Energy efficiency optimization using dynamic tuning of Grace Hopper frequencies

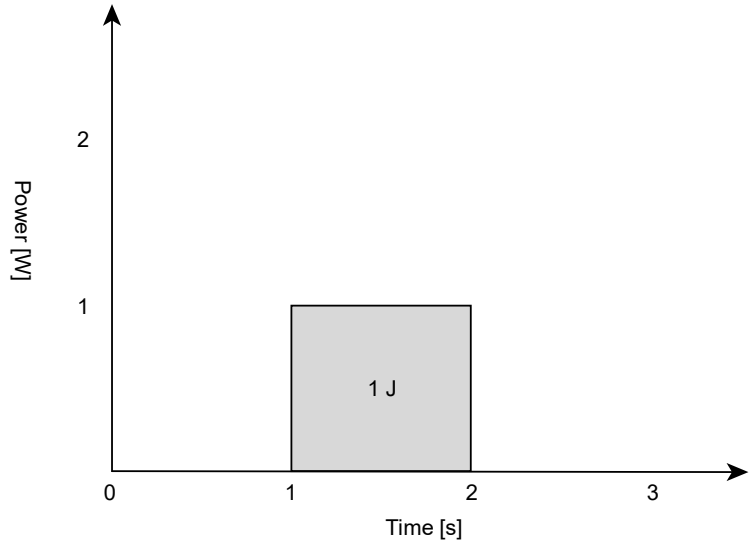
O. Vysocky, O. Meca
HiPEAC'26

EUPEX

Energy

- Power [W]
- 1 W * 1 s = 1 J
- 1 W * 1 h = 1 Wh = 3 600 J

$$Energy = Power \times Time$$



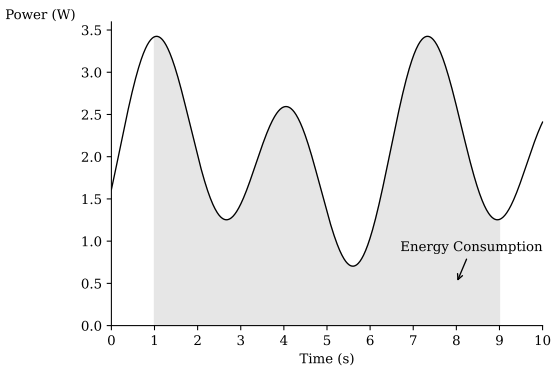
EUPEX

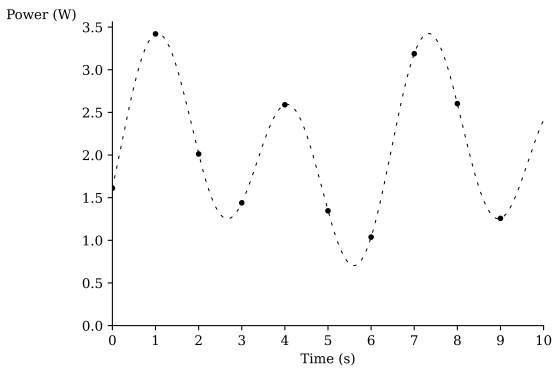
Energy

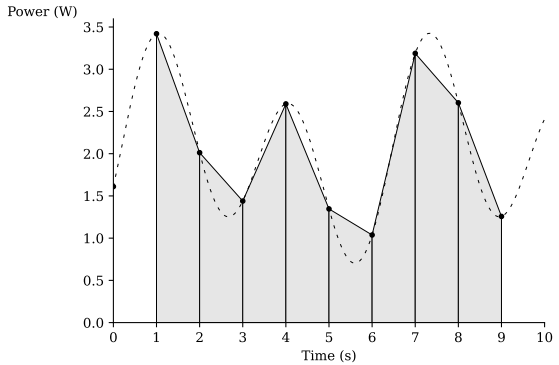
- Power [W]
- 1 W * 1 s = 1 J
- 1 W * 1 h = 1 Wh = 3 600 J

$$Energy = Power \times Time$$

$$Energy(t) = \int_0^t Power(x)\,\mathrm{d}x \approx \frac{\sum_{i=0}^{n} PowerSample_i}{SamplingFrequency}$$



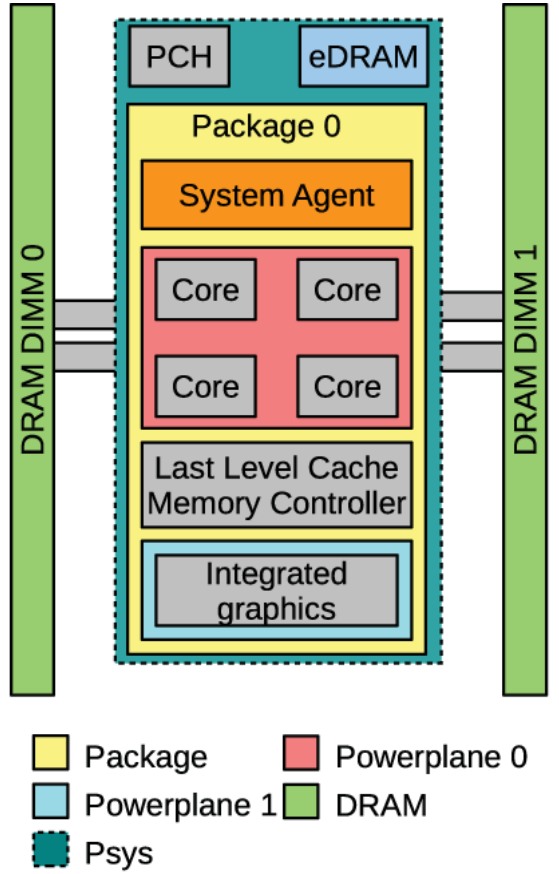




Img source, Luís Cruz (TU Delft)

Intel Running Average Power Limit (RAPL)

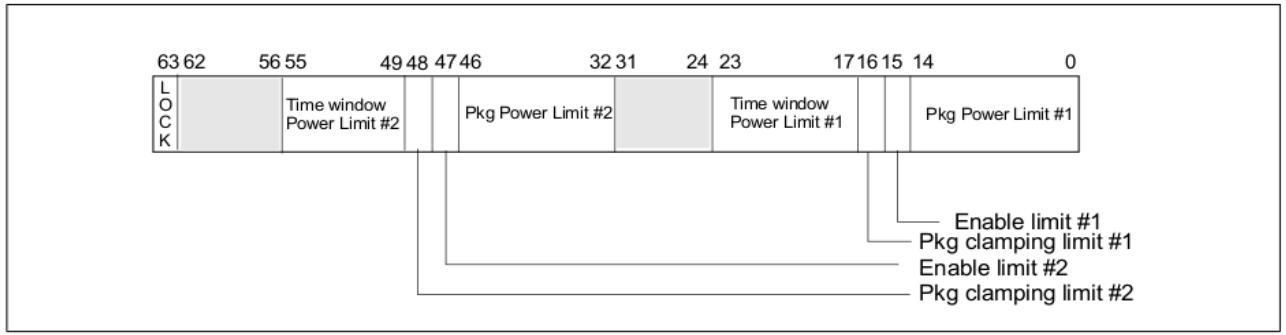
- Sysfs: /sys/devices/virtual/powercap/intel-rapl/intel-rapl:X/intel-rapl:0:Y
- Power domains:
  - **Package**: limits the power consumption for the entire package of the CPU, this includes cores and uncore components
    - short (1.2 * TDP, ~ milliseconds) and long window (TDP, ~ second)
  - **DRAM**: is used to power cap the DRAM memory = memory monitoring, P-State scaling
    - only for server architectures, no client
    - single time window
    - in default is turned off
  - **PP0/Core**: is used to restrict the power limit only to the cores of the CPU
    - no new server
    - single time window
  - **PP1/Graphic**: is used to power limit only the graphic component of the CPU
    - no server
    - single time window
  - **PSys/Platform**: controls entire System on Chip
    - short and long window
    - available from Skylake architecture
    - requires support from vendor

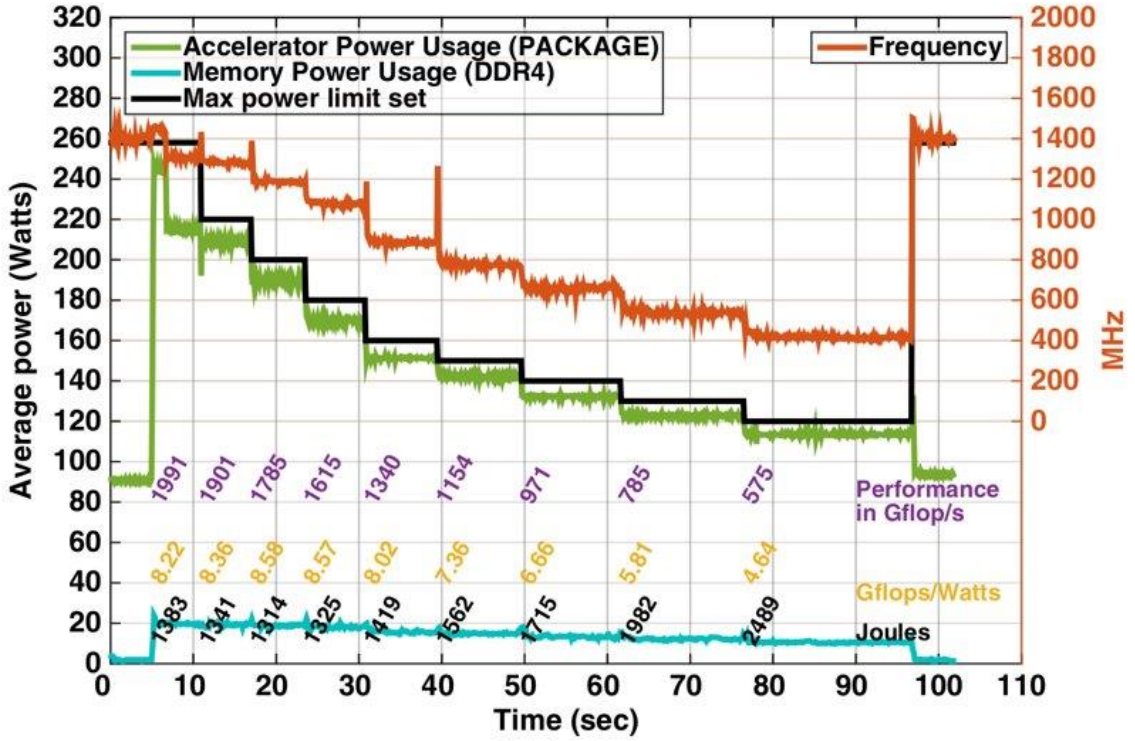


Img src Khan, K. et al. "RAPL in Action."

Intel Running Average Power Limit (RAPL)

- MSR MSR_PKG_POWER_LIMIT (0x610)





Haidar et al: Investigating power capping toward energy-efficient scientific applications

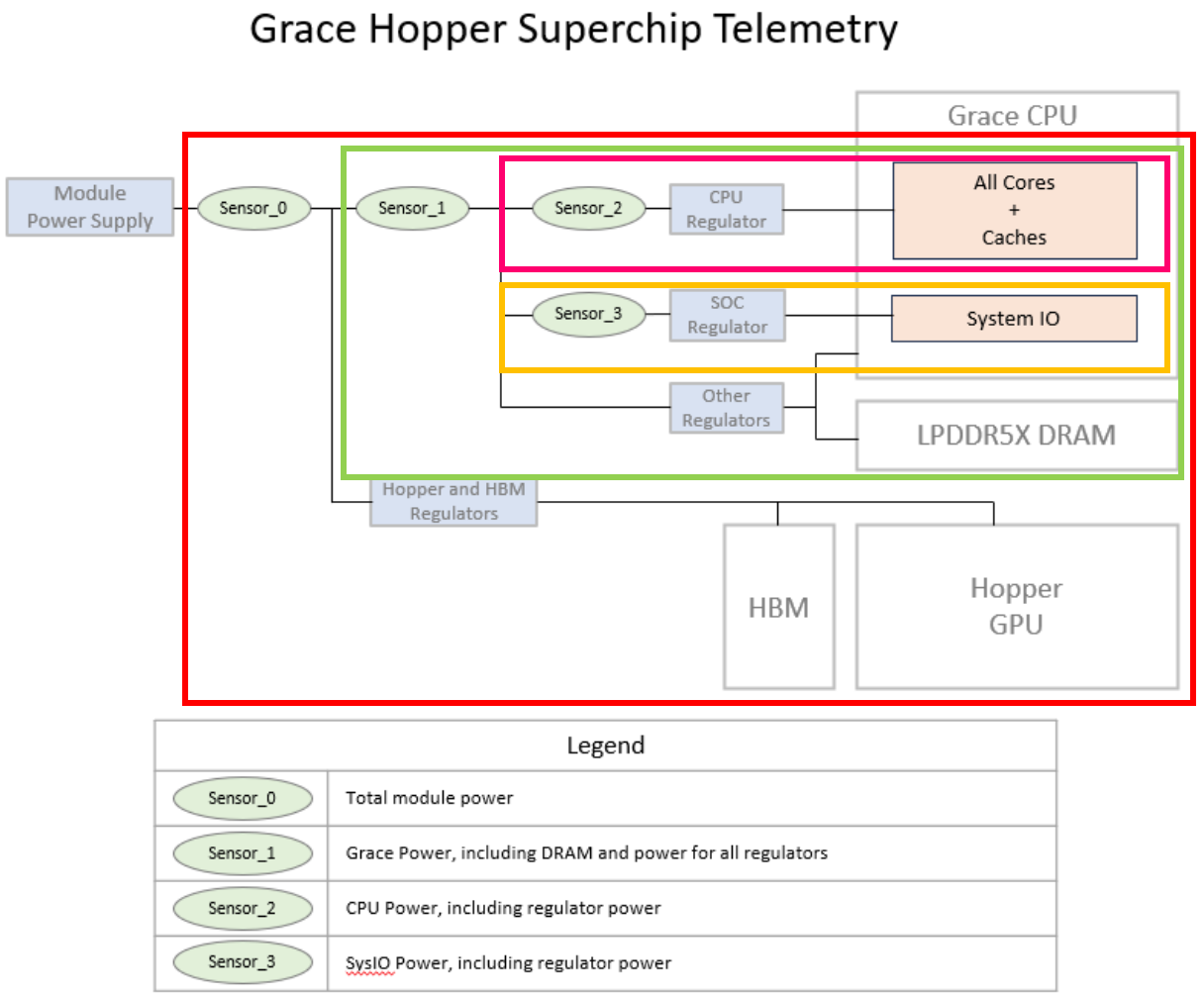
GraceHopper power monitoring

- Standard HWMON interface
- `/sys/class/hwmon/hwmon*/device/power1_average`
- `/sys/class/hwmon/hwmon*/device/power1_average_interval`
- Over 50-1000ms interval at the set of sensor points
- Average power in the window, no energy accumulation
- The same is valid for GraceBlackwell as well

GraceHopper power monitoring

HWMON

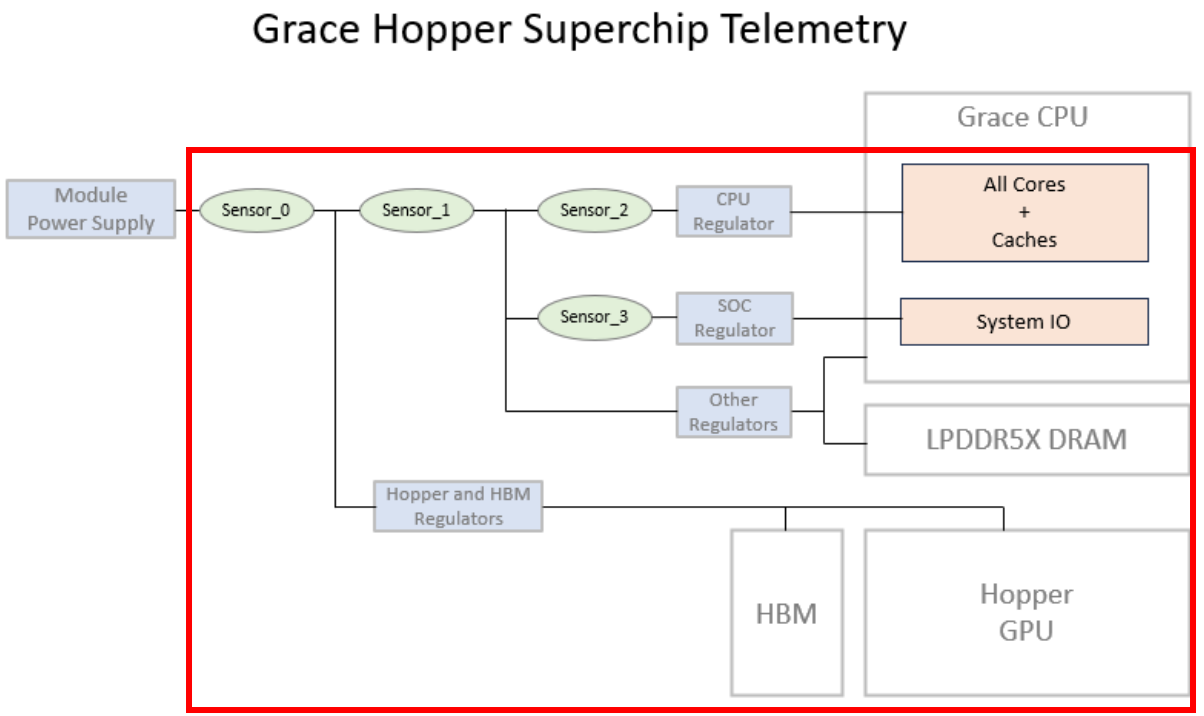
- **Module**
- **Grace = CPU+SysIO+DRAM**
- **CPU**
- **SysIO**
- **GPU = Module - Grace**



GraceHopper power monitoring

NVML

- nvmlDeviceGetTotalEnergyConsumption

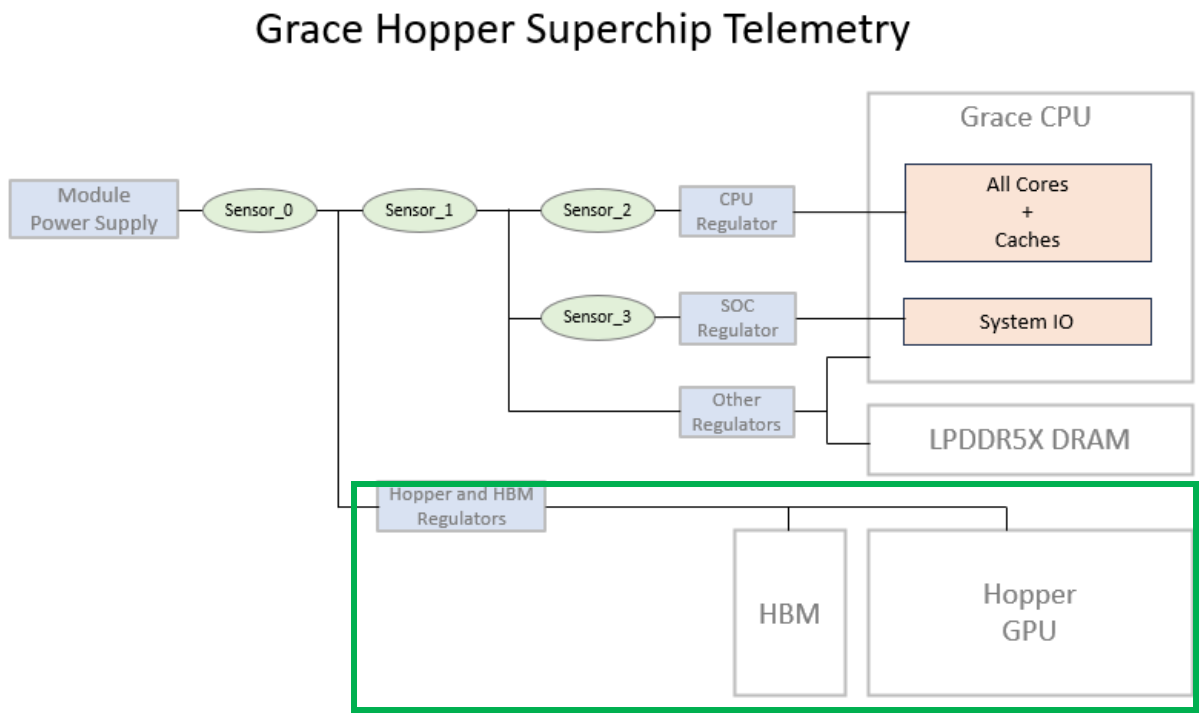


Legend	
Sensor_0	Total module power
Sensor_1	Grace Power, including DRAM and power for all regulators
Sensor_2	CPU Power, including regulator power
Sensor_3	SysIO Power, including regulator power

GraceHopper power monitoring

NVML

- nvmlDeviceGetPowerUsage

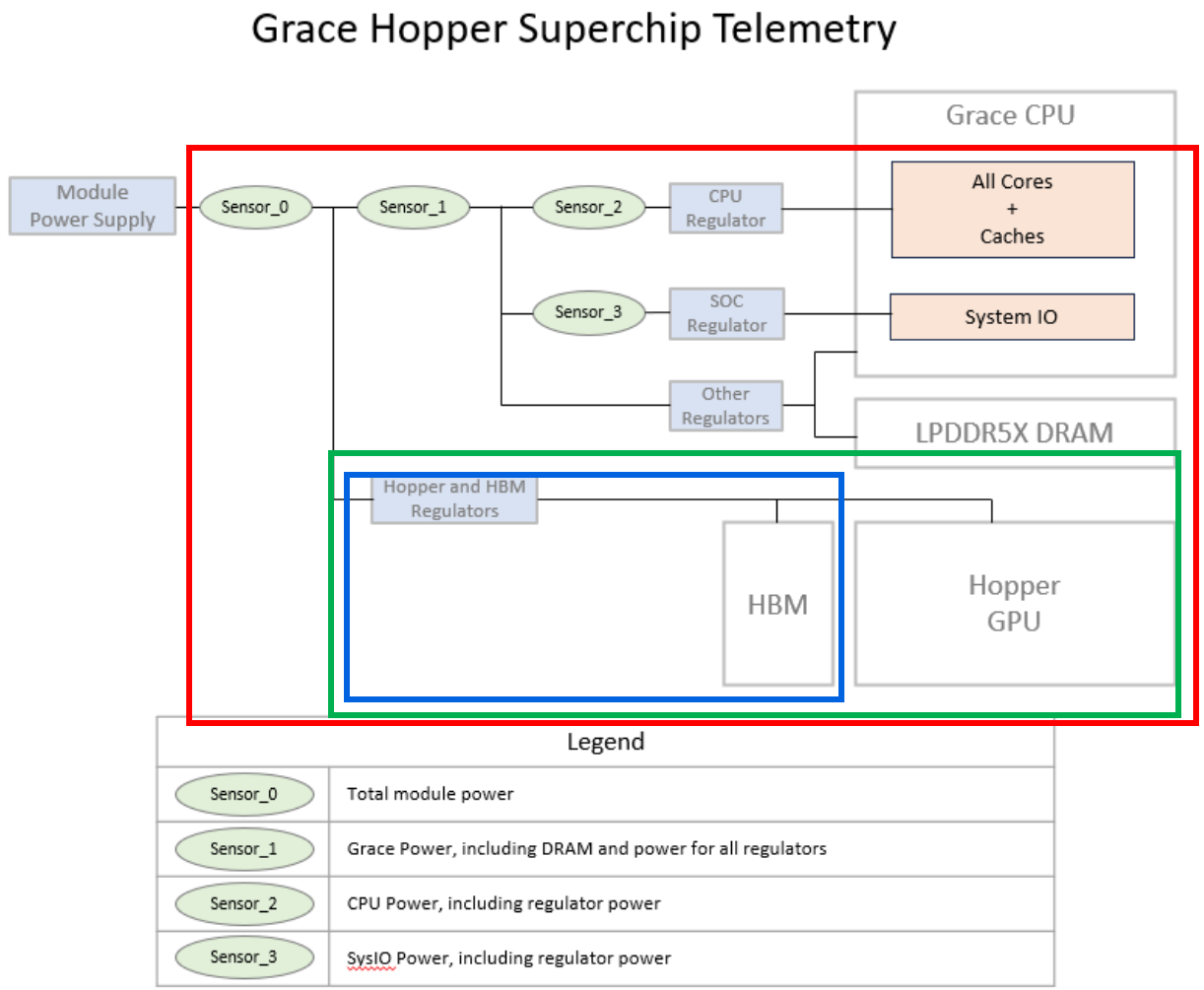


Legend	
Sensor_0	Total module power
Sensor_1	Grace Power, including DRAM and power for all regulators
Sensor_2	CPU Power, including regulator power
Sensor_3	SysIO Power, including regulator power

GraceHopper power monitoring

NVML

- nvmlDeviceGetFieldValues
  - **GPU_POWER**
  - **MODULE_POWER**
  - **MEMORY_POWER**

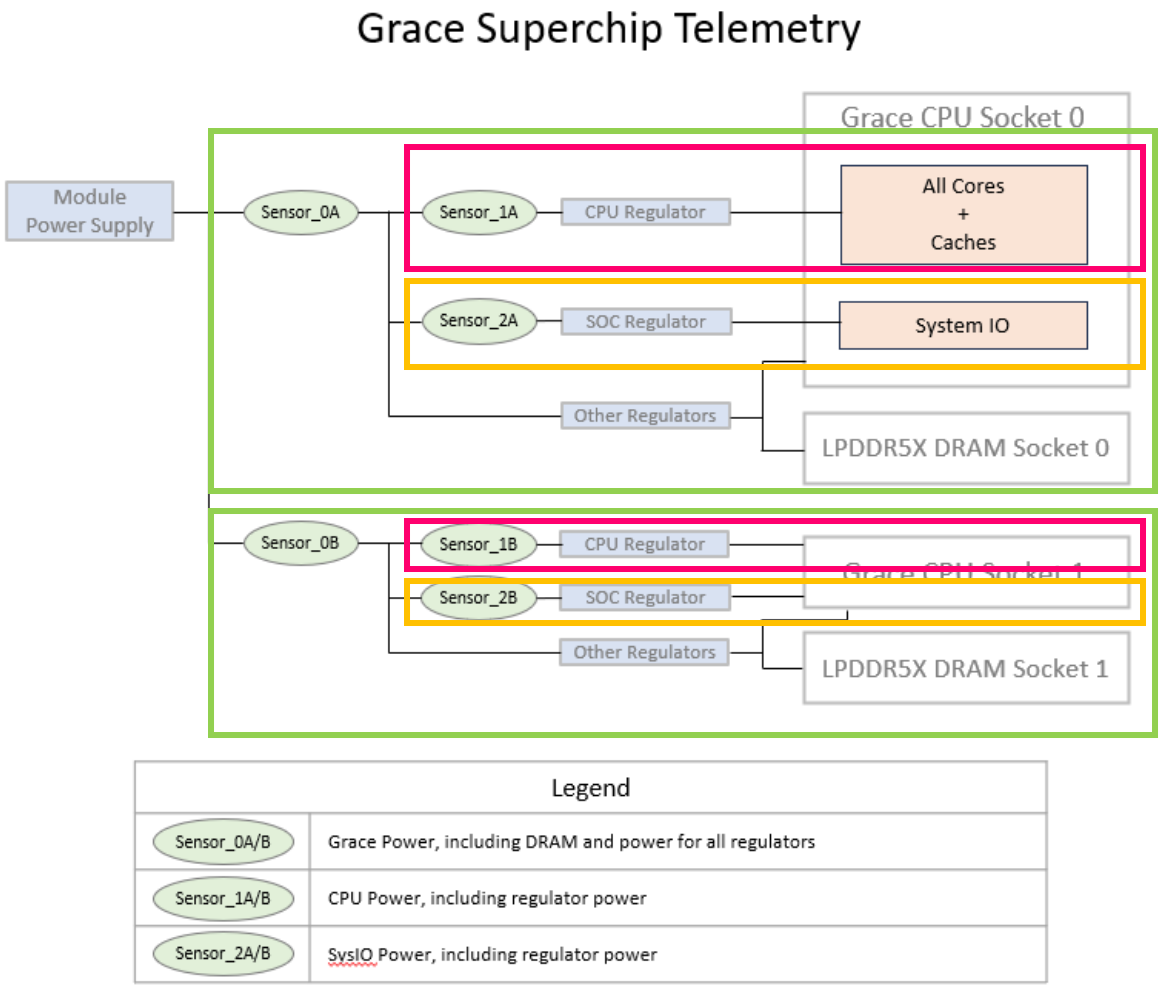


Grace superchip power monitoring

HWMON

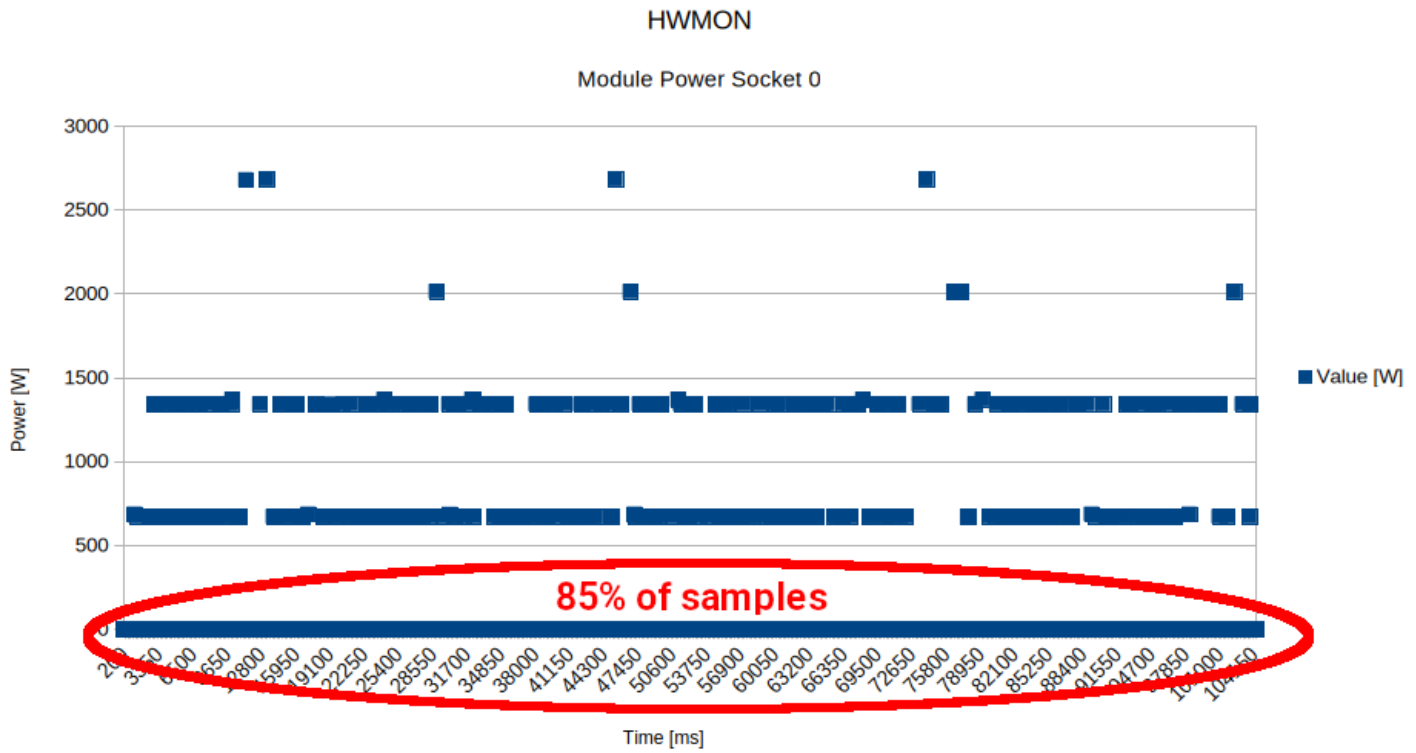
- Grace
- CPU
- SysIO
- DRAM = Grace - CPU - SysIO

No GPU, no NVML



Grace/GraceHopper power monitoring

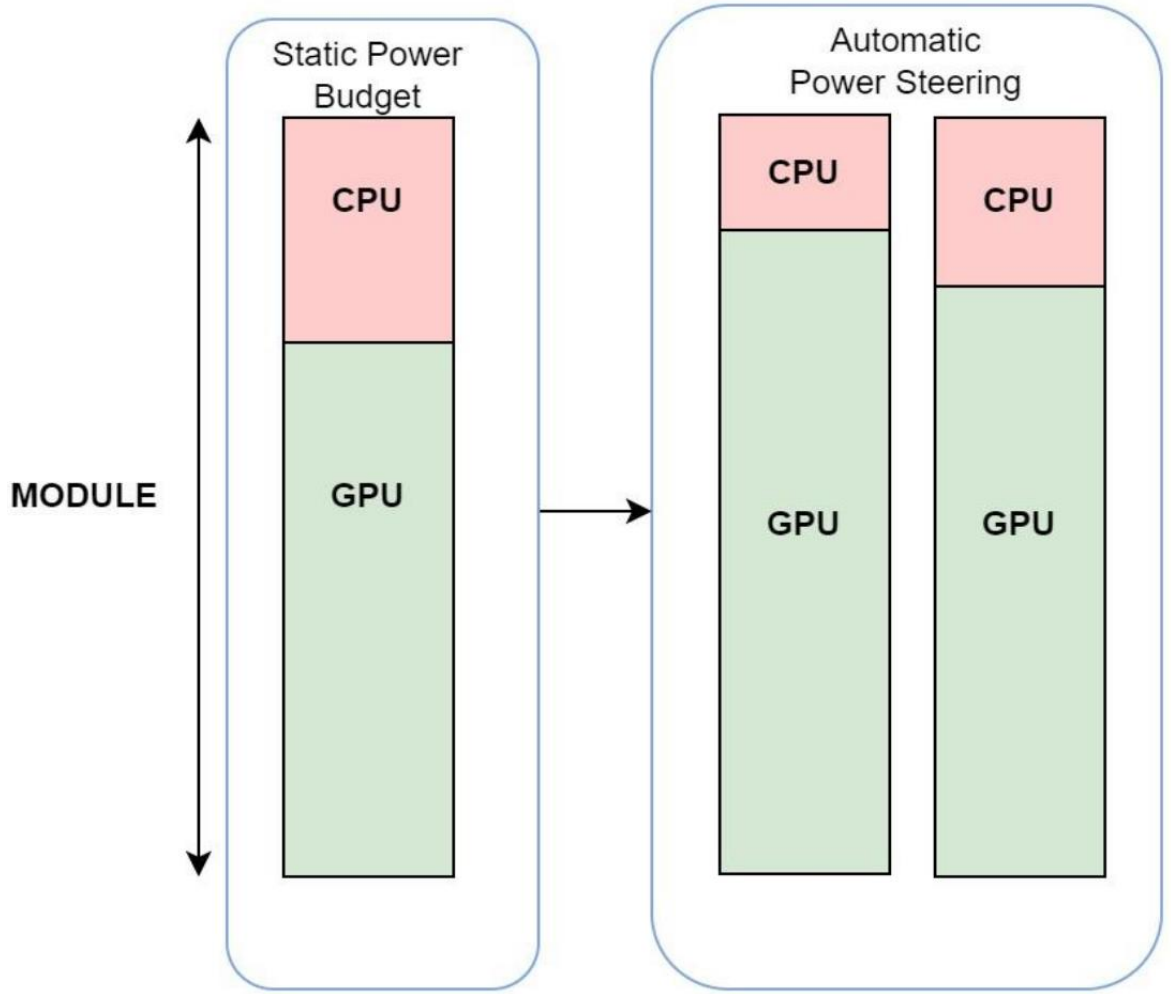
- Average power in the window, no energy accumulation
- Upgrade your BIOS
- Fixed version released in 2024H2



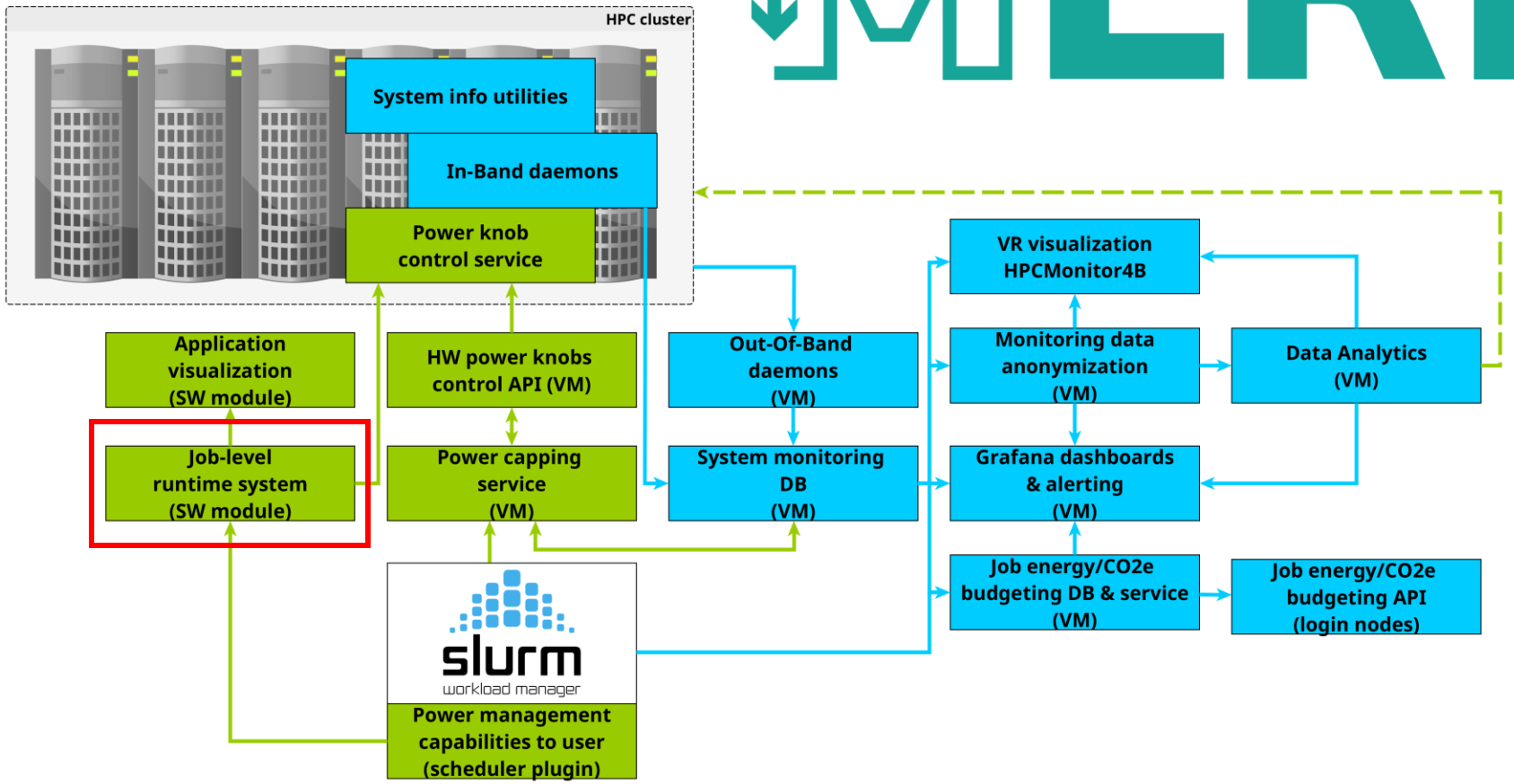
GH200 power management

- CPU frequency - `/sys/devices/system/cpu/cpu*/cpufreq/scaling_setspeed`
- GPU SM frequency - `nvmlDeviceSetApplicationsClocks`
- GPU HBM frequency - `nvmlDeviceSetApplicationsClocks`
- CPU power limit - `/sys/class/hwmon/hwmonX/device/power1_cap`
- GPU power limit - `nvmlDeviceSetPowerManagementLimit_v2`
- Module power limit - `nvmlDeviceSetPowerManagementLimit_v2`

Module power capping

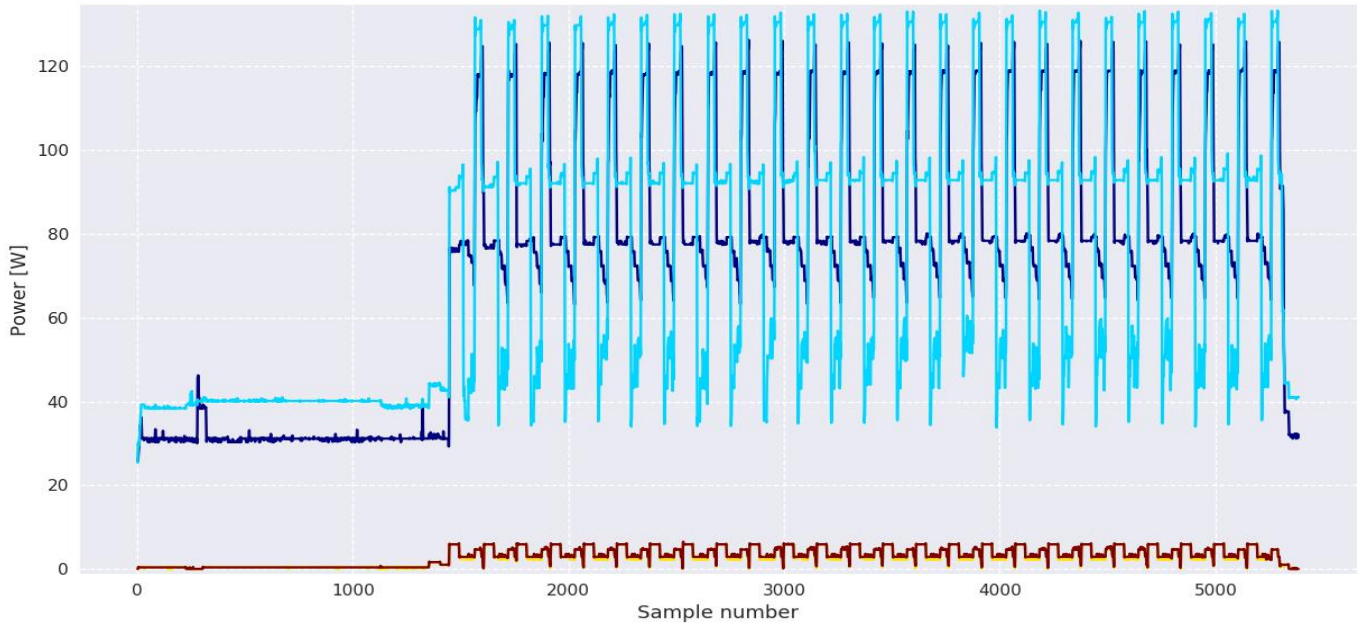


MERIC SW suite



READEX methodology

- H2020 READEX, 2015-2018
- Complex parallel application has different requirements during execution, so it gives a possibility to be dynamically tuned for energy savings without performance penalty.



- Goal was to create a tools-aided methodology for automatic tuning of parallel applications. Dynamically adjust system parameters to actual resource requirements.

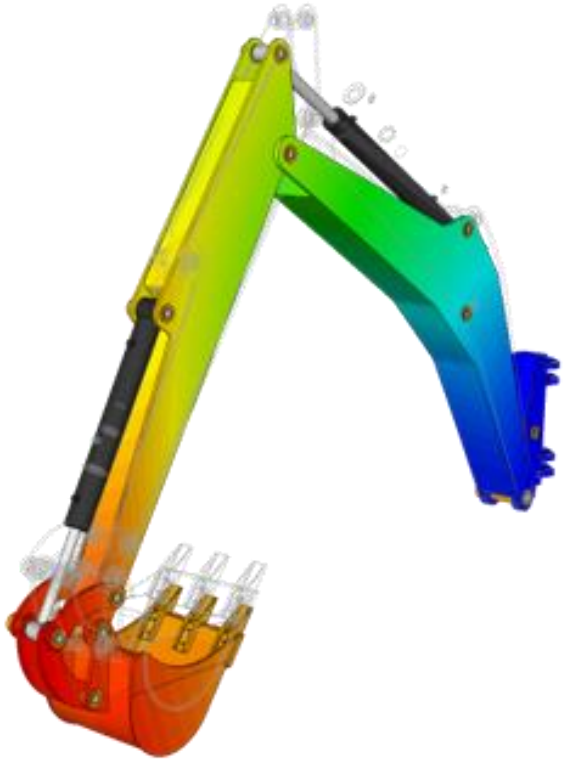
EuroHPC sites energy measurement

	JUPITER	LUMI	Leonardo	MN5	Vega	MeluXina	Karolina	Discoverer	Deucalion
AMD CPU									
intel_rapl kernel module									
amd_energy kernel module									
msr_safe kernel module									
perf									
Intel CPU									
intel_rapl kernel module									
msr_safe kernel module									
perf									
A64FX									
perf									
Nvidia GPU									
NVML									
AMD GPU									
ROCm									
GraceHopper									
HWMON									

EUPEX

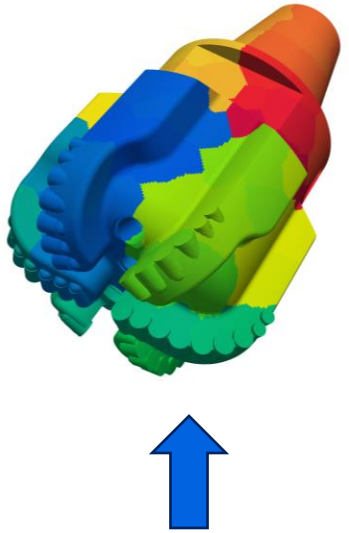
ESPRESO FEM

- Highly parallel framework for engineering application
  - C++, OpenMP, MPI, CUDA, HIP, SYCL
- Input:
  - Unstructured mesh (millions of nodes/elements)
  - Physical parameters, boundary conditions
- Physical simulation:
  - Finite element method
  - FETI solver (Finite Element Tearing and Interconnection)

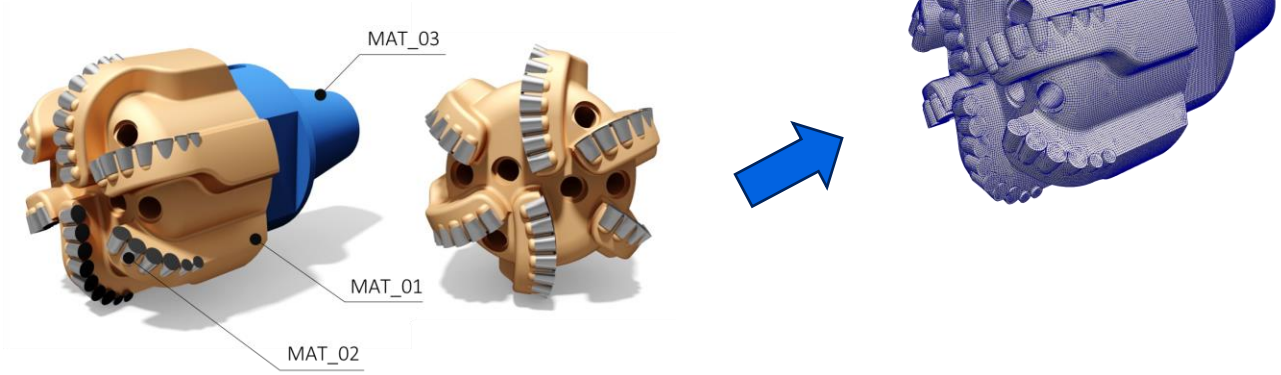


ESPRESO FEM

- FETI solver
  - Massive parallel solver based on methods of domain decomposition
  - Individual domains solved separately by a third-party sparse direct solver
  - Overall solution computed by iterative solver

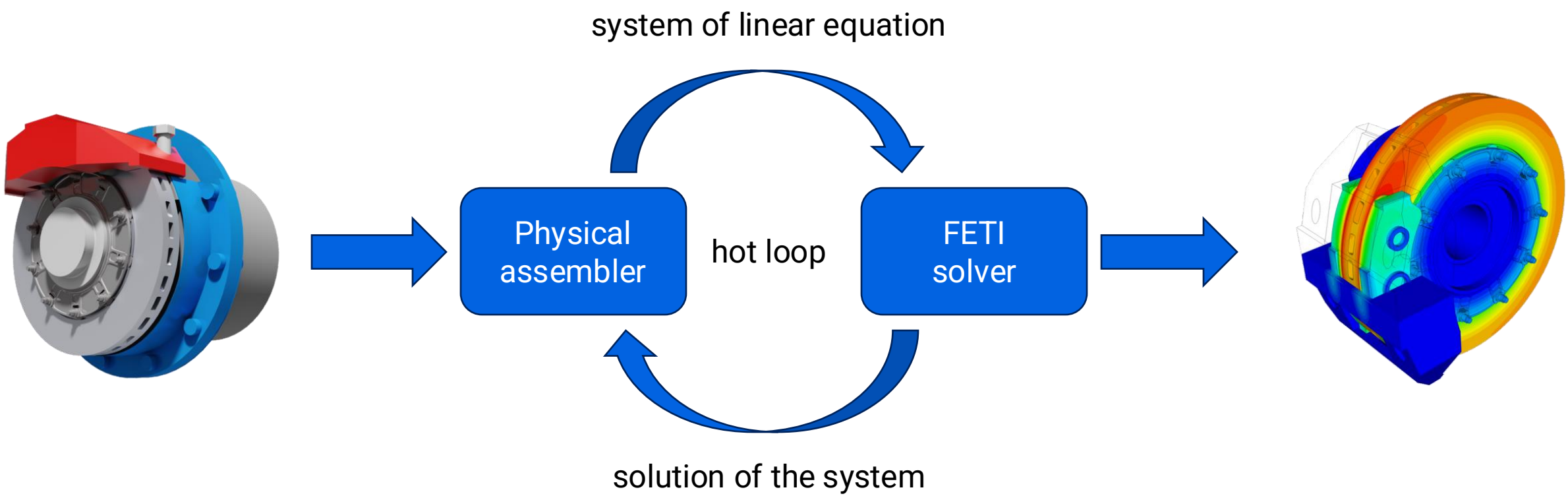


- Scalability approved on many HPC clusters
- EUPEX
  - Single node performance
  - Porting to Arm & GPU acceleration



EUPEX

ESPRESO FEM


system of linear equation
Physical assembler
hot loop
FETI solver
solution of the system

ESPRESO FEM

- Physical assembler
  - Transform physical parameters into a system of linear equations
  - Local matrix kernels applied to each mesh element
  - Cross-element vectorization for optimal utilization SVE instruction set for Arm CPUs

K. Kadlubiak, O. Meca, L. Říha, T. Brzobohatý. **An approach for dynamically adaptable SIMD vectorization of FEM kernels**. Computer Physics Communications, Volume 304, 2024. https://doi.org/10.1016/j.cpc.2024.109319

- **Speedup 2.41 - 5.88 (4.27 at average)**
- The complexity of the kernel is dynamically adapted to the complexity of a solved physical phenomena

ESPRESO FEM

- FETI solver
  - Solve a sparse system of linear equations
  - GPU acceleration: computation of Schur complement of sparse domain matrices

J. Homola, R. Vavřík, O. Meca, T. Brzobohatý, L. Říha. **Assembly of FETI dual operator using CUDA**. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Milano, Italy, 2025, pp. 365-374, DOI: 10.1109/IPDPSW66978.2025.00062.

- Fine-tuned utilization of cuBLAS and cuSPARSE kernels
- **Up to 10 speedup of GPU preprocessing**

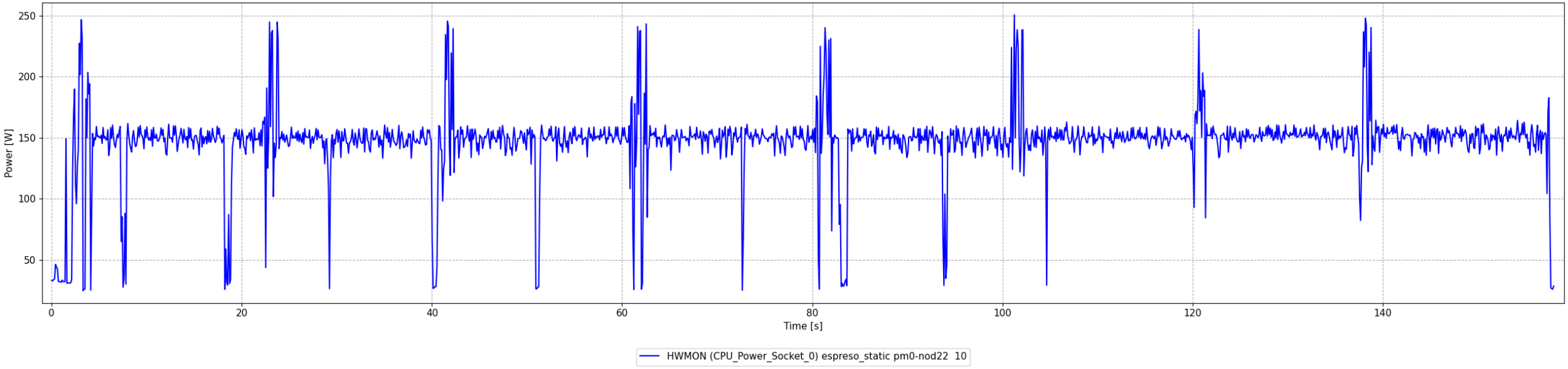
ESPRESO FEM

- FETI solver
  - Solve a sparse system of linear equations
  - GPU acceleration: computation of Schur complement of sparse domain matrices

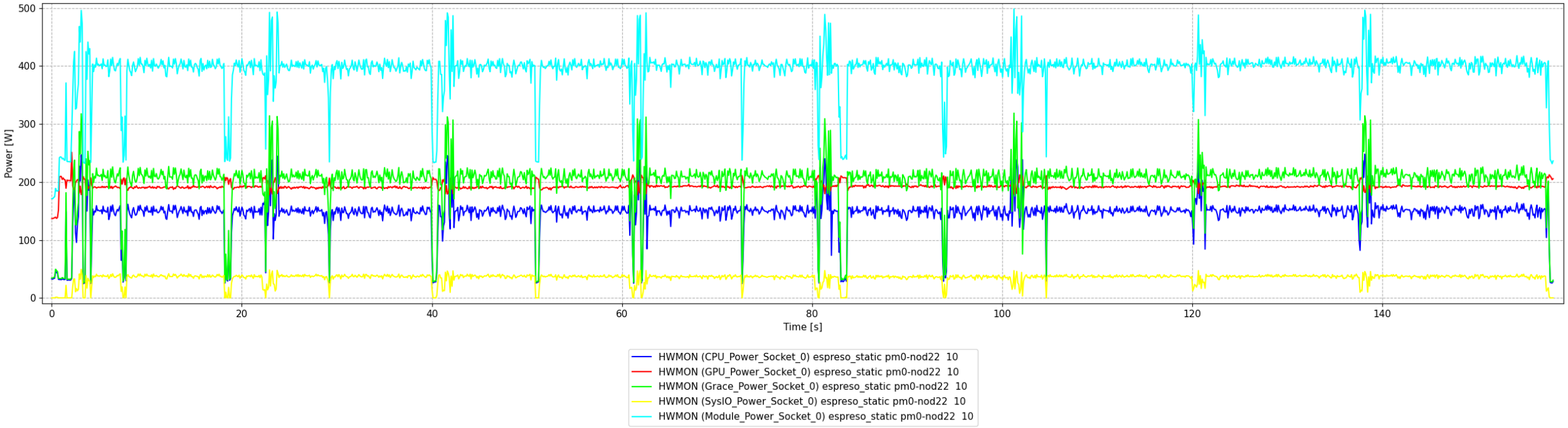
J. Homola, O. Meca, T. Brzobohatý, L. Říha. **Utilizing Sparsity in the GPU-accelerated Assembly of Schur Complement Matrices in Domain Decomposition Methods.** Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2025, pp. 1464 – 1476. DOI: 10.1145/3712285.3759904.

- Utilization of a sparsity pattern in domain decomposition methods to improve acceleration
- **Up to 3.3 speedup for large matrices**

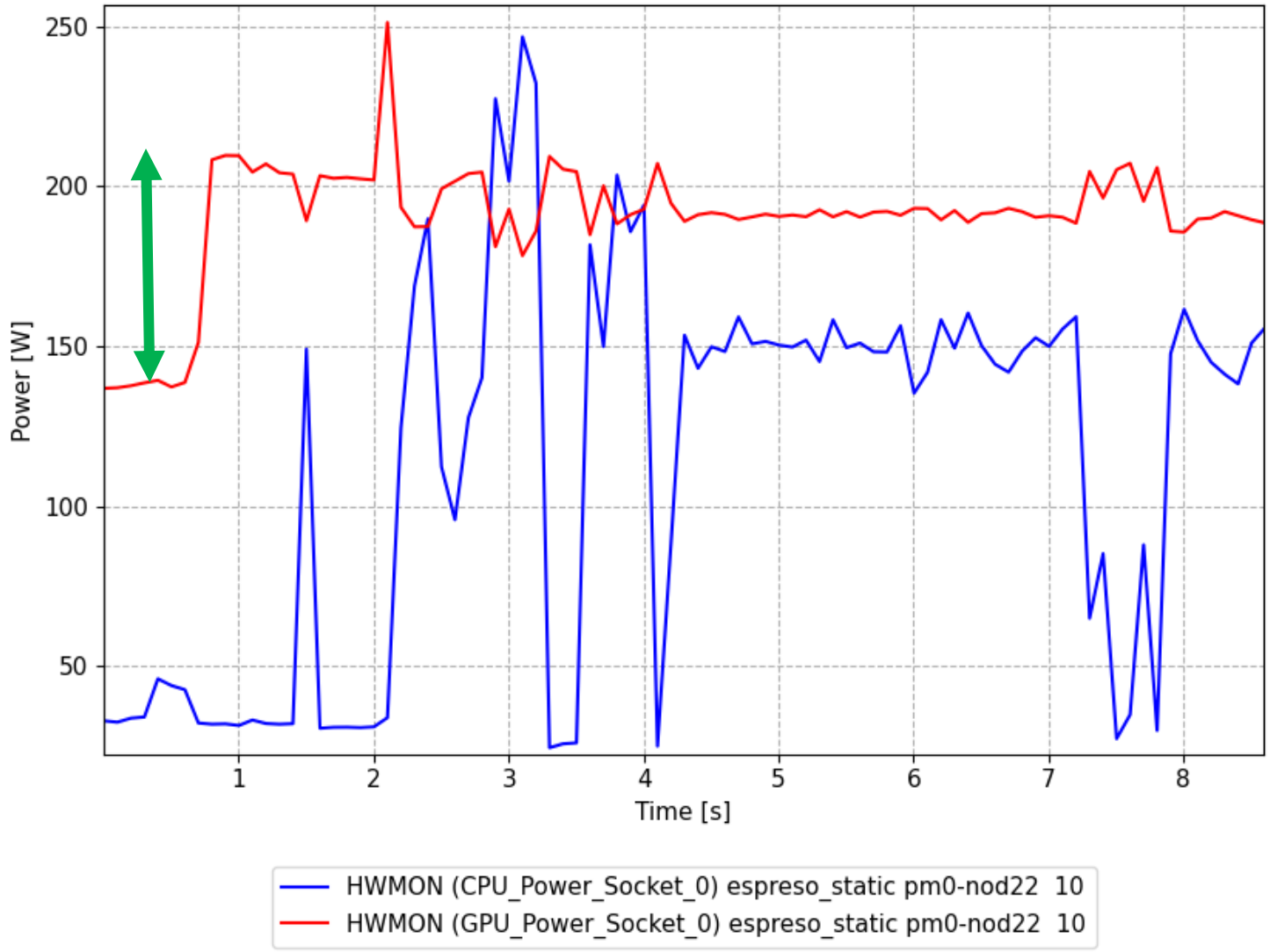
CPU workload - CPU power consumption



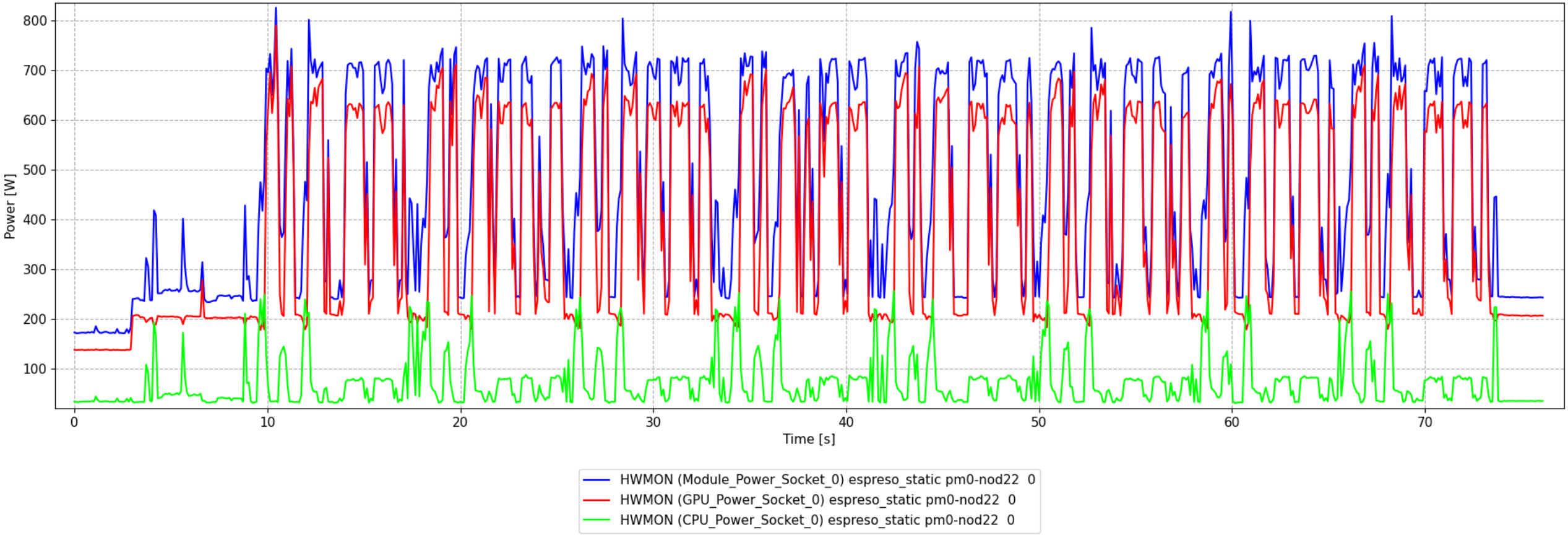
CPU workload - GH200 power consumption



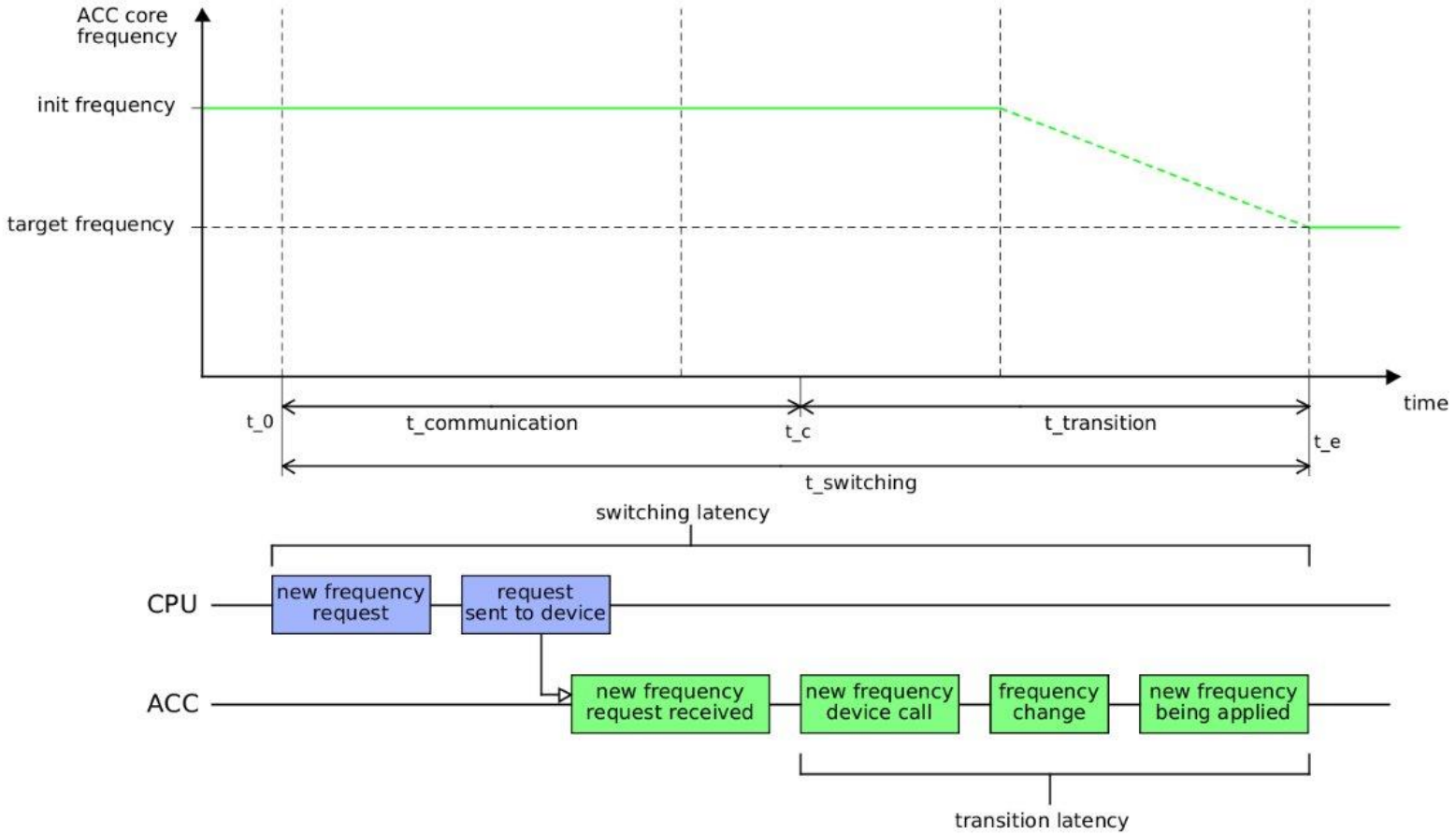
Idle power consumption



GPU workload - power consumption



GPU frequency change



Visualization of CPU to ACC communication while issuing the ACC frequency change request. The dashed line shows the frequency change.

GPU switching latency

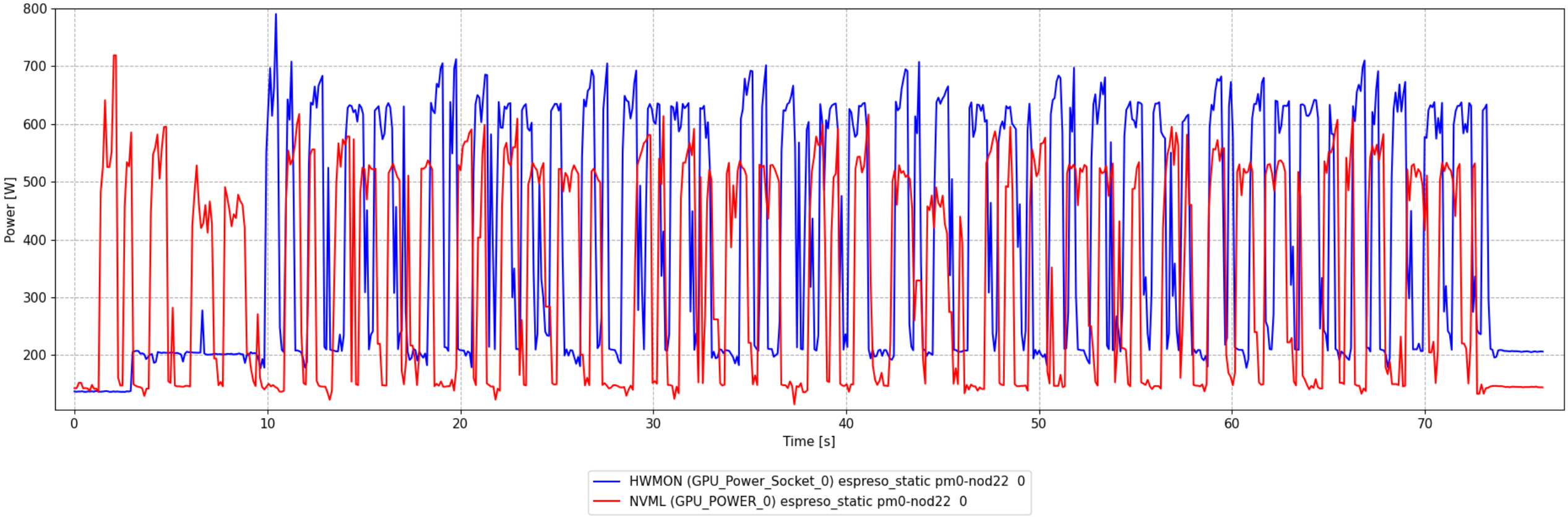
Initiabl Frequency [MHz] \ Target Frequency [MHz]	705	795	885	975	1095	1170	1260	1275	1290	1350	1410	1500	1665	1770	1830	1875	1920	1980
705		12.370	24.447	10.810	12.011	11.304	245.369	260.804	10.902	12.913	18.105	18.417	17.168	24.686	22.362	10.648	261.650	22.346
795	9.999		18.865	20.843	9.912	9.620	245.403	201.442	9.711	14.533	22.467	22.072	18.362	17.251	15.133	12.962	246.805	22.572
885	12.275	15.174		16.235	11.466	15.884	245.967	256.423	15.500	12.634	14.566	15.200	13.015	16.203	13.015	14.424	294.722	32.591
975	22.044	8.578	23.053		10.730	13.809	246.252	262.165	24.325	21.214	11.174	22.791	13.539	18.346	10.954	16.261	290.120	24.026
1095	13.463	22.892	10.898	15.285		13.977	477.318	263.368	10.369	14.203	11.363	12.715	11.490	15.582	11.562	10.375	302.414	19.479
1170	22.652	14.539	18.161	13.348	12.661		245.255	261.746	20.976	21.091	21.628	12.720	12.777	22.192	17.463	22.192	289.421	20.285
1260	21.902	17.604	18.593	17.761	20.199	18.610		17.852	20.461	19.380	16.872	11.227	18.456	18.743	17.533	19.690	304.845	14.879
1275	23.206	20.660	22.838	20.896	22.629	23.195	24.814		23.143	20.280	15.901	22.285	20.655	19.650	21.766	20.122	303.514	13.227
1290	9.704	20.108	20.516	8.661	19.967	21.387	11.955	9.826		10.780	16.712	18.380	10.740	17.160	146.854	25.762	306.059	174.150
1350	21.346	20.448	21.407	15.290	22.037	16.080	281.823	270.188	21.962		5.572	11.152	20.952	19.520	18.065	19.853	303.097	20.524
1410	7.144	10.168	17.922	11.711	7.708	10.455	289.860	288.611	40.594	89.338		193.691	207.739	32.955	206.769	201.386	290.660	210.565
1500	20.659	21.880	7.269	12.714	22.527	13.647	214.905	214.531	22.739	13.191	9.376		12.751	19.202	11.235	20.709	302.169	12.840
1665	12.943	21.323	21.112	6.836	12.932	7.181	307.031	294.634	16.823	10.449	20.144	24.922		16.539	14.068	14.518	255.089	24.521
1770	10.340	23.587	16.131	18.074	23.233	20.063	113.783	302.975	37.616	43.385	20.048	203.689	203.899		85.533	159.767	301.611	206.412
1830	10.835	8.987	13.653	13.675	10.687	15.608	464.768	23.398	15.829	6.459	14.277	20.801	16.805	10.652		16.332	282.039	14.175
1875	20.748	20.862	21.918	23.699	21.647	23.388	16.524	450.205	24.392	14.387	12.933	12.984	9.518	20.437	11.159		274.134	16.785
1920	15.406	11.156	13.092	13.803	11.025	22.658	23.297	10.878	11.264	10.358	11.811	20.946	10.871	10.320	13.080	20.142		10.916
1980	18.167	22.732	6.381	10.961	23.524	11.712	8.805	411.839	23.071	11.255	10.035	20.425	11.432	20.486	10.398	20.164	298.295	

GH200

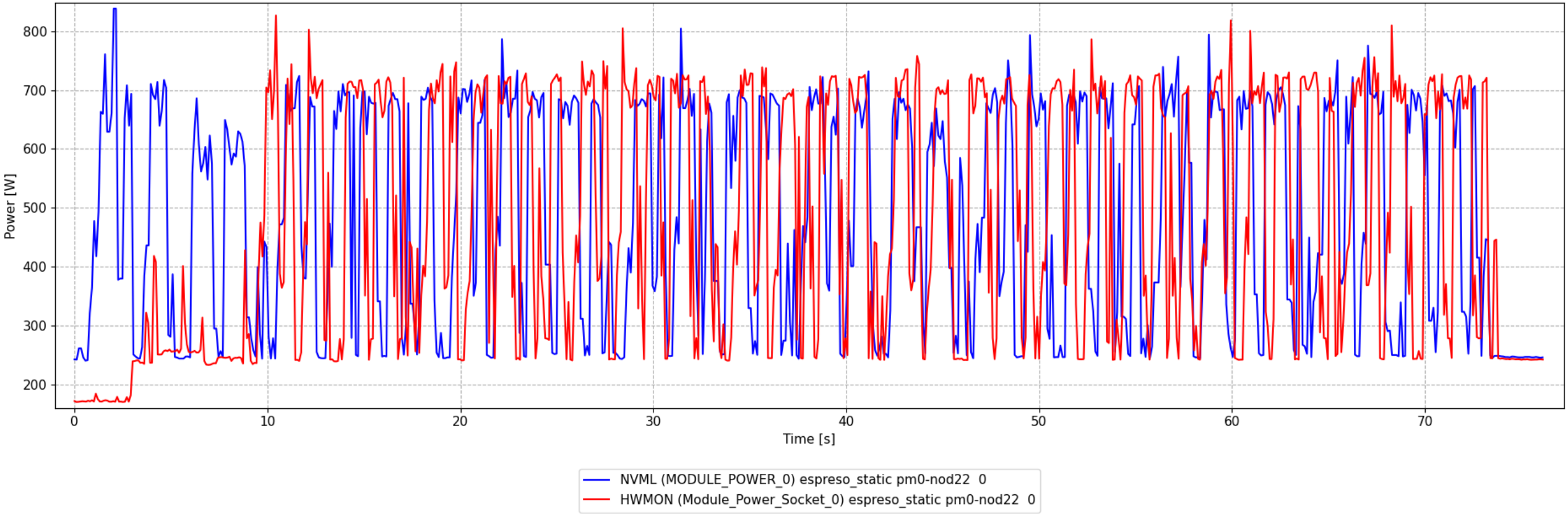
5.6 - 477.3 ms (worst case)

D. Velicka, O. Vysocky, O. Yasal, L. Riha "An In-Depth Study of GPU Frequency-Scaling Latency and Its Optimization on Modern Architectures", Future Generation Computer Systems, *179 (2025): 108331*, https://doi.org/10.1016/j.future.2025.108331

HWMON vs NVML (GPU)



HWMON vs NVML (Module)



Energy measurement

- Module NVML energy - 23.970 kJ
- Module NVML power - 37.154 kJ
- Module HWMON - 36.398 kJ
- GPU HWMON - 30.215 kJ
- ***Which one should we trust?***

Conclusion

- ESPRESO FEM shows how to optimize for Arm CPU and/or GPU
  - Performance optimization is the best approach how to improve energy efficiency
  - Despite performance optimizations it gives opportunity to improve energy usage
- **Initial** results of ESPRESO energy consumption optimization in GH200 (without performance penalty)
  - CPU run – 2%
  - GPU run – 8%
- GraceHopper (GraceBlackwell) power monitoring and power management is tricky
  => Further validation study is necessary