!/bin/bash
#SBATCH -J omnistat
#SBATCH -o rochpl.16nodes.%j.out -A ven114
#SBATCH -N 16 -n 128 -t 00:55:00 -S 0

# Setup and launch Omnistat (wrapper version)
ml use /sw/frontier/amdsw/modulefiles
ml omnistat-wrapper
export OMNISTAT_VICTORIA_DATADIR=/tmp/omnistat/${SLURM_JOB_ID}
${OMNISTAT_WRAPPER} usermode --start --interval 0.1

# Let's run some Linpack...
for SIZE in 102400 358400 768000; do
    ${OMNISTAT_DIR}/omnistat-annotate --mode start --text  "RocHPL N=${SIZE}"
    ./run_hpl_16nodes.sh --size ${SIZE}
    ${OMNISTAT_DIR}/omnistat-annotate --mode stop
    sleep 5
done

# That was fun, let's run some HPCG
for iter in `seq 1 2`; do
    ${OMNISTAT_DIR}/omnistat-annotate --mode start --text  "RocHPCG iter=#${iter}"
    ./run_hpcg_16nodes.sh
     ${OMNISTAT_DIR}/omnistat-annotate --mode stop
    sleep 5
done

# Tear down Omnistat
${OMNISTAT_WRAPPER} usermode --stopexporters
${OMNISTAT_WRAPPER} query --interval 0.1 --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}
```



More details on that expanded query tool example...

```
${OMNISTAT_WRAPPER} query --interval 0.1 --job ${SLURM_JOB_ID} --pdf omnistat.${SLURM_JOB_ID}.pdf
```

stdout

```
--------------------------------------------------------------------------------
Omnistat Report Card for Job Id: 456402
--------------------------------------------------------------------------------

Job Overview (Num Nodes = 16, Machine = ORNL Frontier)
 --> Start time = 2025-07-14 20:05:31
 --> End   time = 2025-07-14 20:14:45
 --> Duration   = 553 secs

GPU Statistics:


         | Utilization (%)  |  Memory Use (%)  | Temperature (C)  |    Power (W)     | Energy (kWh) |
   GPU # |    Max     Mean  |    Max     Mean  |    Max     Mean  |    Max     Mean  |    Total     |
  -------------------------------------------------------------------------------------------------
       0 |  100.00   21.67  |   58.93   41.78  |   50.00   41.33  |  322.00  209.07  |   5.40e-01   |
       1 |  100.00   21.76  |   58.23   41.78  |   52.00   44.37  |    0.00    0.00  |   0.00e+00   |
       2 |  100.00   21.71  |   58.24   41.79  |   48.00   39.33  |  315.00  206.80  |   5.28e-01   |
       3 |  100.00   22.06  |   58.24   41.79  |   49.00   41.16  |    0.00    0.00  |   0.00e+00   |
       4 |  100.00   21.75  |   57.94   41.57  |   54.00   41.38  |  321.00  207.15  |   5.36e-01   |
       5 |  100.00   22.04  |   57.94   41.56  |   54.00   43.08  |    0.00    0.00  |   0.00e+00   |
       6 |  100.00   21.75  |   57.94   41.57  |   48.00   39.38  |  311.00  205.82  |   5.31e-01   |
       7 |  100.00   21.92  |   57.94   41.57  |   51.00   40.18  |    0.00    0.00  |   0.00e+00   |

Vendor Energy Data:
  ------------------------------------------------------------------
  Approximate Total Memory Energy Consumed = 2.03e-01 kWh ( 7.58 %)
  Approximate Total CPU    Energy Consumed = 2.25e-01 kWh ( 8.39 %)
  Approximate Total Accel  Energy Consumed = 2.14e+00 kWh (79.60 %)
  ------------------------------------------------------------------
  Approximate Total Node Energy Consumed   = 2.68e+00 kW
```

New in v1.6.0

omnistat.456402.pdf

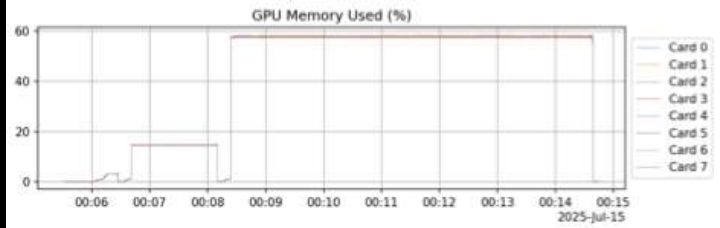
Omnistat Report Card: JobID = 456402
Start Time: 2025-07-14 20:05:31
End Time: 2025-07-14 20:14:45
Duration: 553 secs

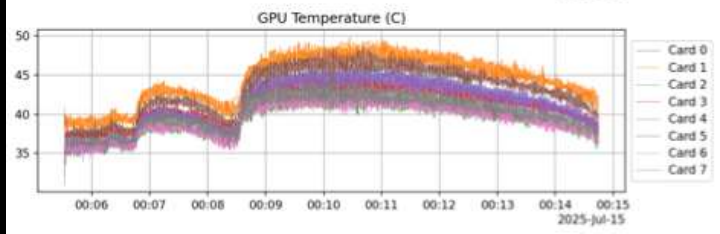
GPU Statistics

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	21.67	58.93	41.78	50.00	41.33	322.00	209.07	5.40e-01
1	100.00	21.76	58.23	41.78	52.00	44.37	0.00	0.00	0.00e+00
2	100.00	21.71	58.24	41.79	48.00	39.33	315.00	206.80	5.28e-01
3	100.00	22.06	58.24	41.79	49.00	41.16	0.00	0.00	0.00e+00
4	100.00	21.75	57.94	41.57	54.00	41.38	321.00	207.15	5.36e-01
5	100.00	22.04	57.94	41.56	54.00	43.08	0.00	0.00	0.00e+00
6	100.00	21.75	57.94	41.57	48.00	39.38	311.00	205.82	5.31e-01
7	100.00	21.92	57.94	41.57	51.00	40.18	0.00	0.00	0.00e+00

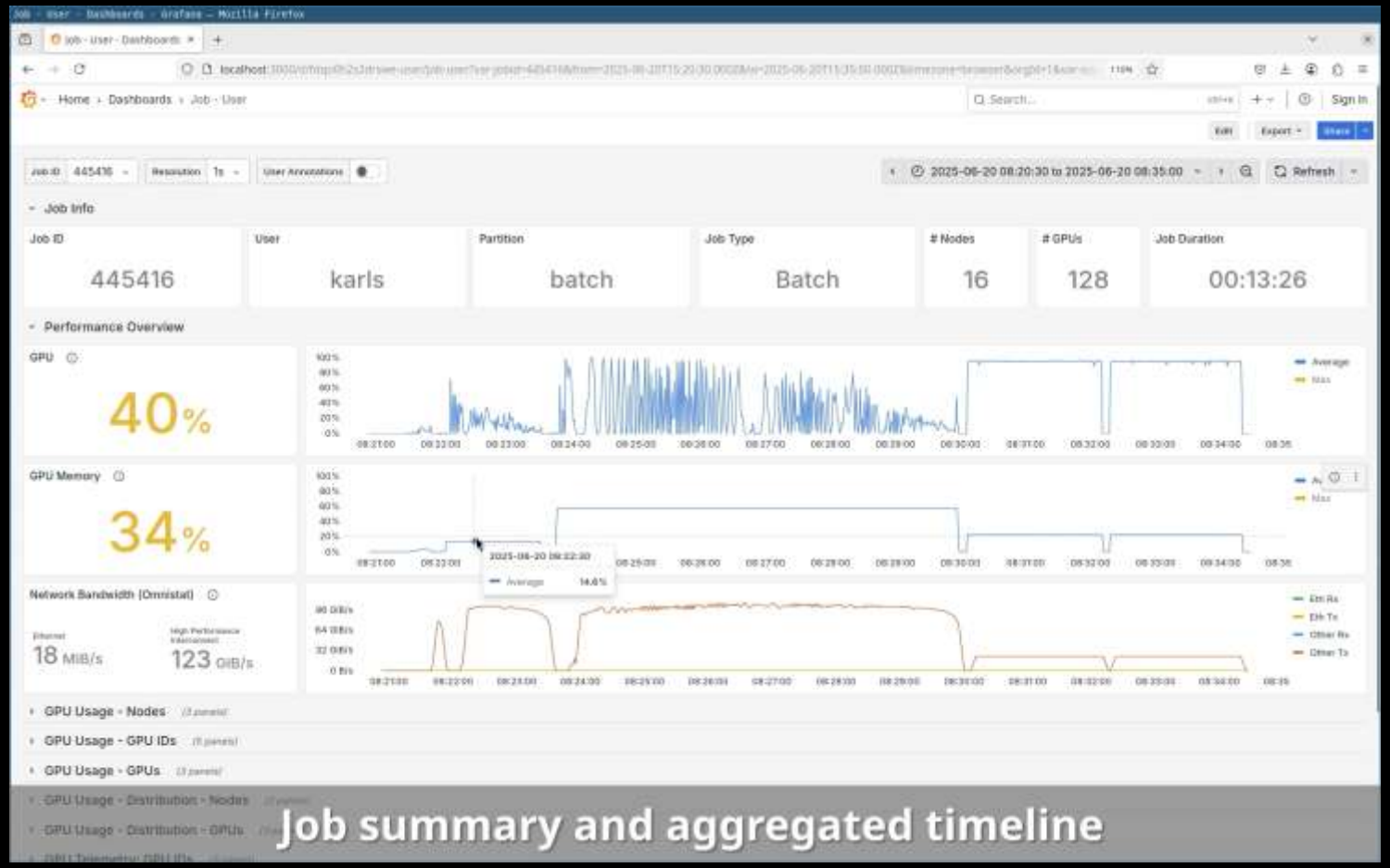
Vendor Energy Data Totals

Type	Energy Consumed (kWh)	% of Total
Memory	2.03e-01	7.58 %
CPU	2.25e-01	8.39 %
Accelerator (GPU)	2.14e+00	79.60 %
Total	2.68e+00	100.00 %





ORNL Demo #2 - Download data for local Grafana exploration



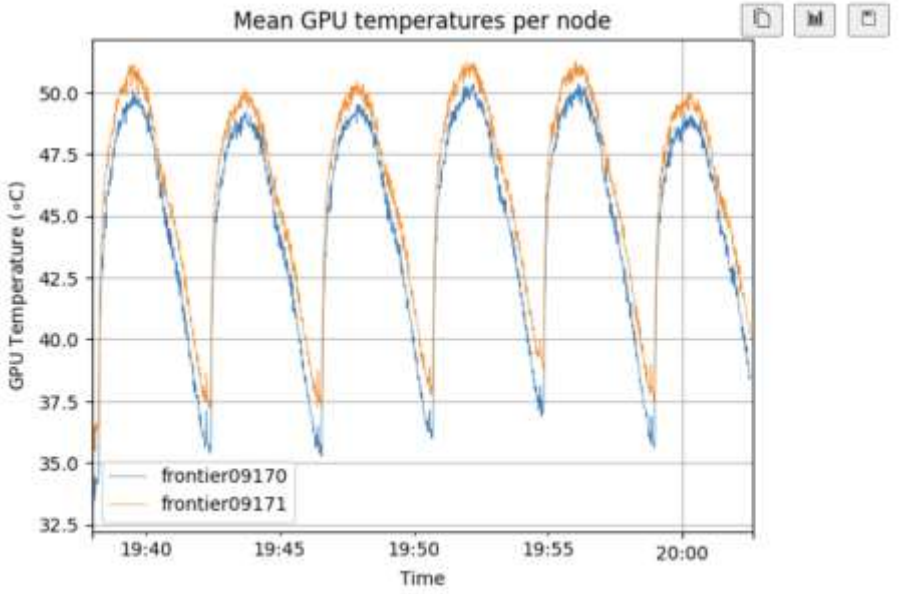
https://github.com/ROCm/omnistat/wiki/Local-Job-Dashboard

Accessing telemetry data directly

```
omnistat-query --job <jobid> --interval 1.0 --export <export-dir>
```

```python
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("data.csv", header=[0, 1, 2], index_col=0)
df.index = pd.to_datetime(df.index)

# Create a new dataframe with node-level averages (averaged across all GPUs per node)
node_mean_df = df["rocm_temperature_celsius"].T.groupby(level=['instance']).mean().T

# Plot results from first two compute nodes
node_mean_df[node_mean_df.columns[0]].plot(linewidth=0.5)
node_mean_df[node_mean_df.columns[1]].plot(linewidth=0.5)
plt.title("Mean GPU temperatures per node")
plt.xlabel("Time"); plt.ylabel("GPU Temperature ($\circ$C)")
plt.grid(); plt.legend()
plt.show()
```

✓ 0.1s Python



- You can also use the omnistat-query tool to export data into CSV format
- Useful for loading up into pandas and plotting directly
- For more detailed instructions, please consult the user guide: Exporting time series data

New in v1.6: CU Occupancy

- Optional collector that reads values from `sysfs`
- Takes the number of in-flight waves and translates them into a number of compute units based on the maximum number of waves per CU (e.g. in MI2xx 104 CUs x 32 = 3328)
- Not a measure of performance
  - Occupancy doesn't tell us how waves are distributed
  - We don't know the factors constraining occupancy
- Current implementation in Omnistat aggregates CU occupancy per GPU, including all processes

Monitoring vector add kernel with increasing problem sizes

New in v1.8: Hardware counter sampling

Automate use of ROCprofiler-SDK to collect counters

- Enables deeper GPU performance insights with minimal setup and impact on performance
- Device collection: continuously samples GPU activity, independent of running workloads or kernel execution

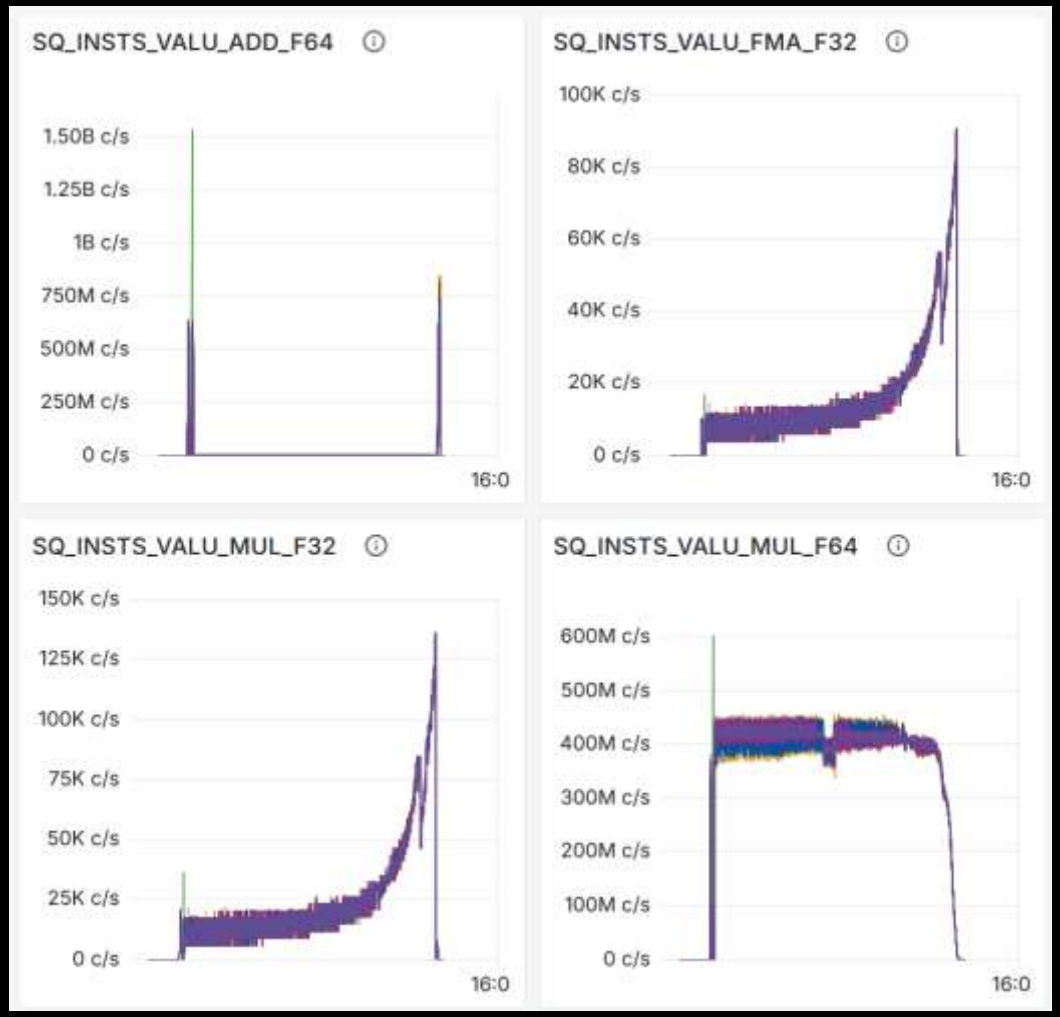
Challenges

- Limit on number of concurrent counters you can define at a time (GPU block dependent)

  → *Multiplexing allows rotating through counter sets to overcome hardware limits*

- High-level metrics derived from counters

  → *Predefined sets of counters and aggregation*



Instruction counters for different operations and data types

Hardware counter collection: constant sampling

Customize config file to enable counter collection:

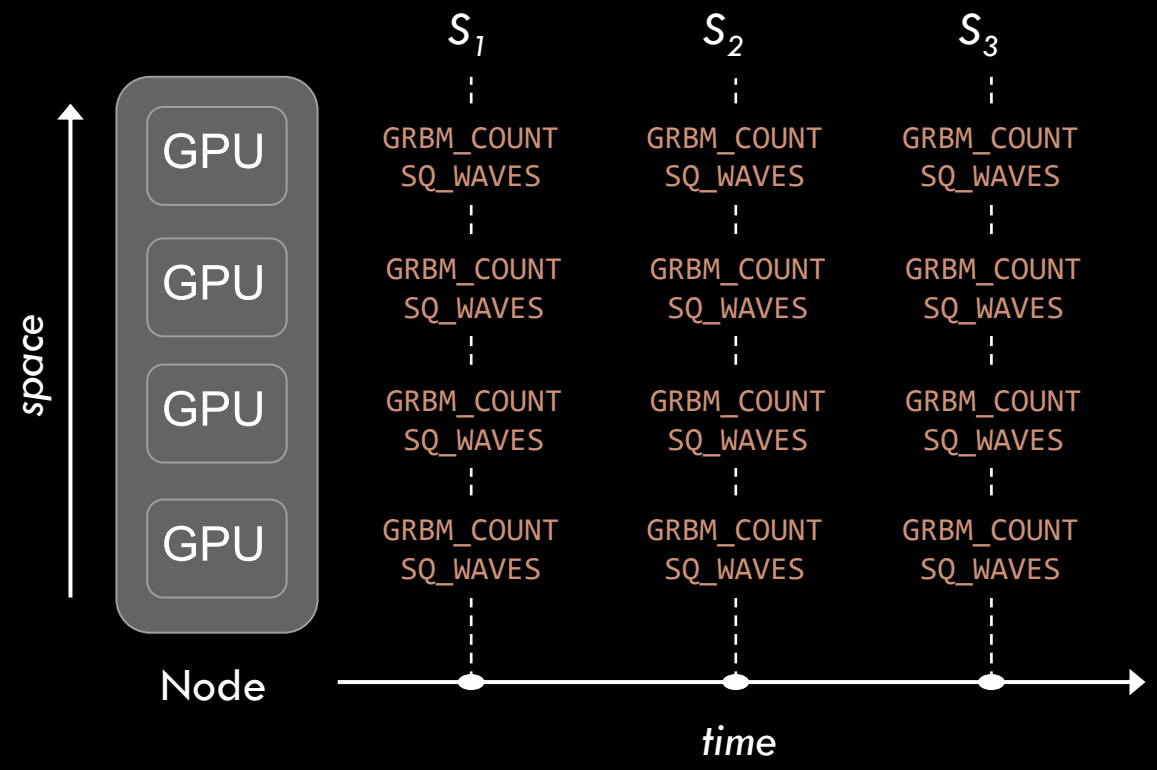
```
/sw/frontier/amdsw/omnistat/1.8.0/omnistat/config/omnistat.ornl
```

GitHub link to omnistat.ornl config file

```
[omnistat.collectors]
enable_rocprofiler = True

[omnistat.collectors.rocprofiler]
profile = example

[omnistat.collectors.rocprofiler.example]
sampling_mode = constant
counters = ["GRBM_COUNT", "SQ_WAVES"]
```

Set variable in application environment:

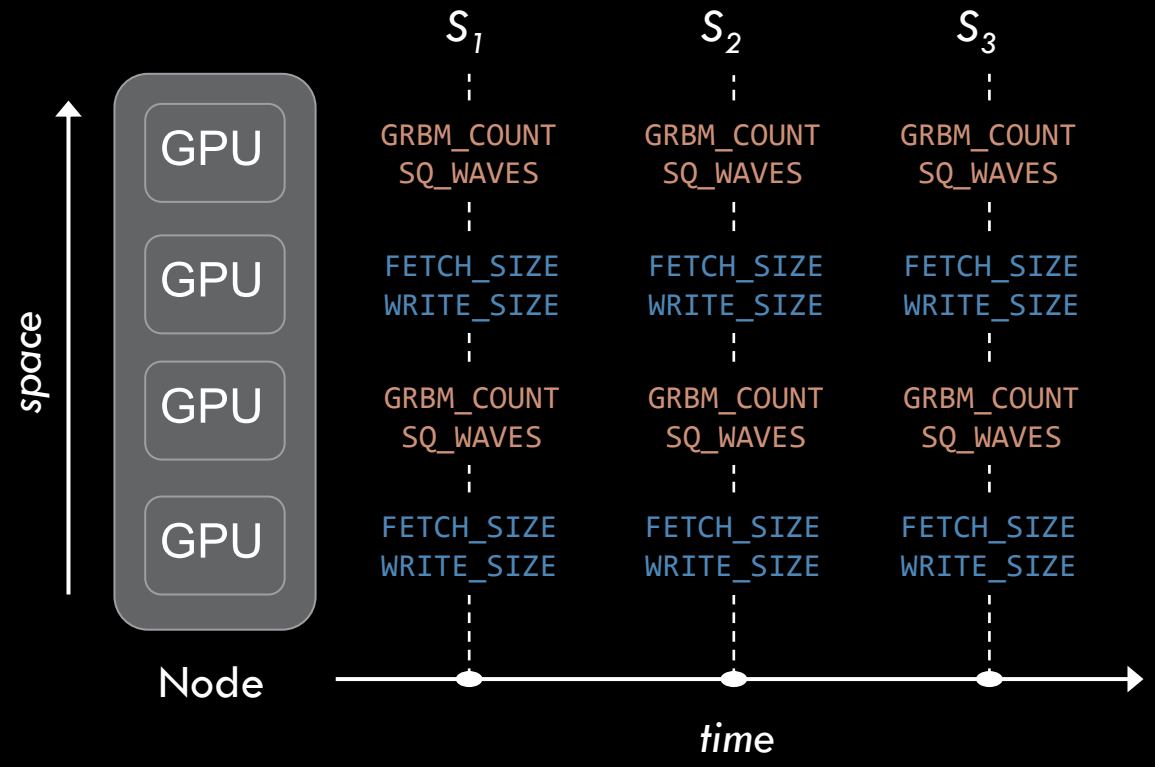
```
HSA_TOOLS_LIB=$ROCM_PATH/lib/librocprofiler64.so
```



Hardware counter collection: GPU ID sampling

- Multiplexing in space (GPU IDs)
- Nested lists can be used to assign multiple sets of counters per GPU

```
[…]

[omnistat.collectors.rocprofiler]
profile = example_gpuid

[omnistat.collectors.rocprofiler.example_gpuid]
sampling_mode = gpu-id
counters = [
        ["GRBM_COUNT", "SQ_WAVES"],
        ["FETCH_SIZE", "WRITE_SIZE"]
    ]
```



Predefined profiles for Frontier

- FLOPS
  - 16 instruction counters: `SQ_INSTS_VALU_(ADD|MUL|TRANS|FMA|MFMA_MOPS)_(F64|F32|F16|BF16)`
    - FP64 (vector, matrix)
    - FP32 (vector, matrix)
    - FP16 (vector, matrix)
    - BF16 (matrix)
  - Only up to 8 counters per GPU at a time
  - Multiplex
    - GPU IDs 0, 2, 4, 6: FP32 and FP64 vector
    - GPU IDs 1, 3, 5, 7: FP16 vector and all matrix

→

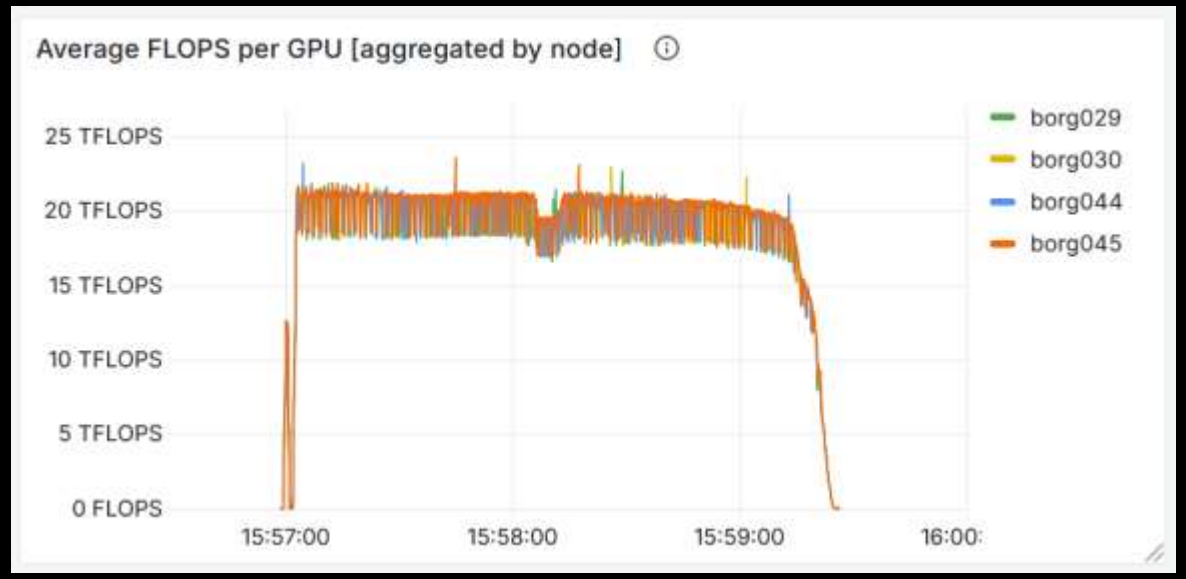
```
[omnistat.collectors.rocprofiler]
profile = flops_gpuid
```

- HBM Bandwidth
  - 2 derived counters: `(FETCH|WRITE)_SIZE`
  - Constant sampling

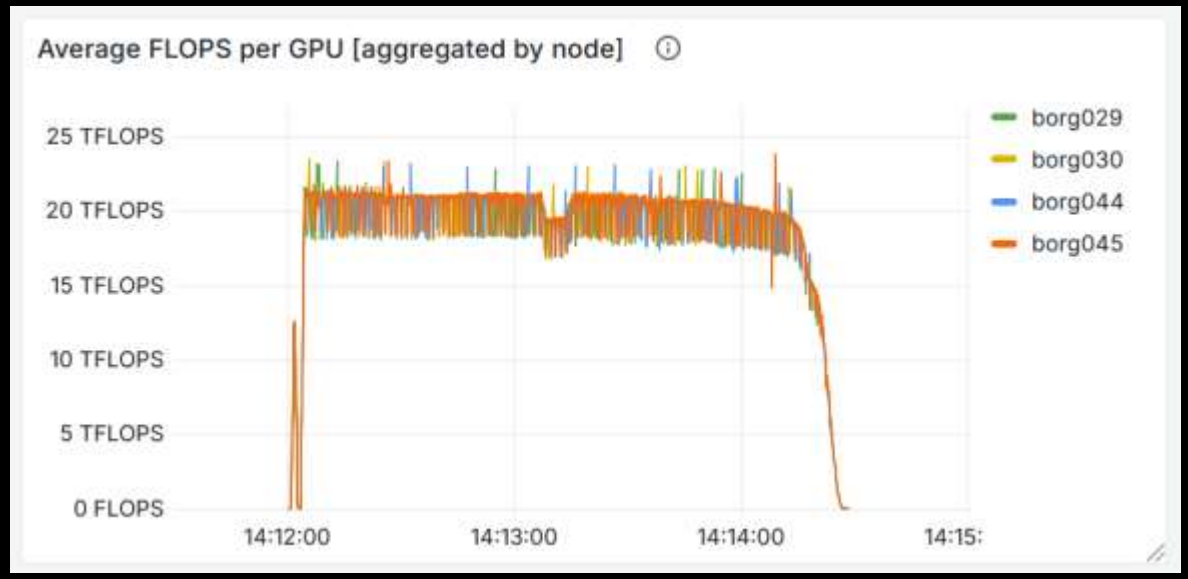
→

```
[omnistat.collectors.rocprofiler]
profile = hbm
```

Example: HPL – FLOPS



Constant sampling with custom profile
8 counters (FP32 and FP64)



GPU ID sampling with `flops_gpuid` profile
16 counters (all floating point instructions)

	Constant sampling	GPU ID sampling
HPL report	617.8 TFLOPS	617.5 TFLOPS
Omnistat estimate	615.3 TFLOPS	615.0 TFLOPS

Example: HBM Bandwidth

HPL

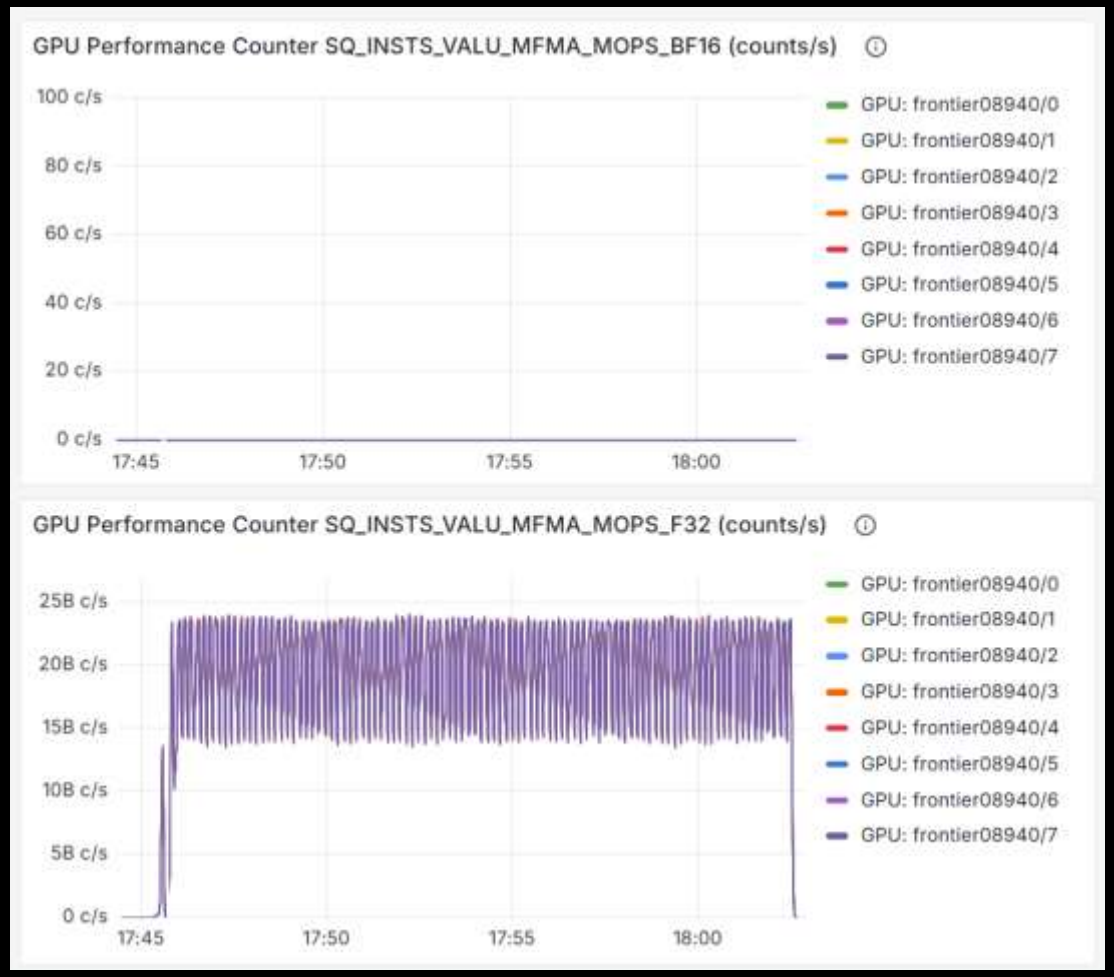
Pytorch
(fine-tuning)

Reads

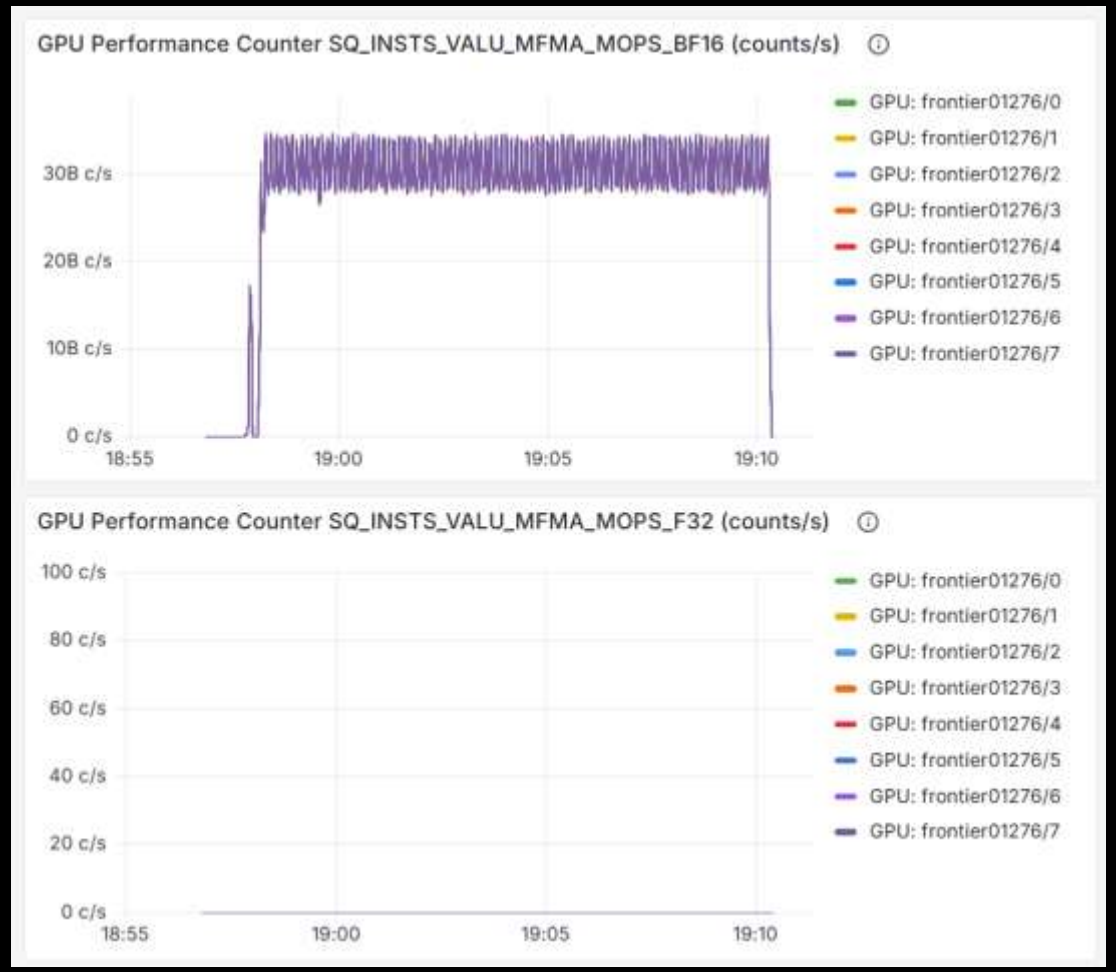
Writes

Example: HydraGNN – Exploring mixed-precision

FP32 baseline



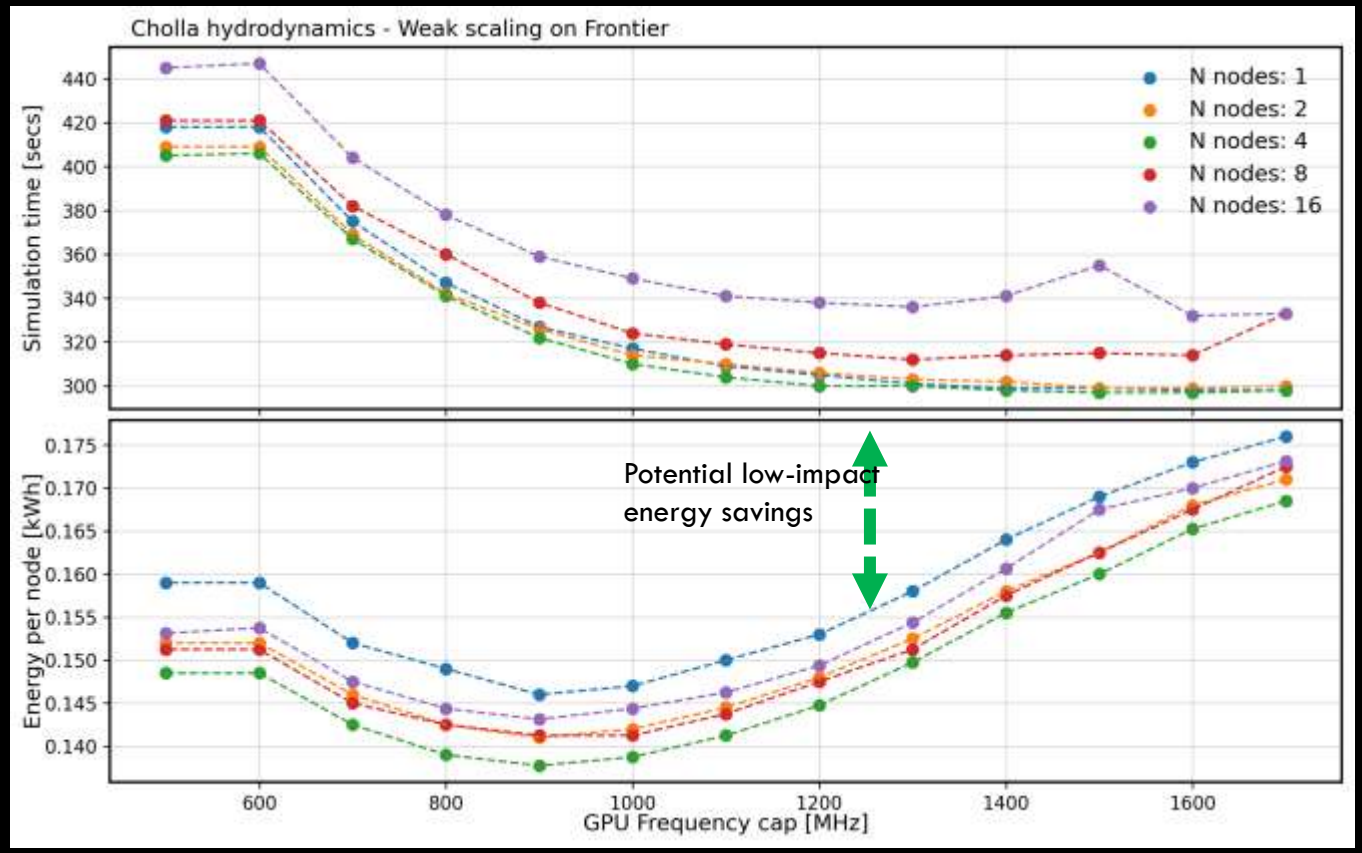
BF16



Hardware counter sampling with Omnistat to confirm lower-precision datatype usage in Pytorch

Real world example: Omnistat collection with data export

Examine energy and runtime impacts as GPU frequency cap varies



Courtesy: Bruno Villasenor Alvarez, AMD

Cholla hydrodynamics code

- *GPU frequency cap varied from 500 Mhz to 1700 Mhz*
- *Weak scaling study performed from 1 to 16 nodes on Frontier*
- *Omnistat monitoring node-level energy as a function of time for each run, delta between stop/start used to quantify total energy per run*

One final demo: cloud-based - consider minor change to startup

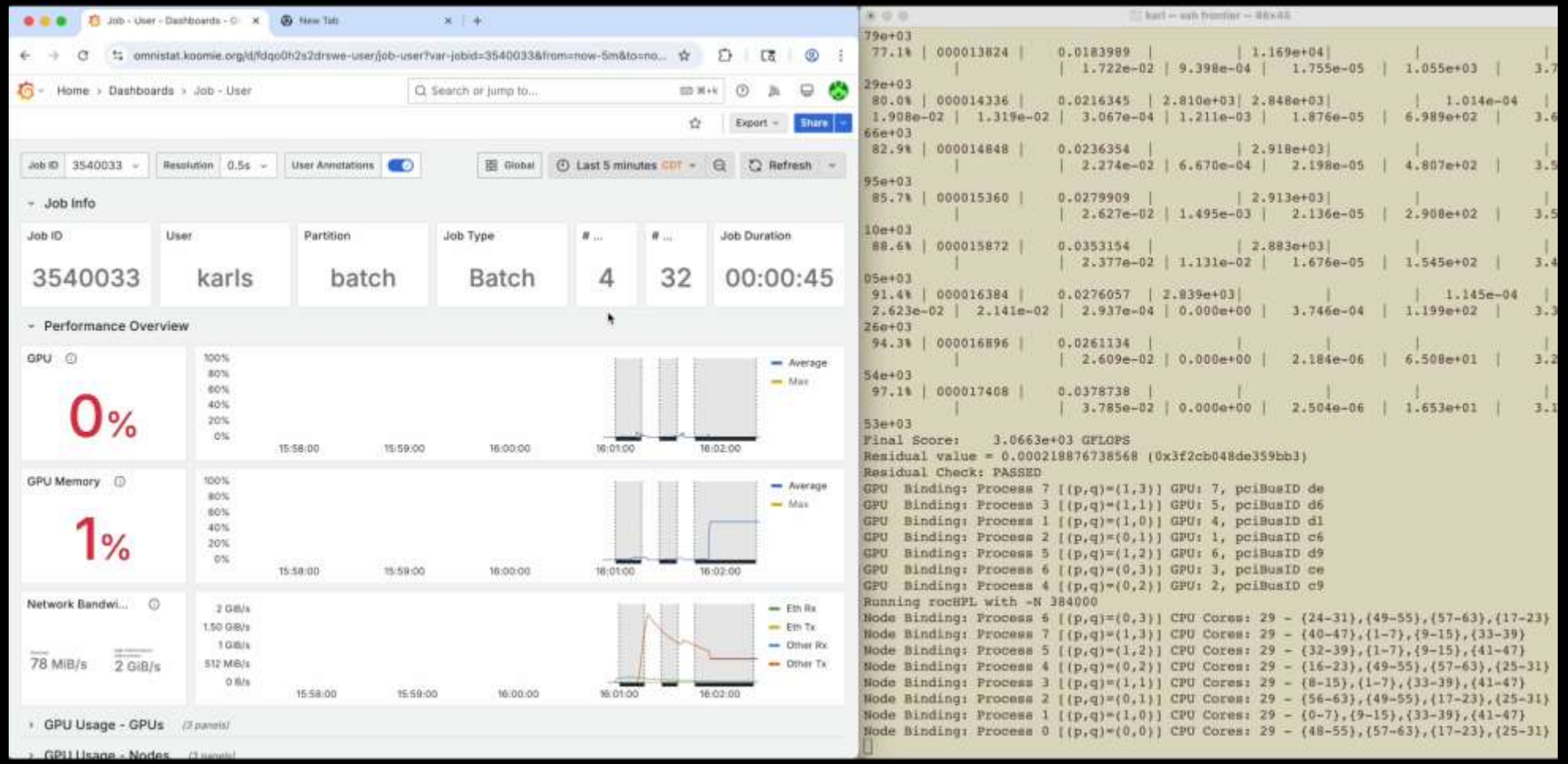
```
#!/bin/bash
#SBATCH -J omnistat
#SBATCH -o rochpl.16nodes.%j.out -A ven114
#SBATCH -N 16 -n 128 -t 00:55:00 -S 0

# Setup and launch Omnistat (wrapper version)
ml use /autofs/nccs-svm1_sw/crusher/amdsw/modules
ml omnistat-wrapper
export OMNISTAT_VICTORIA_DATADIR=/tmp/omnistat/${SLURM_JOB_ID}
export OMNISTAT_CONFIG=$OMNISTAT_DIR/omnistat/config/omnistat.ornl.external
${OMNISTAT_WRAPPER} usermode --start --interval 0.1 --pushinterval 1


# Let's run some Linpack...
for SIZE in 102400 358400 768000; do
    ${OMNISTAT_DIR}/omnistat-annotate --mode start --text  "RocHPL N=${SIZE
    ./run_hpl_16nodes.sh --size ${SIZE}
    ${OMNISTAT_DIR}/omnistat-annotate --mode stop
    sleep 5
done

# That was fun, let's run some HPCG
for iter in `seq 1 2`; do
    ${OMNISTAT_DIR}/omnistat-annotate --mode start --text  "RocHPCG iter=#${iter}"
    ./run_hpcg_16nodes.sh
     ${OMNISTAT_DIR}/omnistat-annotate --mode stop
    sleep 5
done

# Tear down Omnistat
${OMNISTAT_WRAPPER} usermode --stop
```

send data directly to external cloud instance via proxy every 1 minute

```
[omnistat.usermode]
external_victoria = True
external_victoria_endpoint = omnistat.koomie.org
external_victoria_port = 8440
external_proxy=http://proxy.ccs.ornl.gov:3128/
```

Ping Jorda/Karl if you would like to use this option on Frontier

Demo external push - consider minor change to startup



https://github.com/ROCm/omnistat/wiki/External-Push

Application Figures of Merit (FOM) with Telemetry

Many apps have a FOM we care about - *wouldn't it be nice if we could overlay that with telemetry as a function of time?*

Consider HPCG iterative example...

stdout

```
Performing (at least) 69 CG sets in 52.0 seconds ...
CG set  1 / 69    4476.1765 GFlop/s     (559.5221 GFlop/s per process)     1%    51.3 sec left
CG set  2 / 69    4477.5082 GFlop/s     (559.6885 GFlop/s per process)     2%    50.6 sec left
CG set  3 / 69    4475.6103 GFlop/s     (559.4513 GFlop/s per process)     4%    49.9 sec left
CG set  4 / 69    4475.1108 GFlop/s     (559.3888 GFlop/s per process)     5%    49.2 sec left
CG set  5 / 69    4474.9748 GFlop/s     (559.3718 GFlop/s per process)     7%    48.5 sec left
CG set  6 / 69    4473.7517 GFlop/s     (559.2190 GFlop/s per process)     8%    47.8 sec left
CG set  7 / 69    4472.9295 GFlop/s     (559.1162 GFlop/s per process)    10%    47.1 sec left
CG set  8 / 69    4471.3407 GFlop/s     (558.9176 GFlop/s per process)    11%    46.4 sec left
CG set  9 / 69    4470.3629 GFlop/s     (558.7954 GFlop/s per process)    13%    45.7 sec left
...
```

underlying C++

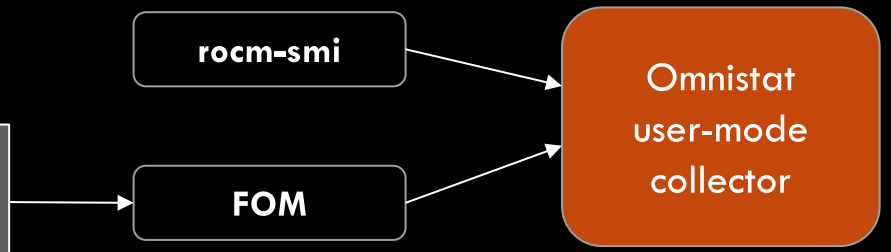
```
        printf("CG set %0d / %0d %7.4lf GFlop/s (%7.4lf GFlop/s per process) %d%c %0.1
               actualCgSets + 1,
               numberOfCgSets,
               gflops,
               gflops / A.geom->size,
               (int)((double)(actualCgSets + 1) / numberOfCgSets * 100.0),
               c,
               total_runtime - times[0] > 0.0 ? total_runtime - times[0] : 0.0);
```

can we log this whilst running?

Application Figures of Merit (FOM) with Telemetry

- Although not yet advertised in docs, Omnistat user-mode collector provides an endpoint to supply custom FOMs
  - "name" and "value" supplied as JSON
  - timestamp encoded at time of receipt

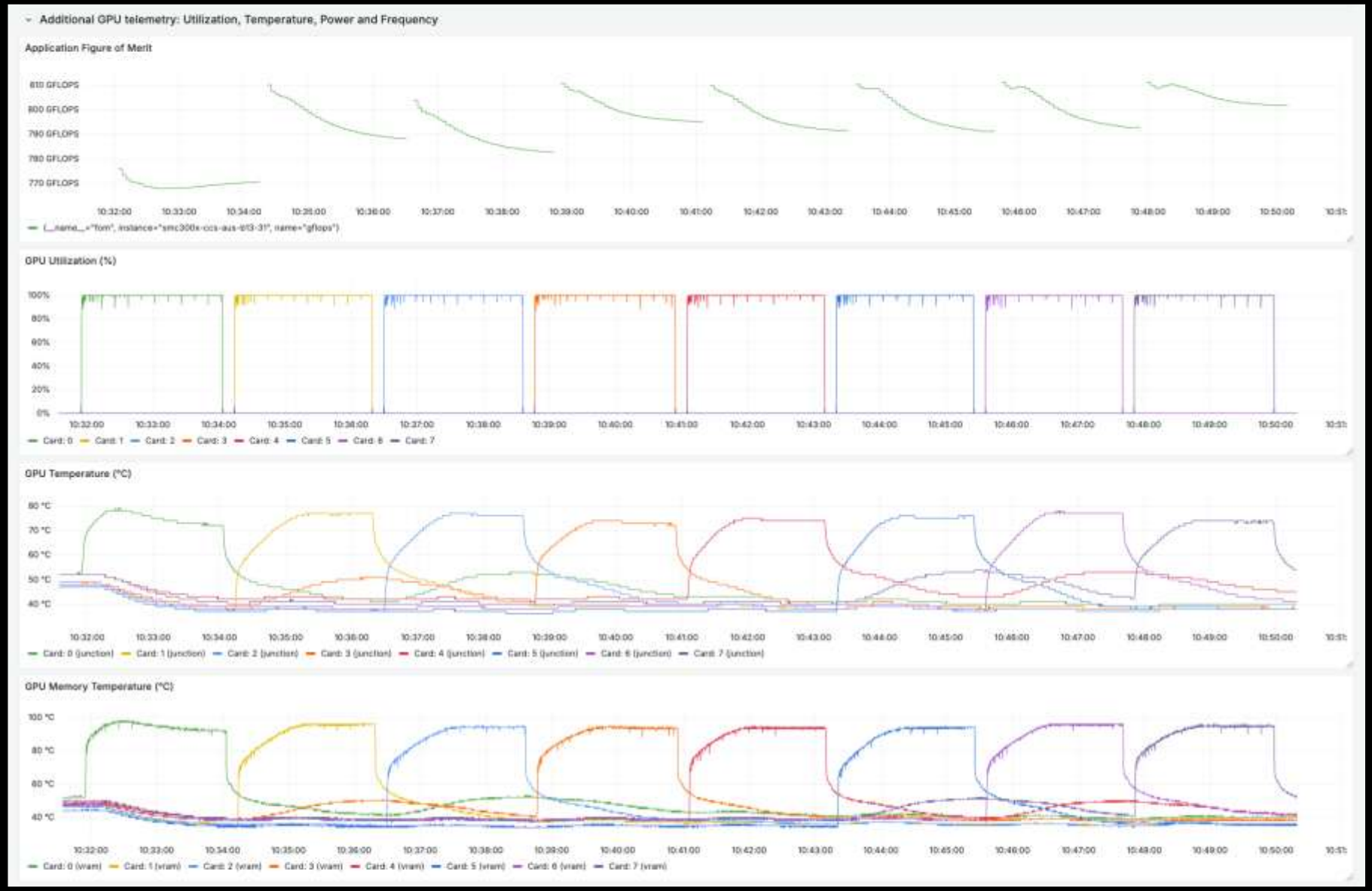
```
$ curl -X POST http://localhost:8001/fom  \
         -H "Content-Type: application/json" \
         -d '{"name": "gflops", "value": 511.264069}'
```



- Note: does require underlying code modification...
  - have mostly been using libcurl for C/C++

Omnistat with Figure-of-Merit (FOM) logging

- rocHPCG example (MI325X)
- single GPU case execution
  - iterated over each GPU
  - FOM == GFLOPs
- Dashboards display FOMs and telemetry together



Omnistat Summary

- Open-source, scale-out telemetry tool developed over the last 2 years
- Targets administrative 24x7 collection (system-mode) and user-mode collection on large clusters
- Designed for large scale: used by AI developers on up to 68K GPUs on Frontier
- Pre-installed for use on Frontier

- *We welcome suggestions for future enhancements and collaboration opportunities…*
- *Feel free to ping us with questions/issues:*

karl.schulz@amd.com, jorda.polo@amd.com

Courtesy: Pasini, et. al, *J. Supercomputing* 81, 618 (2025)

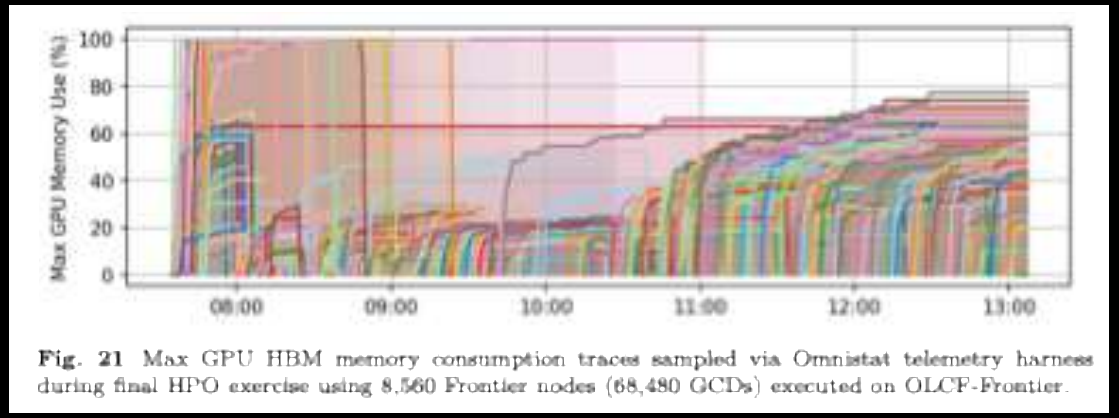


Fig. 21 Max GPU HBM memory consumption traces sampled via Omnistat telemetry harness during final HPO exercise using 8,560 Frontier nodes (68,480 GCDs) executed on OLCF-Frontier.

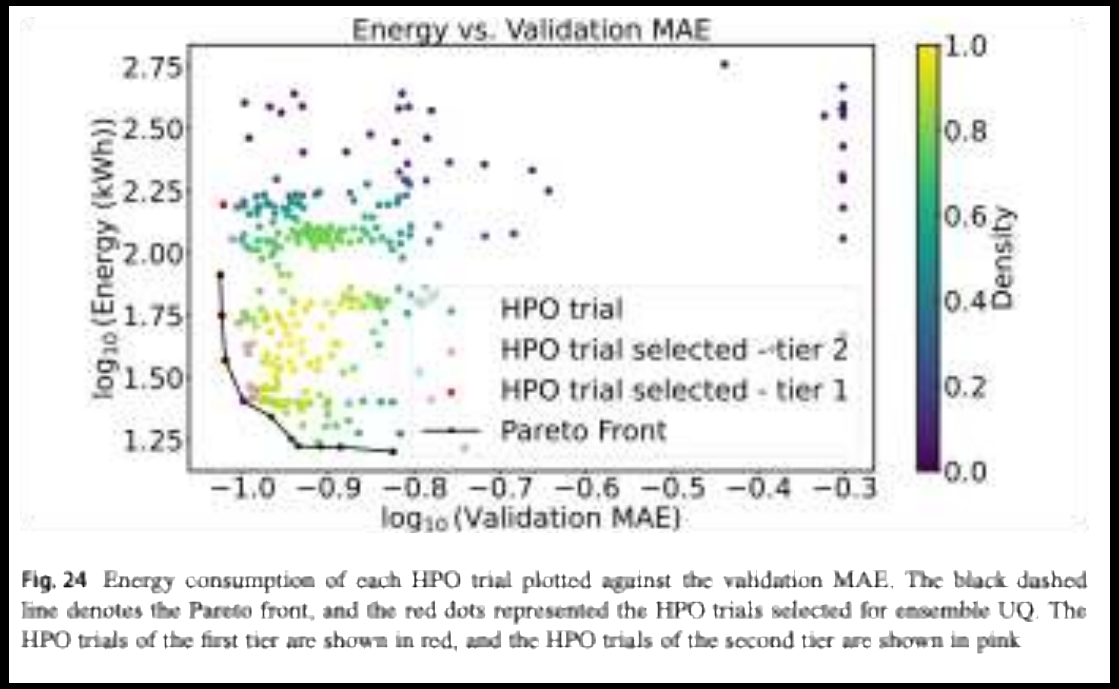


Fig. 24 Energy consumption of each HPO trial plotted against the validation MAE. The black dashed line denotes the Pareto front, and the red dots represented the HPO trials selected for ensemble UQ. The HPO trials of the first tier are shown in red, and the HPO trials of the second tier are shown in pink

Additional Discussion / Questions?