ZeroSum:

User Space Monitoring of Resource Utilization and Contention on Heterogeneous HPC Systems

Kevin A. Huck

Oregon Advanced Computing Institute for Science and Society (OACISS)

PDF version of this talk: https://tinyurl.com/zerosum-pop-2025

UNIVERSITY OF OREGON

For more information:
khuck@cs.uoregon.edu, tau-bugs@cs.uoregon.edu
https://github.com/UO-OACISS/zerosum

Who am I?

- Working in HPC space since ~2003
- Mostly performance tools for US DOE grants/projects
  - SciDAC Computer Science Institutes (PERC, PERI, SUPER, RAPIDS, RAPIDS2)
  - X-Stack program
  - ECP – Exascale Computing Project
- Developing / maintaining TAU https://tau.uoregon.edu
  - Highly portable, scalable measurement/analysis for HPC
- Designer / Developer of APEX https://github.com/UO-OACISS/apex
  - Highly portable, scalable measurement/analysis for asynchronous HPC
- Designer / Developer of ZeroSum https://github.com/UO-OACISS/zerosum
  - User space monitoring

How did this project start?

- April 2023, I was invited to the Dagstuhl seminar: “Driving HPC Operations With Holistic Monitoring and Operational Data Analytics”
- https://www.dagstuhl.de/23171
- Mostly facility/sysadmin focus
- I presented my performance analysis perspective on monitoring (users)
  - *Why* do users monitor?
  - *How* do users monitor?
  - What do users *want* to monitor?
  - What is *lacking*?
- Advocating for the user space
- Discussion inspired ZeroSum

Performance Analysis Perspective on Monitoring

Three main classes of performance optimizations:

1. Algorithmic replacement (usually a high level of difficulty)
   - E.g. replace $O(n^2)$ with O(nlogn)
   - Can involve data structure changes, new dependencies, major rewrites
2. Code optimization (usually a medium difficulty)
   - Improve cache reuse, reduce stalls (branching, instructions, I/O, etc)
3. Optimized launch configuration (low(?) difficulty, high embarrassment potential)
   - Misconfiguration
   - Wrong assumptions from another system/application
   - Changes to system policies/defaults (e.g. reserved cores)

Motivation: Why do users (want to) monitor at runtime?

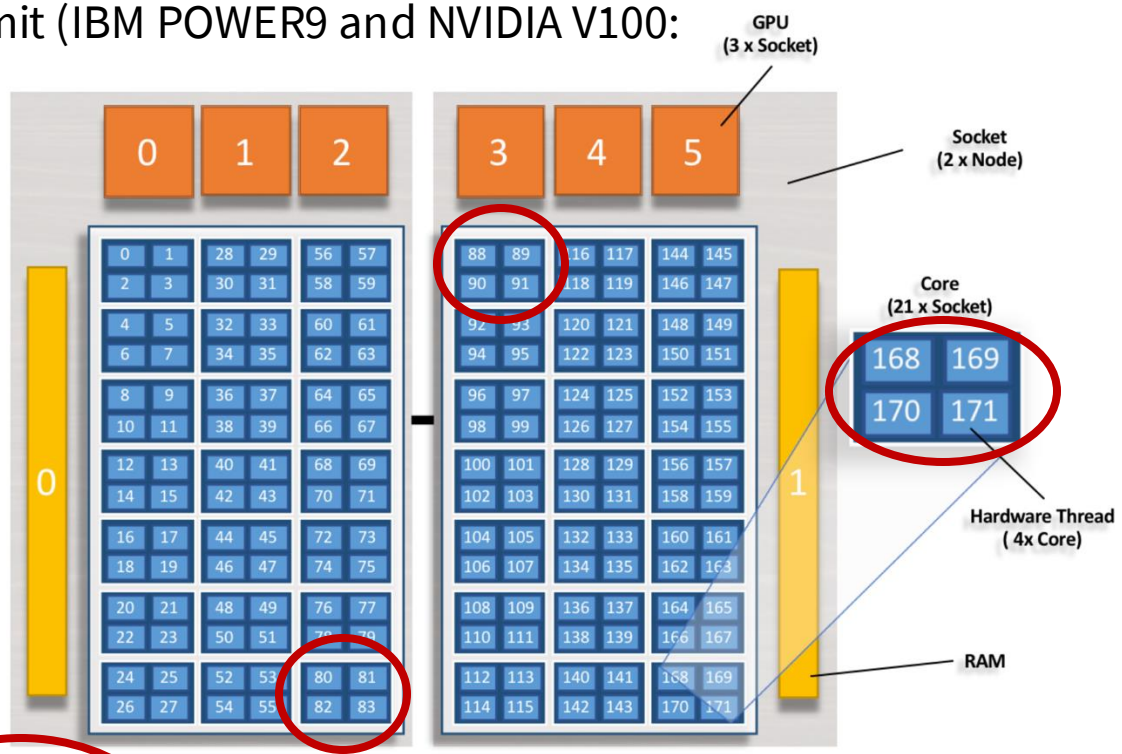
- **Sanity check** / curiosity / impatience (logging, essentially)
- **Check for misconfiguration**
- **Check for efficient utilization**
- Confirmation of expected hardware / operating system behavior
- **Identify cause of failure – deadlock, crash, stalls, etc.**
- Adaptation / computational steering / feedback & control
- ~~Identify system failures~~ (out of scope)
- Debugging/correctness/validation, essentially.

(Mis)Configuration – what could *possibly* go wrong?

- Process placement:
  - Logical/physical mappings, resources assigned/constrained to each process
  - Are there reserved core(s) for system? GPU mapping?
- Thread placement: Socket, NUMA domain, core, thread (HWT)
- Undersubscribing: Wasted hardware, energy, time (under-utilization)
  - Can provide better performance in some situations (memory-bound code)
- Oversubscribing: Increased contention with no realized benefit
- Imbalances
  - What is the communication frequency/volume between pairwise MPI ranks?
- Slurm/PBS/Alps/Torque/Flux are *complicated to use*
  - …especially when combined with MPI, OpenMP, GPUs, or other model settings

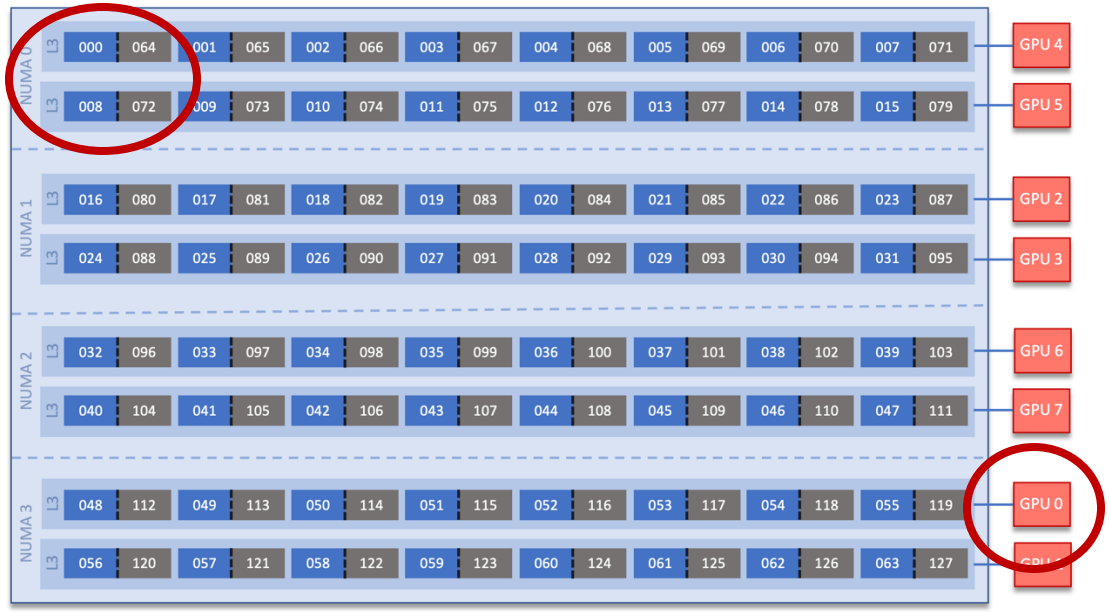
Although accurate, system documentation can be confusing

Summit (IBM POWER9 and NVIDIA V100):



- 1 node
- 2 sockets (grey)
- 42 physical cores* (dark blue)
- 168 hardware cores (light blue)
- 6 GPUs (orange)
- 2 Memory blocks (yellow)

Frontier (HPE Cray EX, AMD Epyc and MI250X):



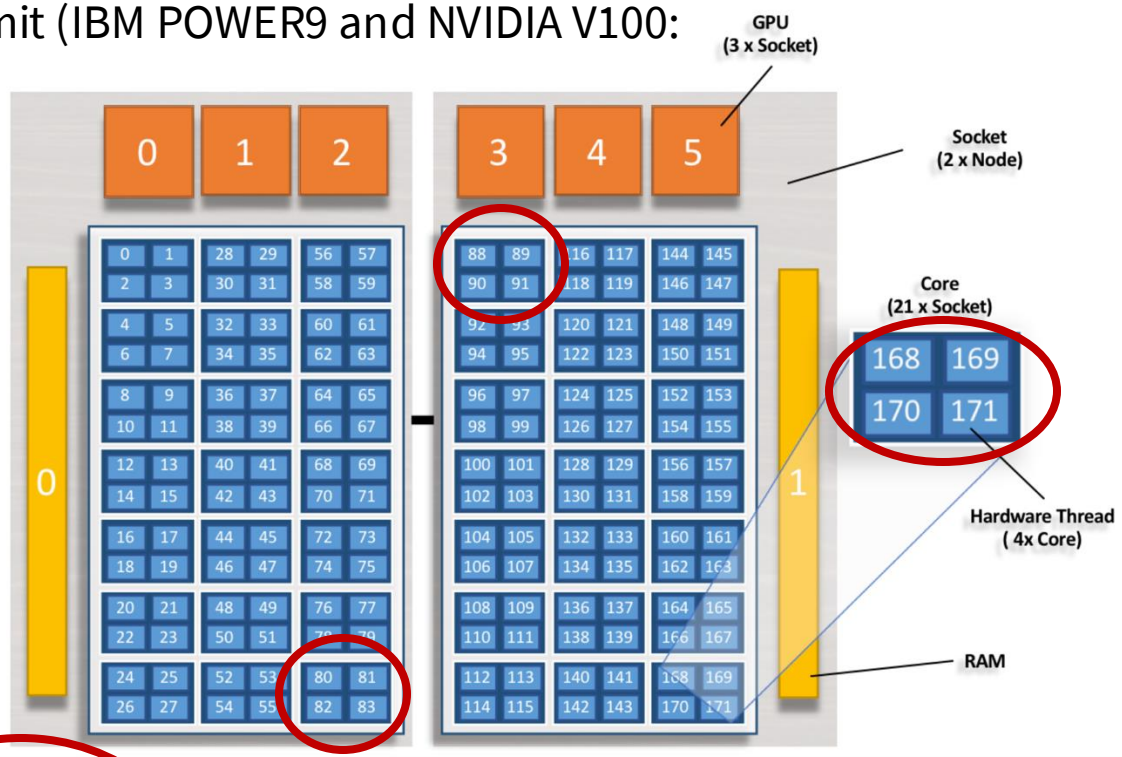
Core indexing, HWT indexing, GPU mapping not universal (or even intuitive)

***Core Isolation:** 1 core on each socket has been set aside for overhead and is not available for allocation through jsrun. The core has been omitted and is not shown in the above image.

Sources: https://docs.olcf.ornl.gov/systems/summit_user_guide.html , https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

Although accurate, system documentation can be confusing

Summit (IBM POWER9 and NVIDIA V100):



- 1 node
- 2 sockets (grey)
- 42 physical cores* (dark blue)
- 168 hardware cores (light blue)
- 6 GPUs (orange)
- 2 Memory blocks (yellow)

Frontier (HPE Cray EX, AMD Epyc and MI250X):

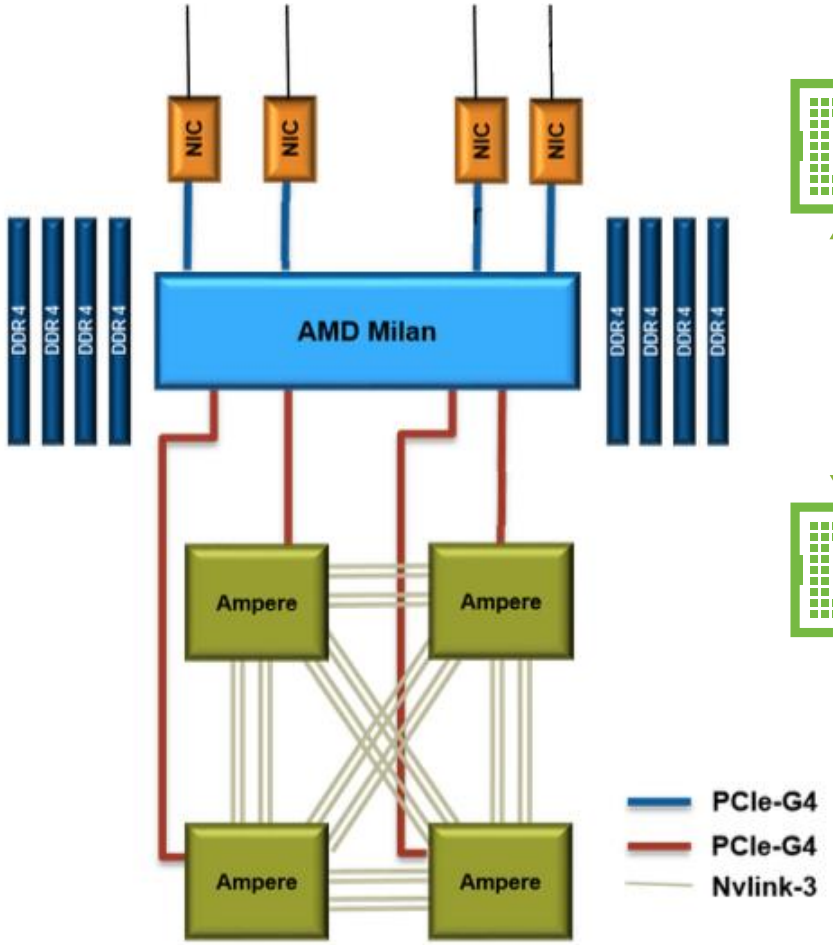
Modified default: system reserves 8 cores…but user controllable

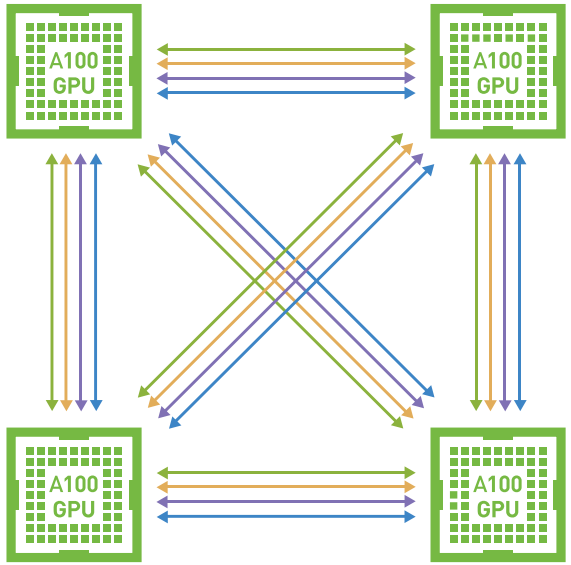
***Core Isolation:** 1 core on each socket has been set aside for overhead and is not available for allocation through jsrun. The core has been omitted and is not shown in the above image.

Sources: https://docs.olcf.ornl.gov/systems/summit_user_guide.html , https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

More systems – information missing

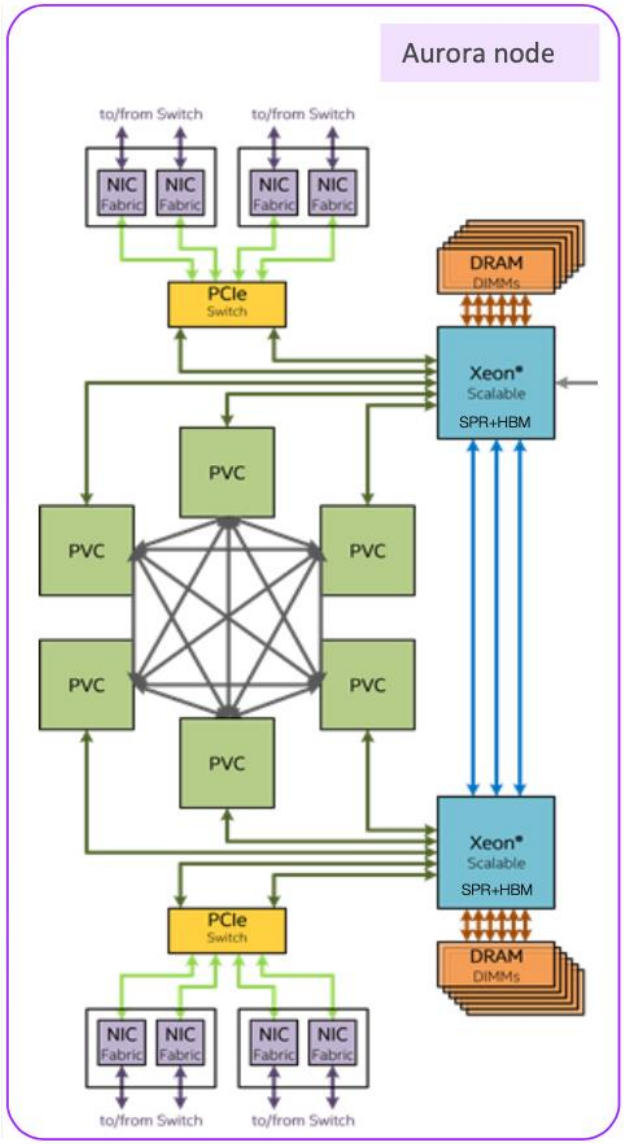
Perlmutter:





No physical layout information available in the documentation, no core or GPU indexing…

Aurora:



Sources: https://docs.nersc.gov/systems/perlmutter/architecture/
https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2022/11/ATPESC-2022-Track-1-Talk-9-Muralidharan-Aurora.pdf

ZeroSum: User Space Monitoring of Resource Utilization and Contention

- Inspired by the *hello_jsub* program from Tom Papatheodore: https://code.ornl.gov/t4p/Hello_jsrun
- Why "ZeroSum"? – An advantage for one side results in an equivalent loss for the other
  - the fixed number of resources available in an allocation – no elasticity
  - the need to periodically use *some* resources to monitor
- Available on GitHub: https://github.com/UO-OACISS/zerosum
- Monitors application threads (LWP), CPU hardware (HWT), Memory, and GPU hardware for all processes, all nodes in the allocation
- SC2023 HUST workshop paper: https://doi.org/10.1145/3624062.3624145

ZeroSum Functionality

- ✓ **Detect the initial/changing configuration of the application**
- ❑ Evaluate the configuration to automatically detect misconfigurations
- ✓ Provide runtime feedback to the user that the program is progressing
- ✓ **Provide a report of how effectively the hardware was utilized**
- ✓ **Provide a report of how much contention was identified in the execution**
- ❑ Provide a way to expose the observed data to other tools that can perform computational steering / runtime optimization / reconfiguration

✓ Implemented
❑ Future work

How to use ZeroSum

- Wrapper script(s) – `zerosum` and `zerosum-mpi`
  - Periodicity – default 1 second
  - Detailed/verbose output – true/false, default false
  - Heartbeat (memory consumption) – true/false, default false
  - Register signal handler – true/false, default false
  - Run in debugger – true/false, default false (also specify executable name)
  - Detect deadlocks – true/false, default false
  - Deadlock period – how many periods of “inactivity” is considered “deadlock”
  - HWT/Core for async thread – defaults to last core/HWT in affinity list for process
- Preloads the library, wraps `__libc_start_main` or creates global static constructor/destructor functions

What does it do? …Configuration Detection

- Query `/proc/[self|pid]/status` to get the allowable cores
- Query `/proc/meminfo` to get total memory available
- Query MPI rank, size, hostname (after `MPI_Initialized()` returns `true`)
- If available, use hwloc to query hardware topology (`lstopo`)
- Asynchronous background thread is started, it periodically queries:
  - `/proc/[self|pid]/status` to get process thread count, memory usage
  - `/proc/[self|pid]/task` directory to get all thread IDs
  - For each thread, query the affinity list for that thread (it may change!), utilization, state, core/HWT it's running on, context switches, other metrics
  - `/proc/stat` to query core utilization of all cores
- OMP-Tools (v5.0+) callback used to identify OpenMP threads at creation*
- NVML/ROCm-SMI/SYCL libraries used to query GPU(s)*
- Now also queries `/sys/cray/pm_counters/*` for power/energy

Utilization Report

- Rank 0 writes a summary report to the screen
- All ranks write a report to a log file – including full time series data as CSV
- All observed threads (LWP) are reported – user/system/idle, context switches, affinity list
- All assigned cores (HWT) are reported – user/system/idle
- All assigned GPUs stats are reported
- Extensive hwloc hierarchy written as JSON file
- Post-processing:
  - combine JSON data with CSV data for sunburst
  - Time-series charts of performance data

Example Output – miniQMC (OpenMP target offload) on Frontier (OLCF)

```
Duration of execution: 210.878 s

Process Summary:
MPI 000 - PID 51334 - Node frontier09085 - CPUs allowed: [1,2,3,4,5,6,7]

LWP (thread) Summary:
LWP 51334: Main,OpenMP - stime: 12.48, utime: 63.94, nv_ctx:   4, ctx: 365488, CPUs: [1]
LWP 51343:     ZeroSum - stime:  0.15, utime:  0.26, nv_ctx:   9, ctx:    679, CPUs: [7]
LWP 51374:       Other - stime:  0.00, utime:  0.00, nv_ctx:   0, ctx:      6, CPUs:
     [1-7,9-15,17-23,25-31,33-39,41-47,49-55,57-63,65-71,73-79,81-87,89-95,97-103,
      105-111,113-119,121-127]
LWP 51384:      OpenMP - stime: 12.60, utime: 64.00, nv_ctx:   3, ctx: 365742, CPUs: [3]
LWP 51385:      OpenMP - stime: 12.63, utime: 64.27, nv_ctx:   2, ctx: 352574, CPUs: [5]
LWP 51386:      OpenMP - stime: 12.74, utime: 63.76, nv_ctx: 473, ctx: 368585, CPUs: [7]
```

Process Summary

LWP (thread) Summary

Note: times are in jiffies

Example Output – miniQMC (OpenMP target offload) on Frontier (OLCF)

HWT (core/thread) Summary

```
Hardware Summary:
CPU 001 - idle:   22.70, system:   12.42, user:   64.52
CPU 002 - idle:   99.82, system:    0.00, user:    0.00
CPU 003 - idle:   23.08, system:   12.60, user:   63.97
CPU 004 - idle:   99.83, system:    0.00, user:    0.00
CPU 005 - idle:   22.79, system:   12.62, user:   64.23
CPU 006 - idle:   99.83, system:    0.00, user:    0.00
CPU 007 - idle:   22.94, system:   12.89, user:   63.81
```

Note: times are in jiffies

GPU Summary

```
GPU 0 - (metric: min  avg  max)
     Clock Frequency, GLX (MHz): 800.000000 1614.691943 1700.000000
     Clock Frequency, SOC (MHz): 1090.000000 1090.000000 1090.000000
     Device Busy %: 0.000000 14.616114 52.000000
     Energy Average (J): 0.000000 8.328571 10.000000
     GFX Activity: 0.000000 17223.704762 38443.000000
     GFX Activity %: 0.000000 13.706161 41.000000
     Memory Activity: 0.000000 623.623810 1536.000000
     Memory Busy %: 0.000000 0.355450 3.000000
     Memory Controller Activity: 0.000000 0.303318 2.000000
     Power Average (W): 90.000000 126.483412 138.000000
     Temperature (C): 35.000000 37.909953 39.000000
     UVD|VCN Activity: 0.000000 0.000000 0.000000
     Used GTT Bytes: 11624448.000000 11624448.000000 11624448.000000
     Used VRAM Bytes: 15044608.000000 4743346651.601895 4839596032.000000
     Used Visible VRAM Bytes: 15044608.000000 4743346884.549763 4839596032.000000
     Voltage (mV): 806.000000 891.848341 906.000000
```

Logs have full time series of all samples

Contention Detection Support

- Voluntary/non-voluntary context switches (analysis todo)
- Minor/major page faults, pages swapped (analysis todo)
- System time analysis (analysis todo)
- Comparing affinity lists – across threads *and* across processes (todo)
- Memory consumption (analysis todo)
- GPU memory consumption (analysis todo)
- Analysis: Reinforcement learning? Or just heuristics?
- However, **can detect deadlocks** – both **active** (i.e. spinning at MPI collective) and **passive** (i.e. waiting for mutex)

Evaluation / Example usage

MPI+OpenMP version of miniQMC on Frontier, 8 processes, 7 threads (64 cores, 1 thread per core, 8 cores reserved)

LWP	Type	stime	utime	nvctx	ctx	CPUs
18351	Main†	1.54	15.17	332905	1838	1
18356	*ZeroSum*	0.42	1.10	194	1007	1
18385	Other	0.00	0.00	0	41	1-127‡
18405	OpenMP	0.31	13.09	232689	5	1
18407	OpenMP	0.44	12.93	353365	11	1
18408	OpenMP	0.21	13.22	92528	3	1
18409	OpenMP	0.47	12.93	394014	10	1
18410	OpenMP	0.37	13.03	302371	7	1
18411	OpenMP	0.41	12.97	348829	10	1

Table 1: Frontier results, default configuration. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

```
export OMP_NUM_THREADS=7
srun -n8 zerosum-mpi miniqmc
```

LWP	Type	stime	utime	nvctx	ctx	CPUs
18552	Main†	3.13	88.40	5	704	1-7
18561	*ZeroSum*	0.79	2.64	2	2790	7
18588	Other	0.00	0.00	0	41	1-127‡
18589	OpenMP	1.10	90.00	9	716	1-7
18590	OpenMP	1.10	93.00	8	724	1-7
18591	OpenMP	1.07	90.52	9	692	1-7
18592	OpenMP	1.10	89.83	14	766	1-7
18593	OpenMP	1.10	90.48	7	728	1-7
18594	OpenMP	1.10	91.93	300	849	1-7

Table 2: Frontier results, configuration requesting 7 cores per process. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

```
export OMP_NUM_THREADS=7
srun -n8 -c7 zerosum-mpi miniqmc
```

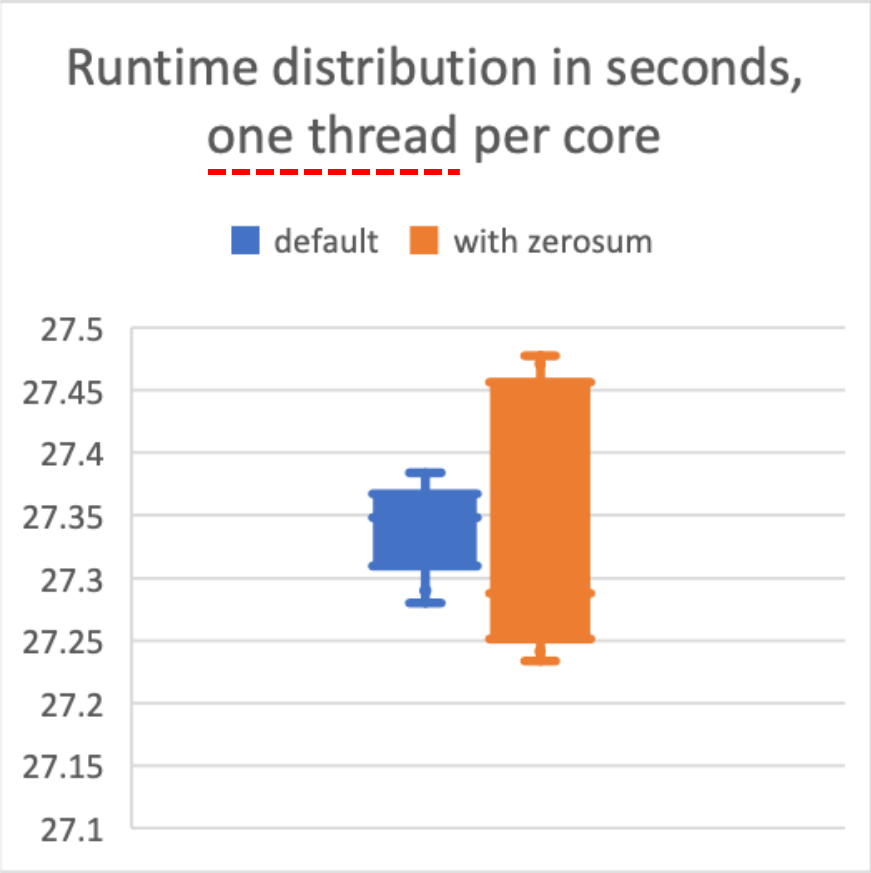
LWP	Type	stime	utime	nvctx	ctx	CPUs
18948	Main†	3.07	88.57	2	386	1
18954	*ZeroSum*	0.71	2.57	2	291	7
18981	Other	0.00	0.00	0	41	1-127‡
18992	OpenMP	1.18	96.36	0	422	2
18993	OpenMP	1.14	96.50	1	391	3
18994	OpenMP	1.18	96.46	0	381	4
18995	OpenMP	1.11	93.89	0	324	5
18996	OpenMP	1.14	93.29	0	370	6
18997	OpenMP	1.14	95.54	208	358	7

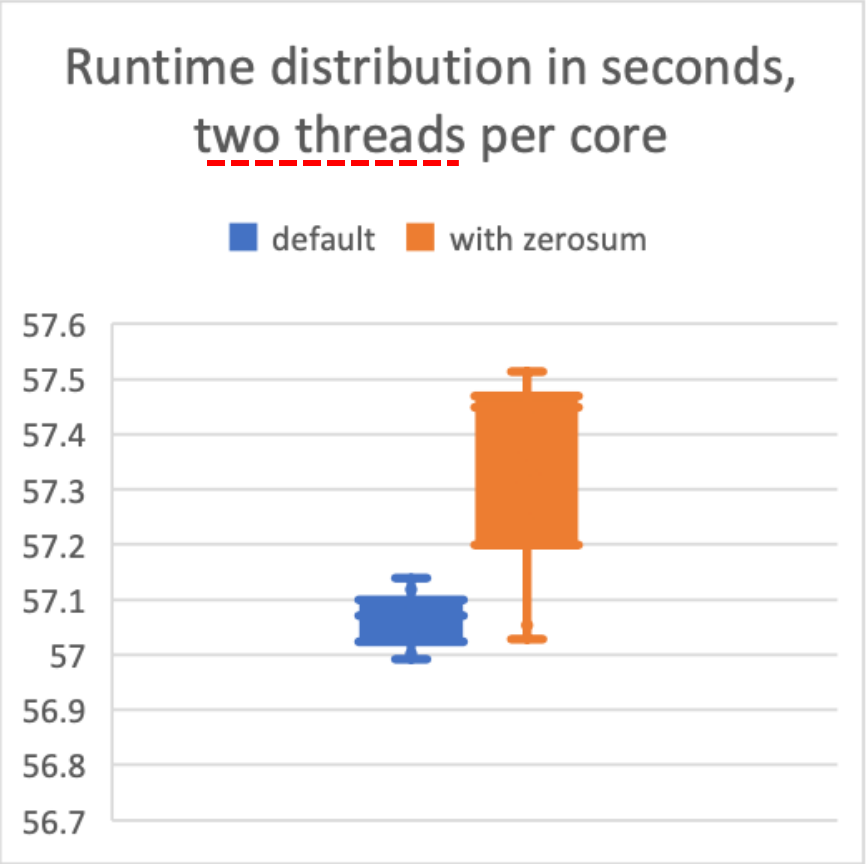
Table 3: Frontier results, configuration requesting 7 cores per process and binding OpenMP threads to cores. †indicates that the main thread is also an OpenMP thread. ‡indicates that the first core of each L3 region was set aside for system processes, not all threads in the sequence 1-127 are allowed but summarized for brevity in the table (see LWP 51274 in Listing 2).

```
export OMP_NUM_THREADS=7
export OMP_PROC_BIND=spread
export OMP_PLACES=cores
srun -n8 -c7 zerosum-mpi miniqmc
```

Note: stime/utime reported as % of total time

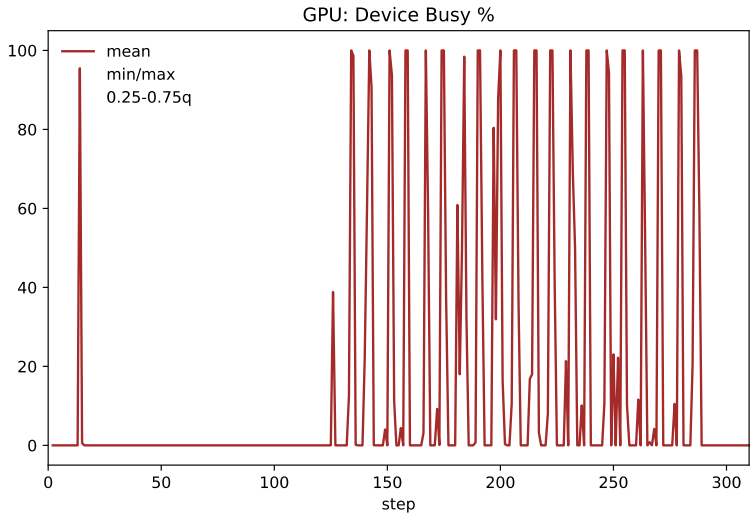
Overhead – less than 0.5% in resource constrained example

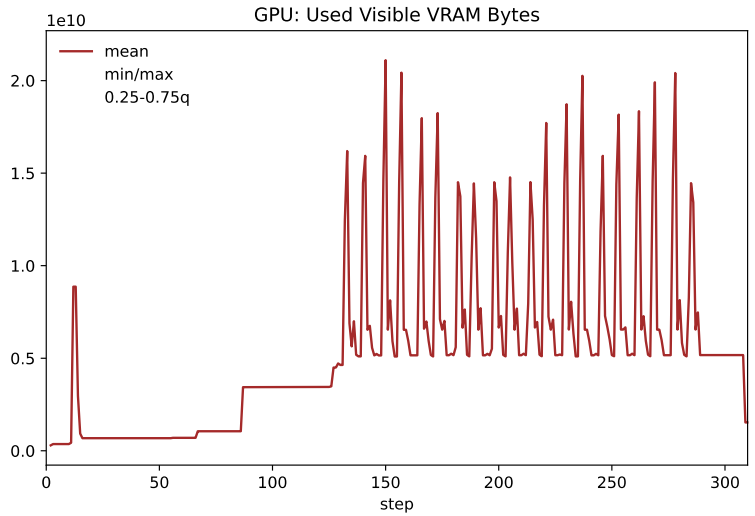


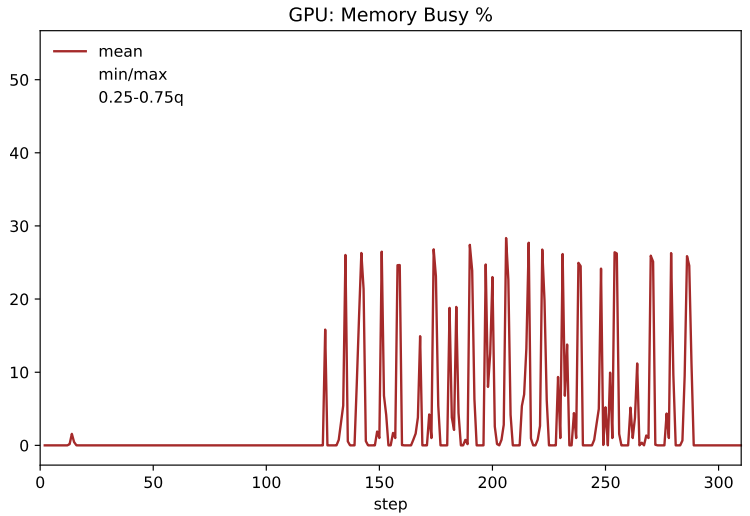


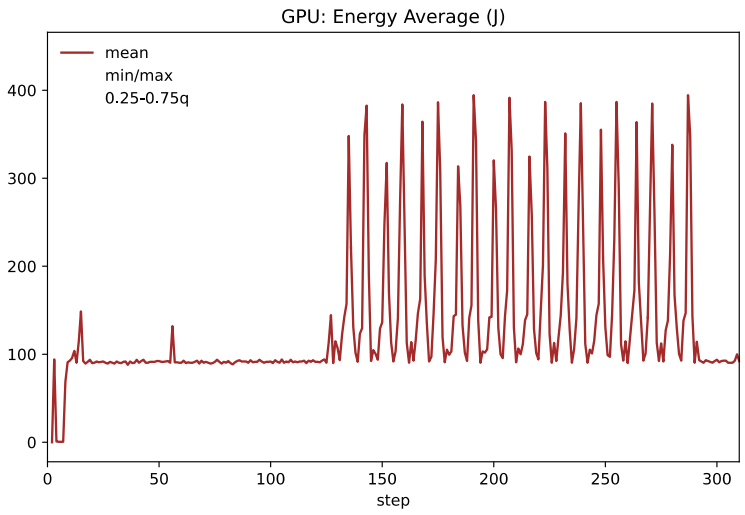
miniQMC time distributions executed 10 times using one OpenMP thread per core (left). In this comparison, the distribution of times with ZeroSum is noisier, but there is no significant observation of measurable overhead. The right figure shows the time distributions using two OpenMP threads per core. In this comparison, the distribution of times with ZeroSum is both noisier and longer tailed, and does show an observation of overhead, averaging about 0.2752 seconds, or 0.5%.

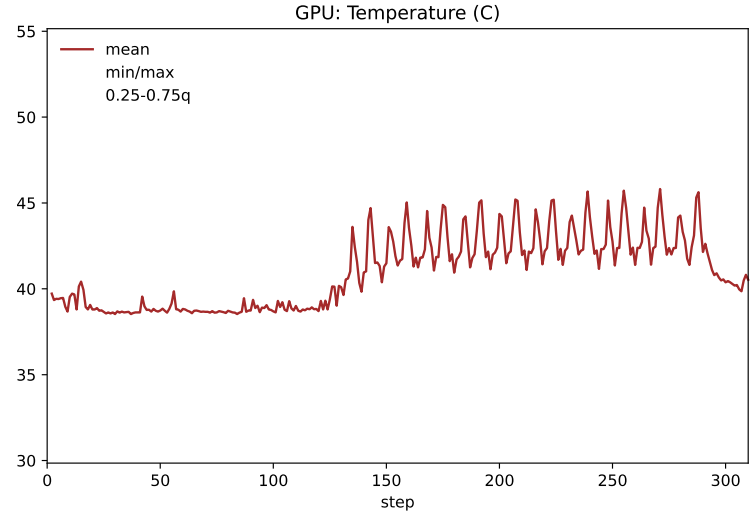
Summary views of collected data – 64 GCDs, XGC running on Frontier

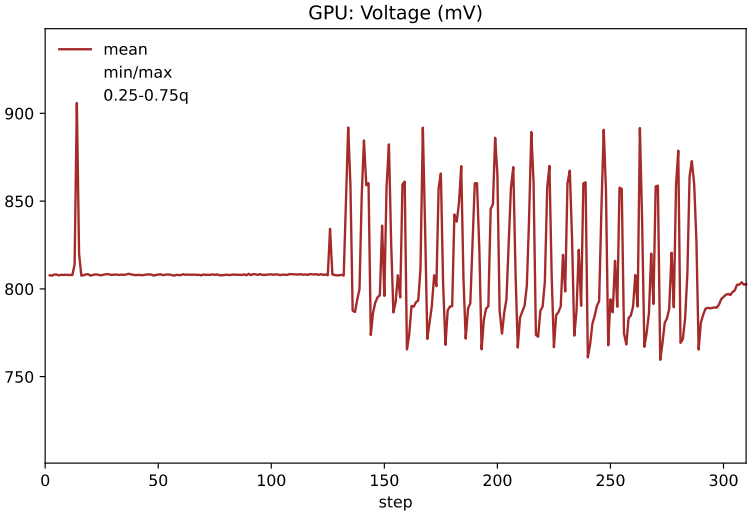






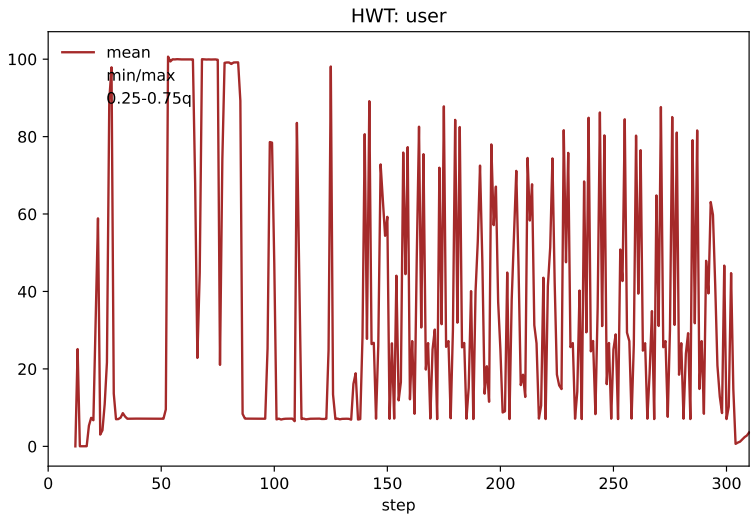


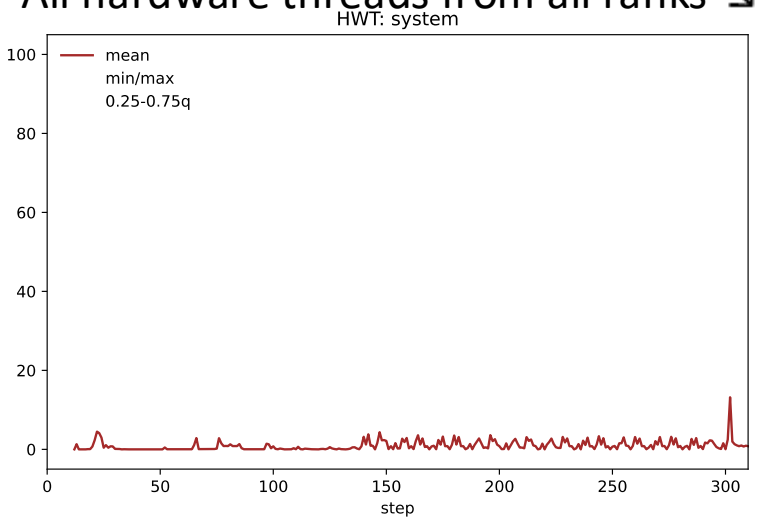


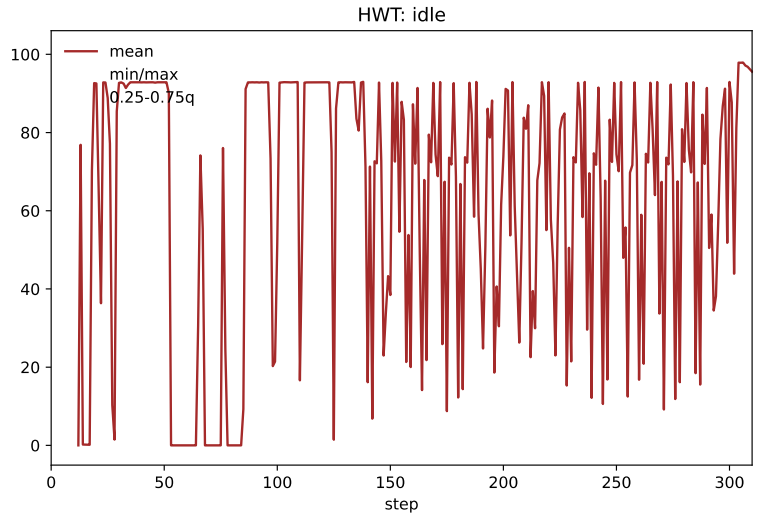


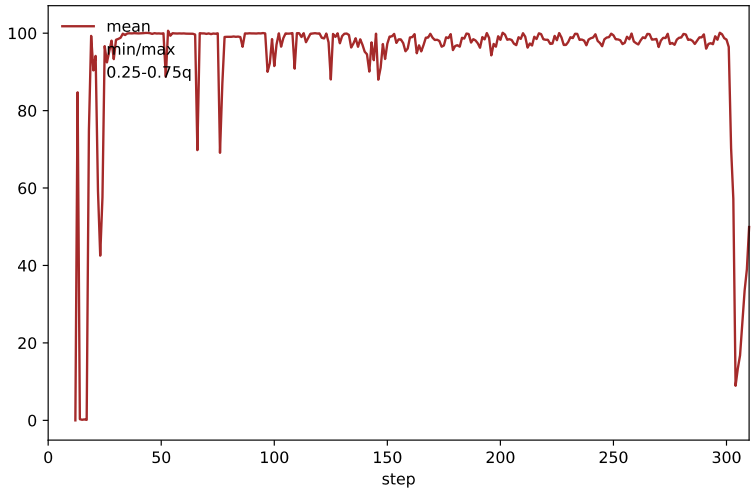
Summary views of collected data – 896 HWTs, XGC running on Frontier

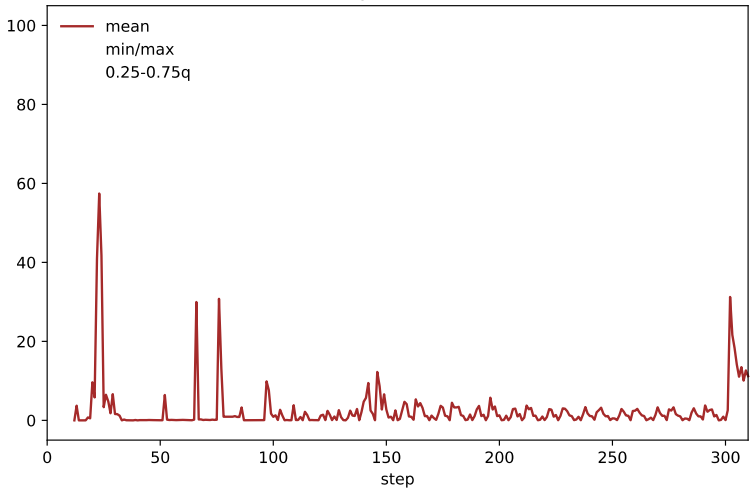
↙ All hardware threads from all ranks ↘

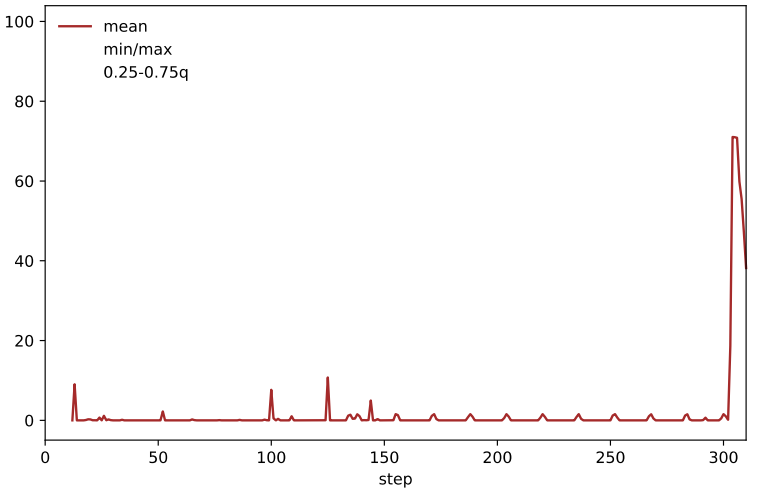








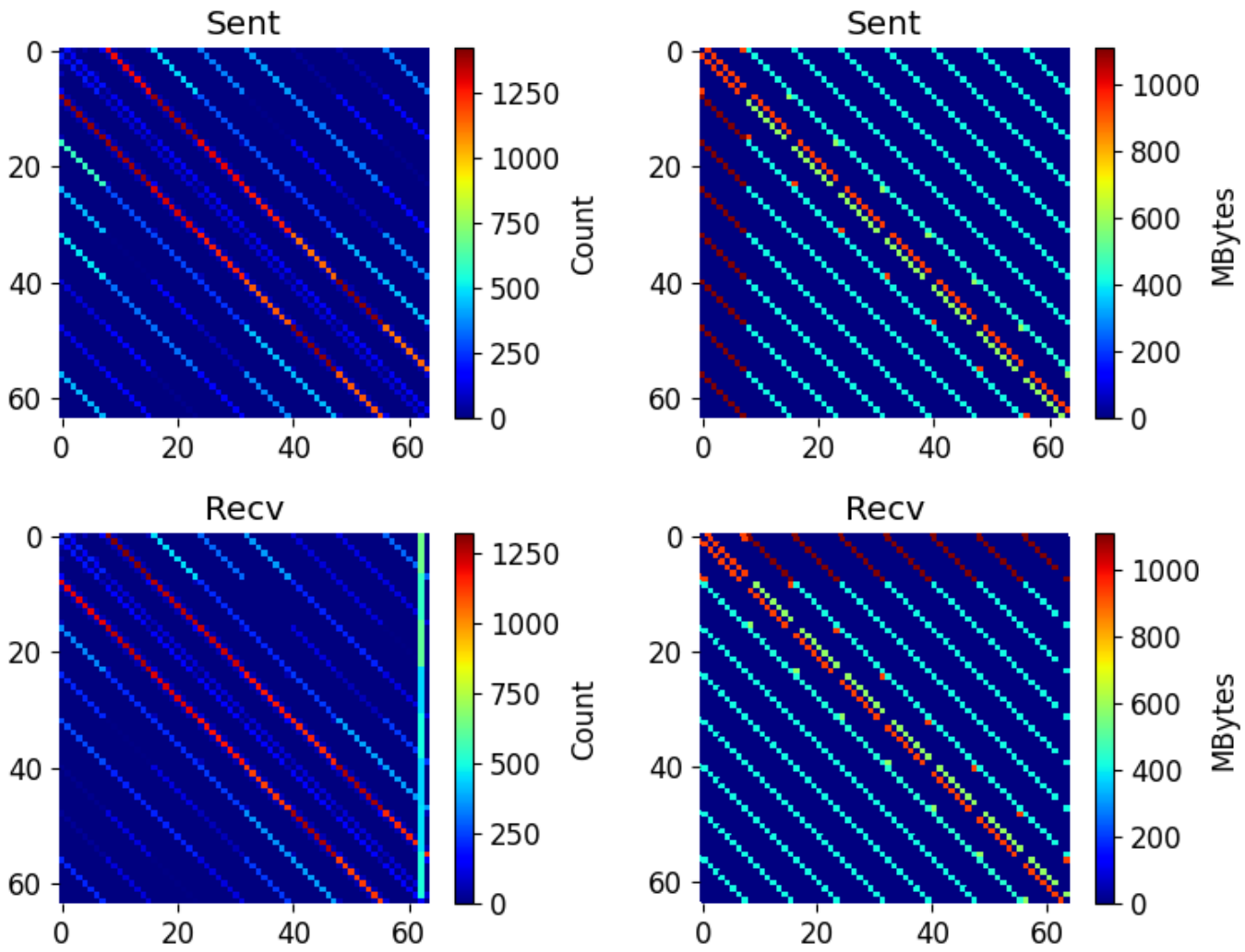




↖ Only main hardware thread from all ranks ↗

MPI P2P data

- MPI implementation includes wrappers for `MPI_Recv/Irecv`, `MPI_Sendrecv`, `MPI_Send/Bsend/Isend/Rsend/Ssend` to capture P2P frequency, volume
- Obviously can be done by other/better performance measurement tools, but for a quick view, can be useful
- What about a P2P graph? Could be interesting…



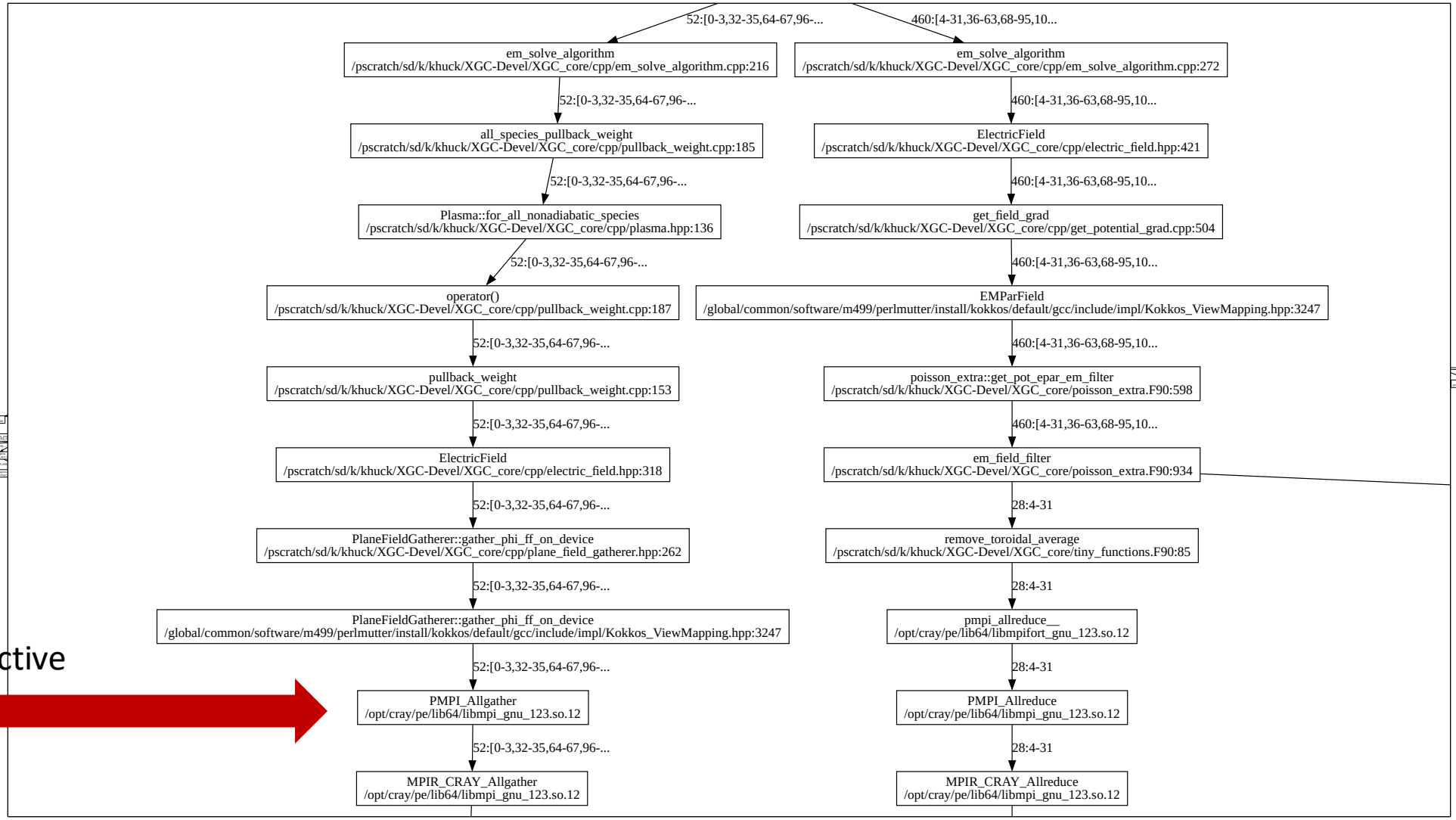
Successful Use Cases

- Octo-Tiger (HPX) thread placement on Fugaku (Fujitsu A64FX CPUs)
  - Detected unusual core indexing, needs to be mapped using hwloc information
- Argobots example
  - Detected that the user was running client & server on same node with overlapping affinity lists
- Grid (https://github.com/paboyle/Grid) on Frontier (lattice QCD library)
  - Helped find/understand ideal resource mapping with custom bind arguments
- XGC on Frontier
  - Empirically demonstrated that 2 threads per core provided better performance than 1 thread per core
  - Helped detect deadlock in GPU-aware libfabric/MPI implementation
  - Helped detect algorithmic deadlock in MPI (application bug)

Deadlock detection – at scale

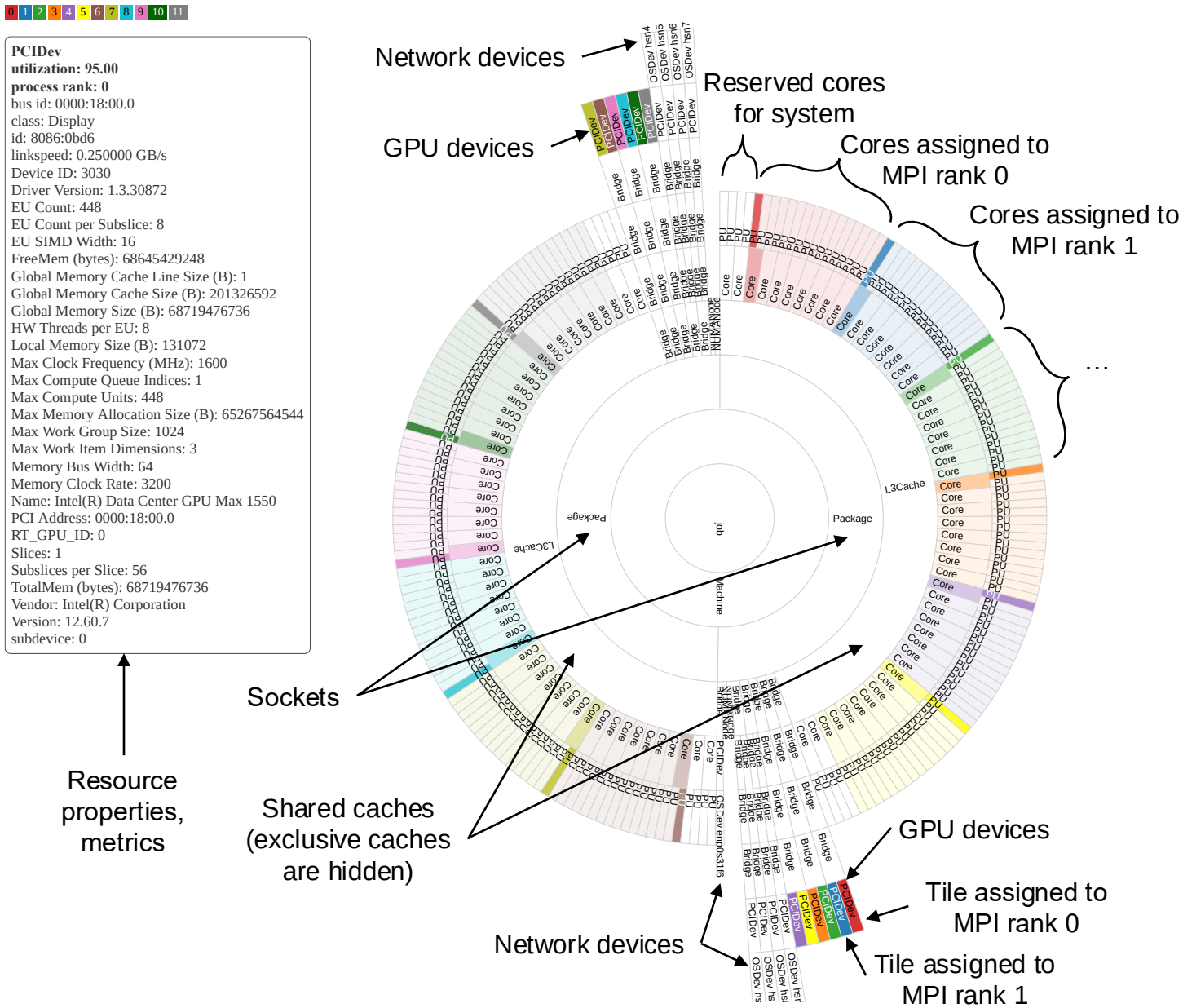
- Runs application in gdb/cuda-gdb/rocgdb/gdb-oneapi in “batch” mode – sequence of commands are scripted
- ZeroSum monitors all threads during execution:
  - If all threads sleeping for N seconds (N is user-configurable):
    - pthread_kill(SIGQUIT) the main thread
  - If all but 1 sleeping for N seconds, *and* that thread doesn’t have at least one minor page fault during that time, it’s *probably* deadlocked at an MPI collective:
    - pthread_kill(SIGQUIT) the main thread
- Gdb will intercept the signal and is scripted to get a backtrace from all threads
- Post-process the backtraces with `zs-stacks.py` and make a tree
  - just like STAT/Cray-STAT
  - *However,* ZeroSum is automated - queue times on Perlmutter were around 8 hours, user didn’t want to babysit the slurm request

Example: XGC deadlocked on Perlmutter with 512 ranks



Only 52 of 64 in communicator participated in collective

HTML/SVG/JS interface: Aurora example



Sunburst HTML template with support from https://d3js.org populated by Python script with JSON and CSV data

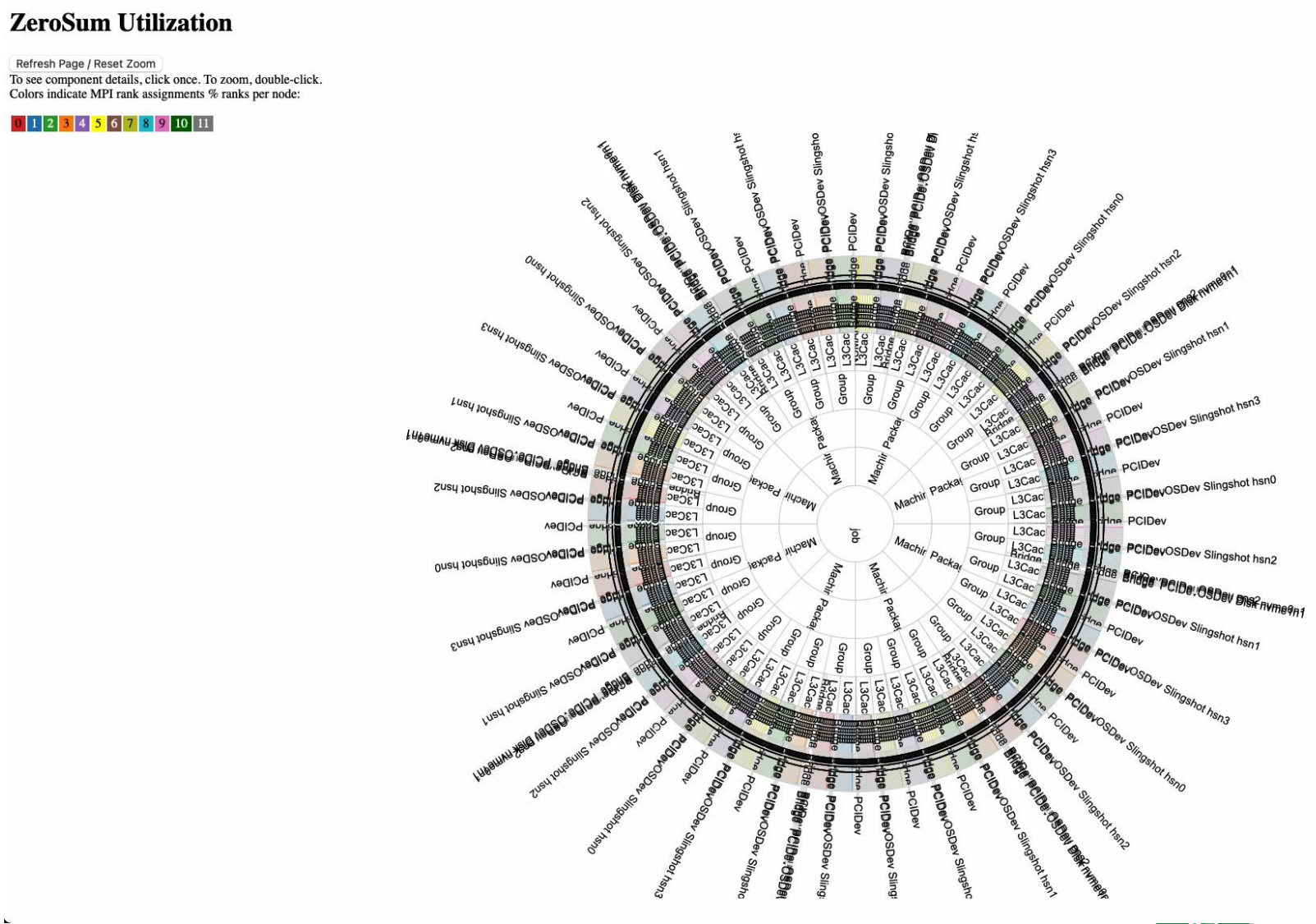
No GUI to install!

Just run: zs-hwloc-sunburst.py

Inspired by MemAXES: https://github.com/LLNL/MemAxes

HWLOC integration with utilization data

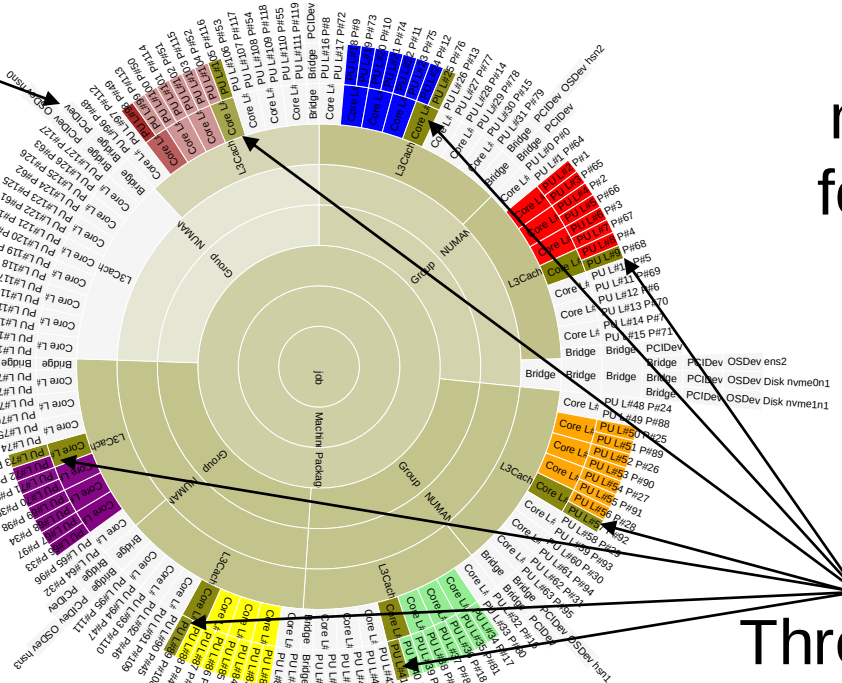
- 64 rank, 8 node job on Frontier
- 14 threads per process, color coded by shmrank (local rank per node)
- Segment color intensity relative to utilization
- Validates that process threads are pinned where requested



Slurm bug(?)

- Comparing --threads-per-core=2 (left) and --threads-per-core=1 (right)
- Threads for rank 7 “spread” throughout the node, all L3 regions *except* where the rank’s GDC is… <u>even with --gpu-bind=closest</u>

GCD assigned to rank 7

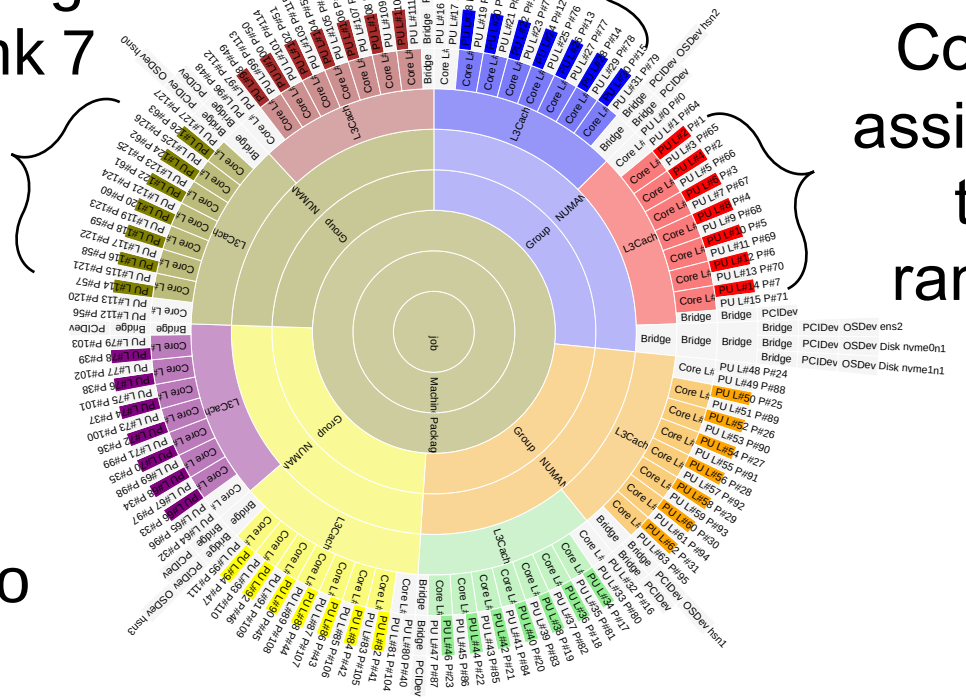


Correct mapping for rank 7

Threads assigned to rank 7

Cores assigned to rank 1

Cores assigned to rank 0



Conclusions, Future Work

- ZeroSum addresses the *configuration optimization* problem
  - https://github.com/UO-OACISS/zerosum
- Still need *automated* misconfiguration/contention analysis
- CSV output data could be better? – use ADIOS2 BP5? HDF5? SQLite?
  - Depends on analysis needs/wants
- Streaming data to Mochi (SOMA) or other service (LDMS)?
  - Enables robust monitoring approach – but likely redundant in many cases
  - Worked with students to implement SIMON (Simple Monitor)
- Integration with performance tools (TAU, APEX, etc)
  - Analysis of application performance data in context of system monitoring data
- Input for automated feedback/control (APEX, Argobots, SOMA, application)
- Finish integration of high-level HW counter data from LIKWID

Acknowledgements

This work was supported by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR) under contract DE-SC0021299.

Parts of this research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.