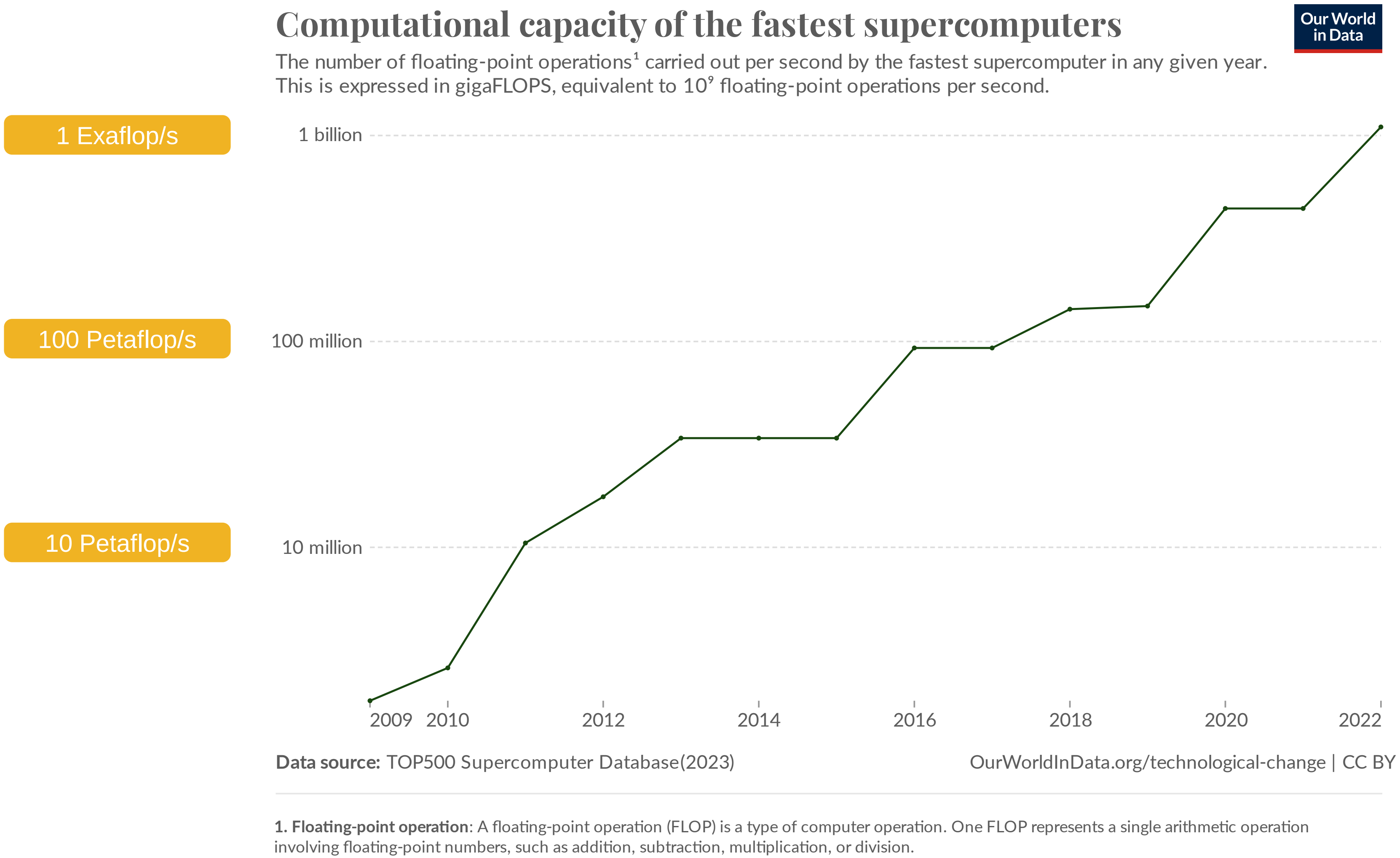
GREEN SOFTWARE ENGINEERING

Prof Michèle Weiland
m.weiland@epcc.ed.ac.uk

The rise and rise of HPC



Every tonne of CO_2 emissions adds to global warming

Global surface temperature increase since 1850–1900 (°C) as a function of cumulative CO_2 emissions ($GtCO_2$)

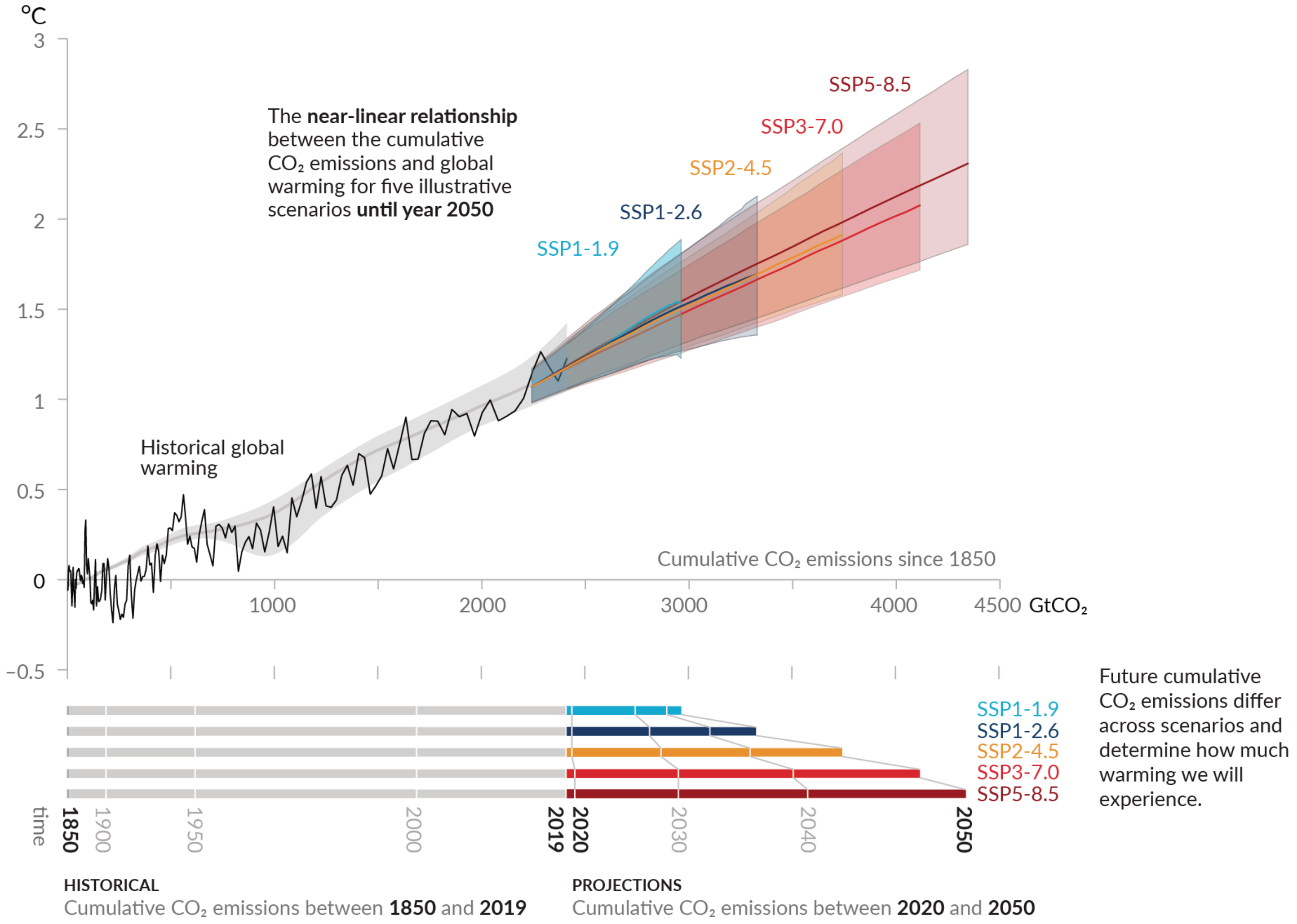


Figure SPM.10 in IPCC, 2021: Summary for Policymakers. In: ***Climate Change 2021: The Physical Science Basis***. *Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change* [Masson-Delmotte, V., P. Zhai, A. Pirani, S.L. Connors, C. Péan, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M.I. Gomis, M. Huang, K. Leitzell, E. Lonnoy, J.B.R. Matthews, T.K. Maycock, T. Waterfield, O. Yelekçi, R. Yu, and B. Zhou (eds.)]. Cambridge University Press, Cambridge, UK and New York, NY,USA, pp. 3–32, doi: 10.1017/9781009157896.001

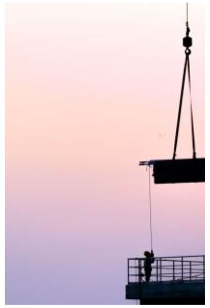
CO^2 emissions from HPC & AI infrastructure

Electricity

- Operation
- Cooling

Manufacturing

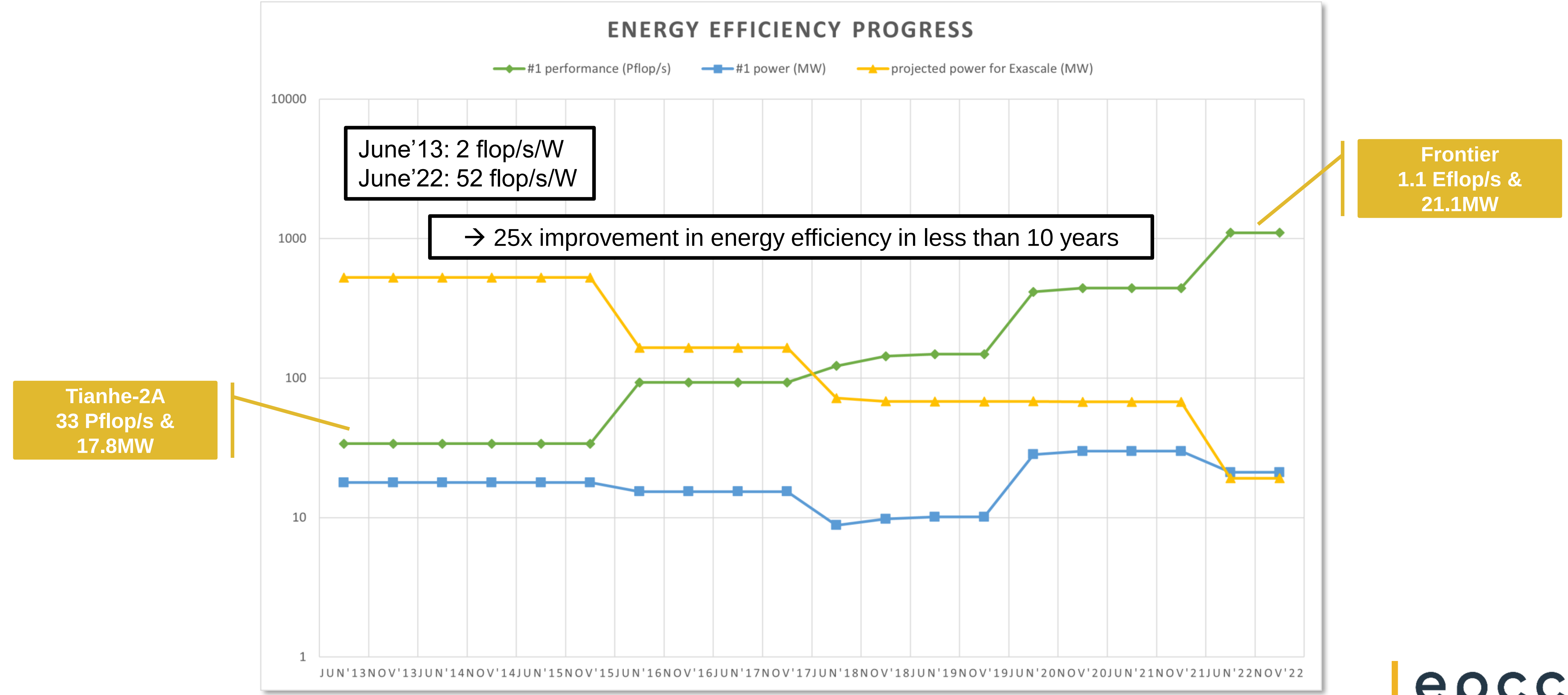
- Computer hardware
- Infrastructure equipment



Construction

- Data centres

Progress in hardware energy efficiency in the past 10 years…

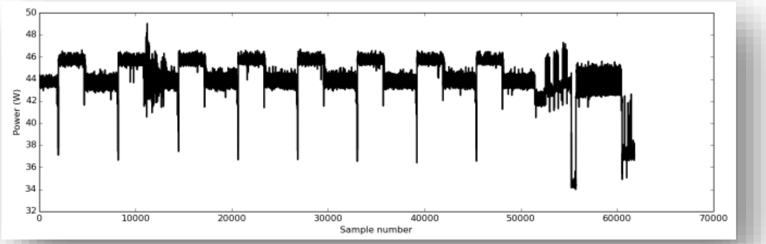


Green software engineering

Energy = Power × time
Joules = Watts × seconds

- Hardware is becoming more efficient – what about software?
  1. Minimising power draw?
  2. Minimising energy use?
  3. Minimising emissions?
  4. Maximising science throughput & utilisation?

➔ Different targets, which require different approaches

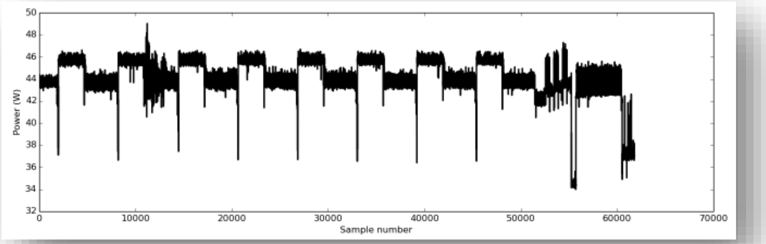


epcc

1. Minimising power draw

Energy = Power × time
Joules = Watts × seconds

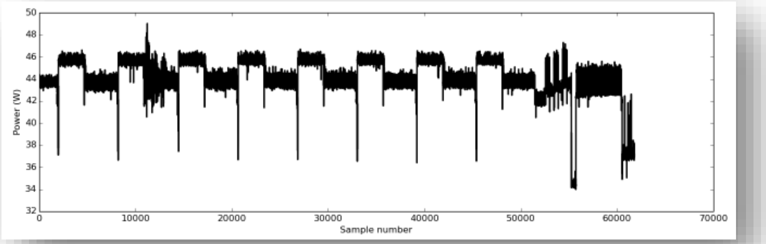
- Reason: power cap (e.g. infrastructure limitations)
- Applications should draw as little power as possible
  - Even at the expense of using more energy
- Avoid power-hungry operations
  - E.g. vector instructions where there is no performance benefit
  - Moving data is cheap in terms of power (compared to compute)



Energy = Power × time
Joules = Watts × seconds

2. Minimising energy use

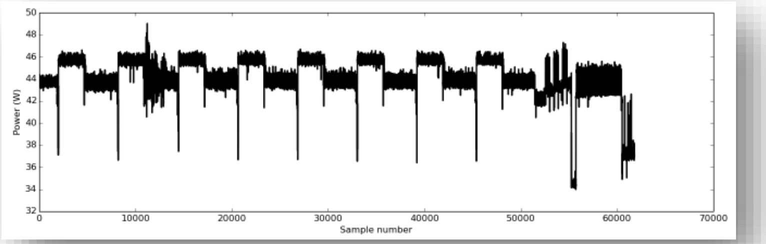
- Reason: operational cost reduction
- Applications use as little energy as possible to get result
  - Even at the expense of using more power
- Optimising runtime is a key (though not the only) factor
  - E.g. recomputing data preferrable to moving data



3. Minimising emissions

Energy = Power × time
Joules = Watts × seconds

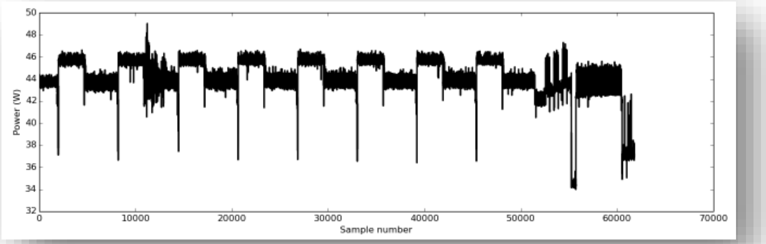
- Reason: sustainability
- Becoming more complex now…
- Emissions do not only depend on the application, but where/when it is run
- However, an efficient application will inherently incur lower emissions than an inefficient one



4. Maximising science throughput

Energy = Power × time
Joules = Watts × seconds

- Reason: getting the most out of investment
- Applications use as much energy as they need to get results fast
  - Power and energy use are secondary to runtime
- Optimising runtime & parallel efficiency are key factors
  - Requires understanding of scaling behaviour



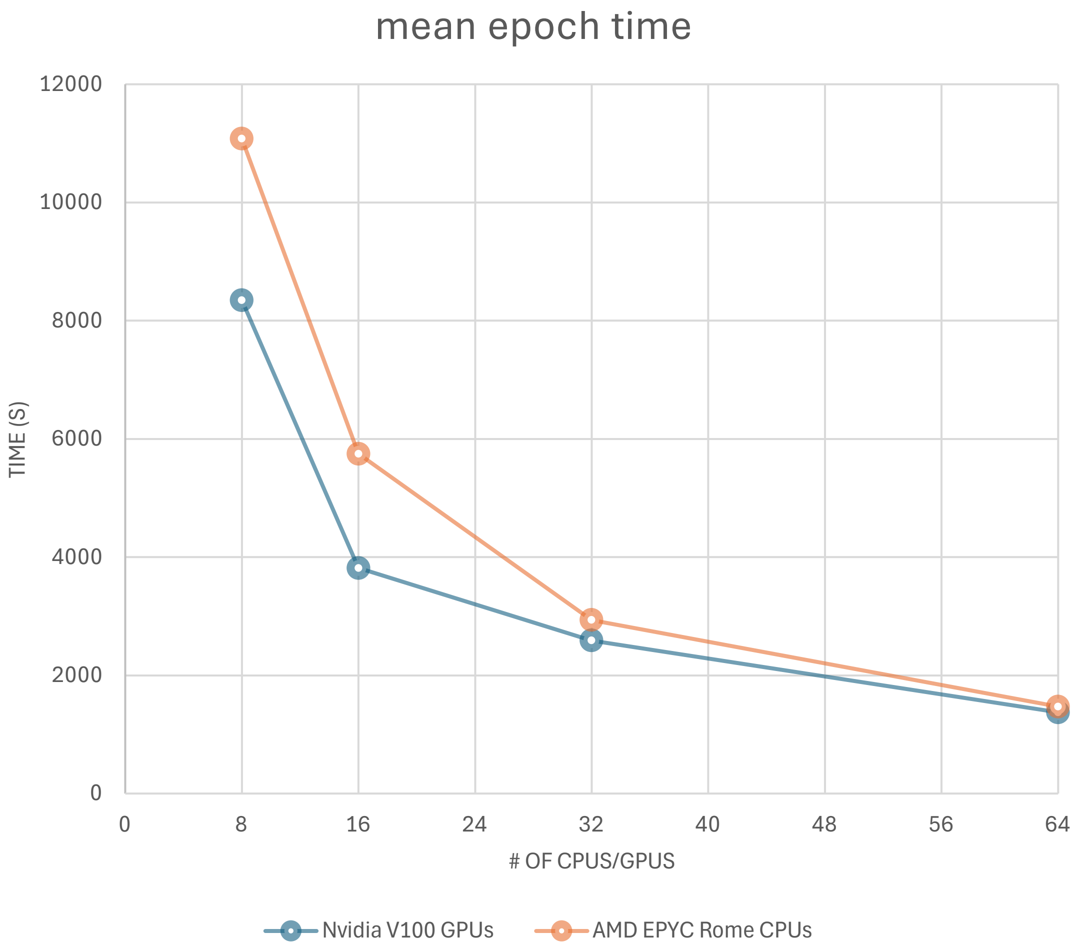
epcc

Pre-requisites to green software engineering

- Impossible to understand how to improve efficiency without performance ***and power*** data
- But can be tricky to get access to accurate power readings
  - Especially on new architectures or in Cloud environments
  - Consistency of data is not guaranteed

➔ This *must* be made simpler

MLPerfHPC - Cosmoflow



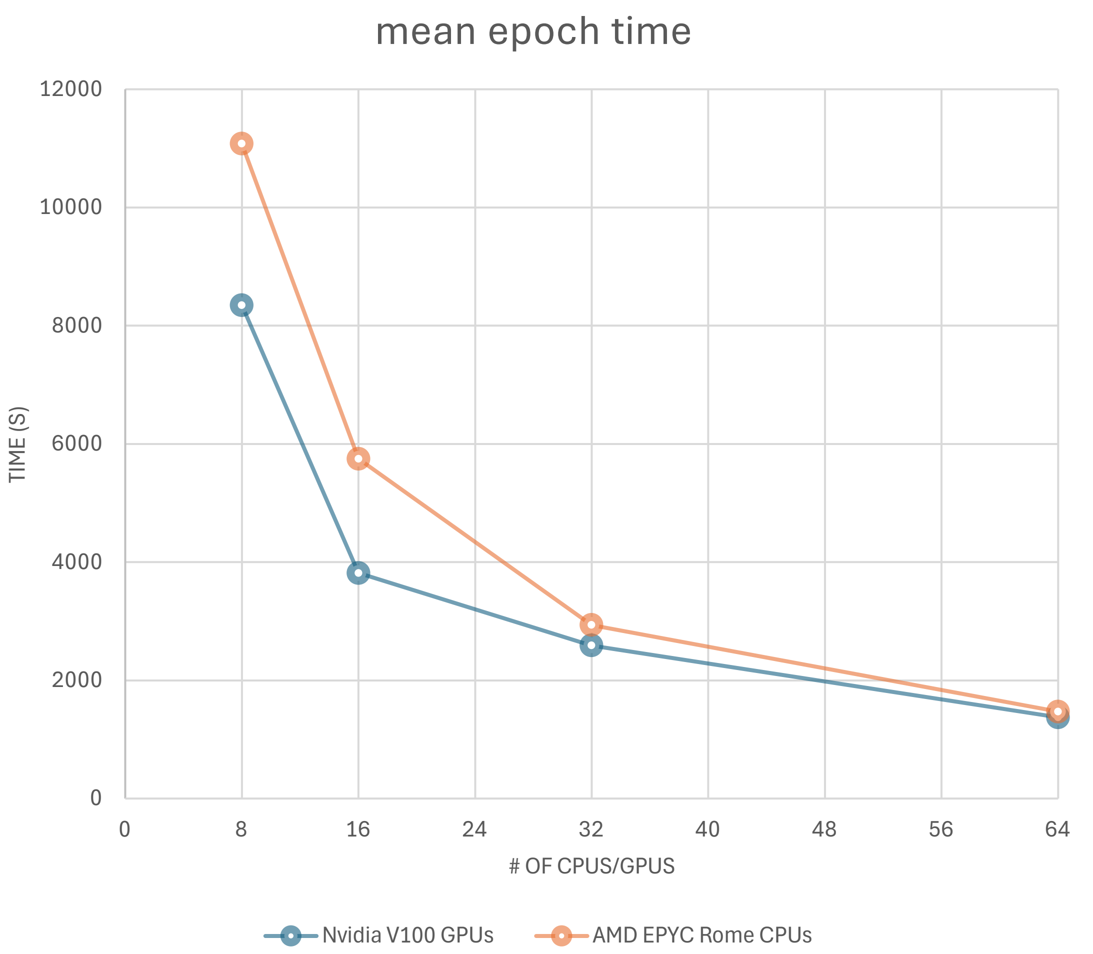
- 3D CNN that estimates initial conditions of the universe based on simulations of distributed matter
- TensorFlow with Keras, uses Horovod for distributed training
- Full dataset is 1.7 TB
  - 524,288 training samples and 65,536 validation samples

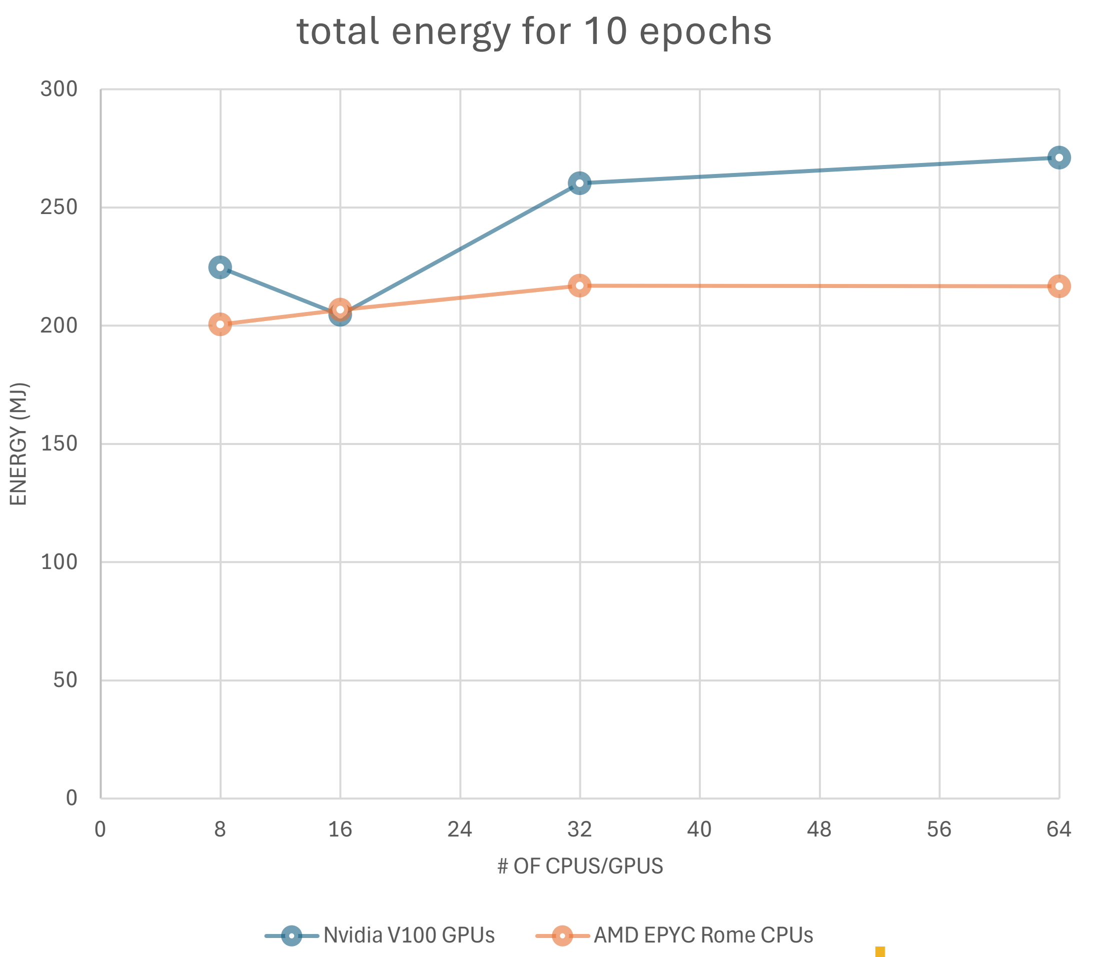
- Comparing two systems
  - HPE EX with AMD EPYC Rome CPUs
    - Two 64-core CPUs per node
    - Average power consumption: ~220W per CPU
    - *Power measurements for full node*
  - HPE ICE XA with Intel Skylake CPUs and Nvidia V100 GPUs
    - Four GPUs per node
    - Average power consumption: 320W per GPU
    - *Power measurements do not include CPUs*

MLPerfHPC - Cosmoflow

- GPU system: better initial performance, but worse scaling
- CPU system: close to GPU performance at scale → better network, better I/O

➔ Is it a reasonable comparison? Full node power (ARCHER2) vs GPUs only (Cirrus)

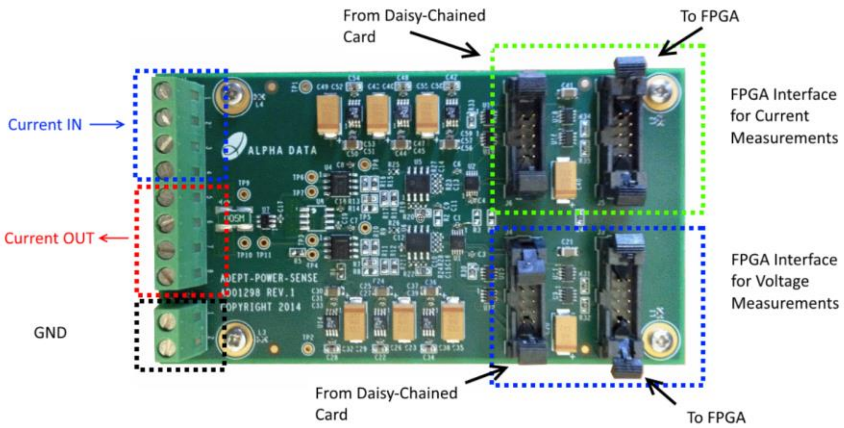




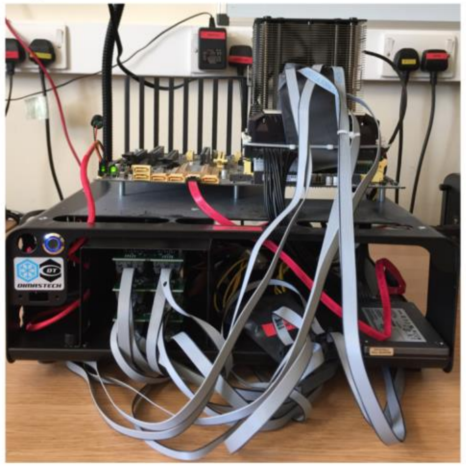
epcc

Measuring power & energy

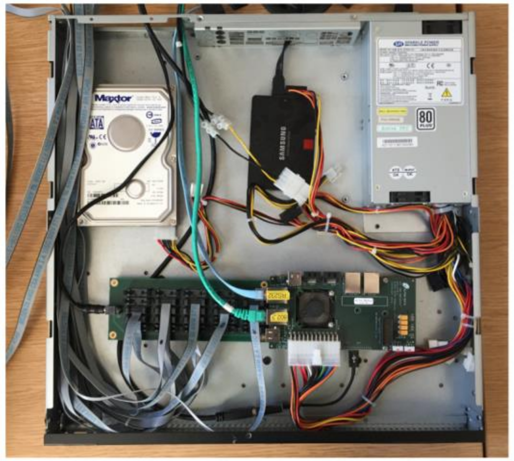
addressing energy in parallel technologies

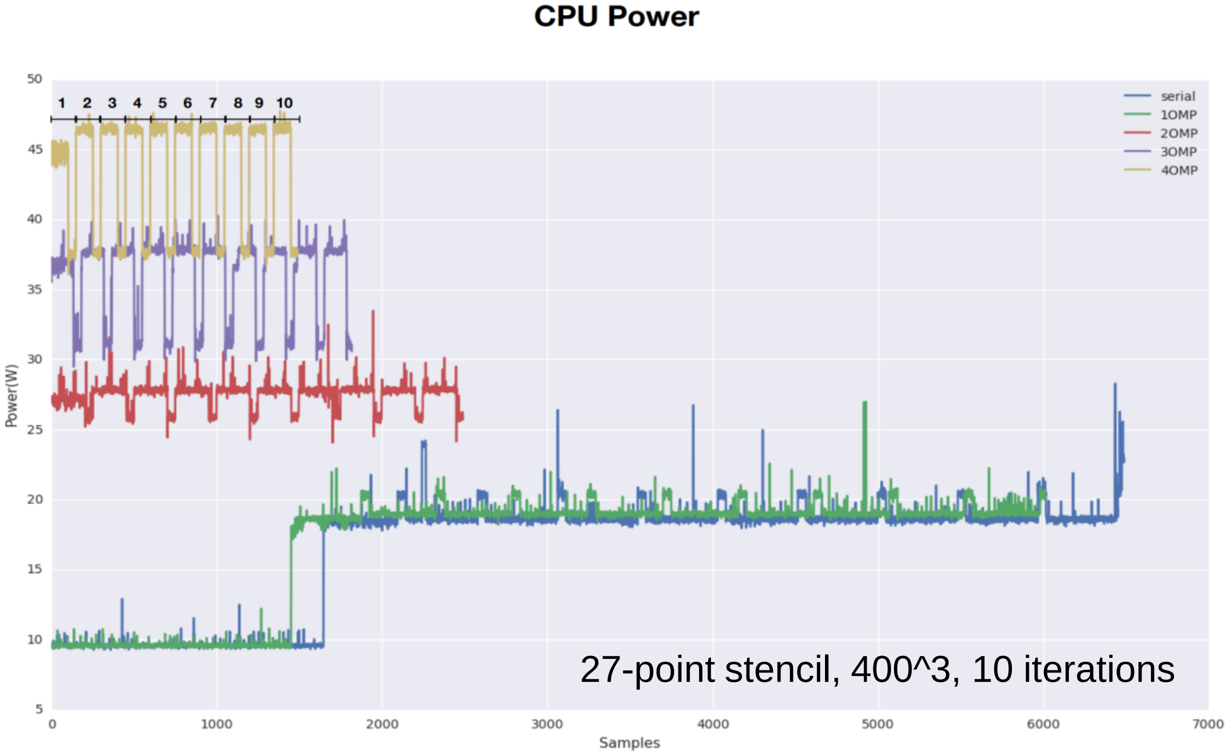


System under test

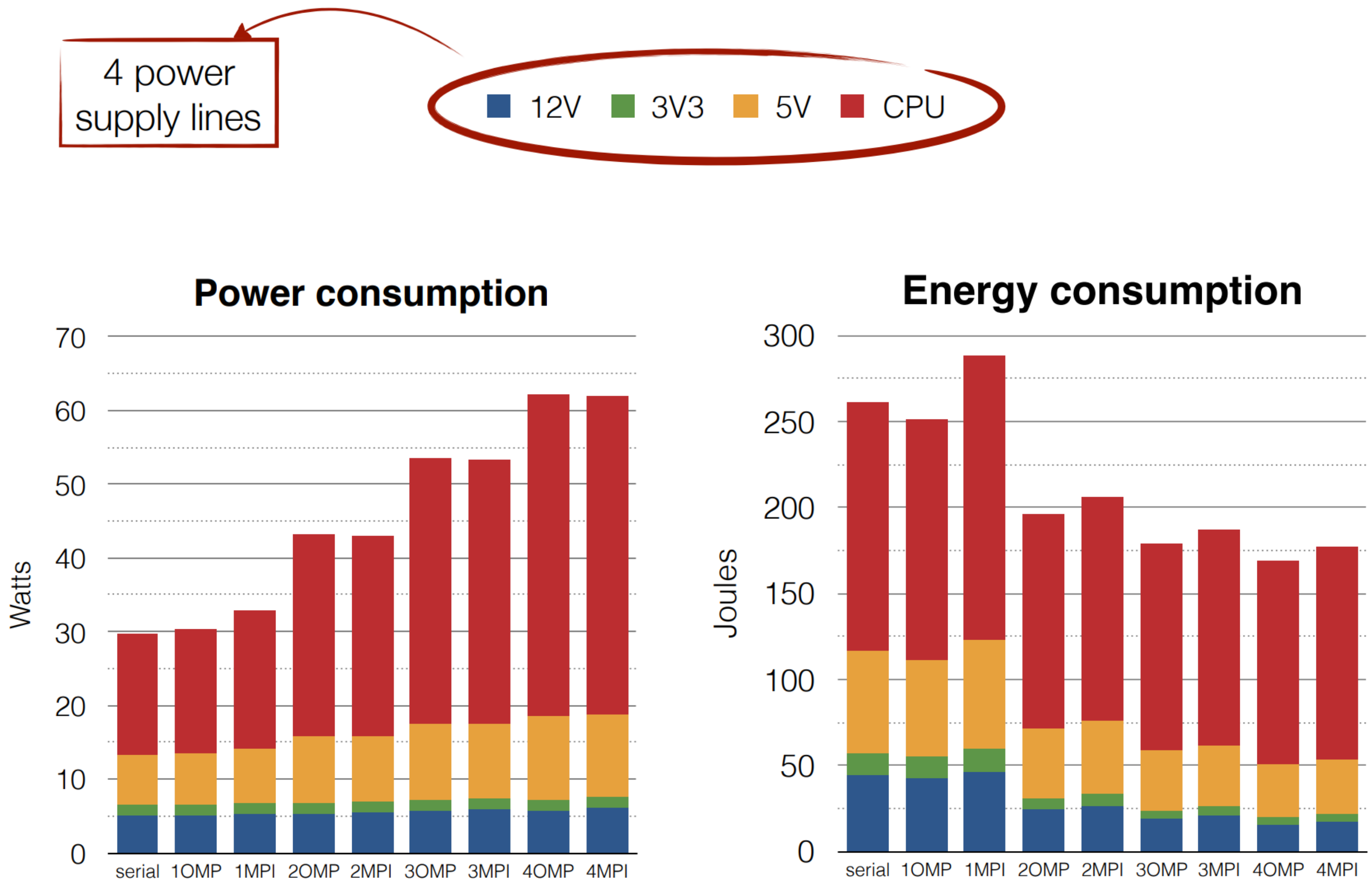


Measurement control





Adept
addressing energy in parallel technologies

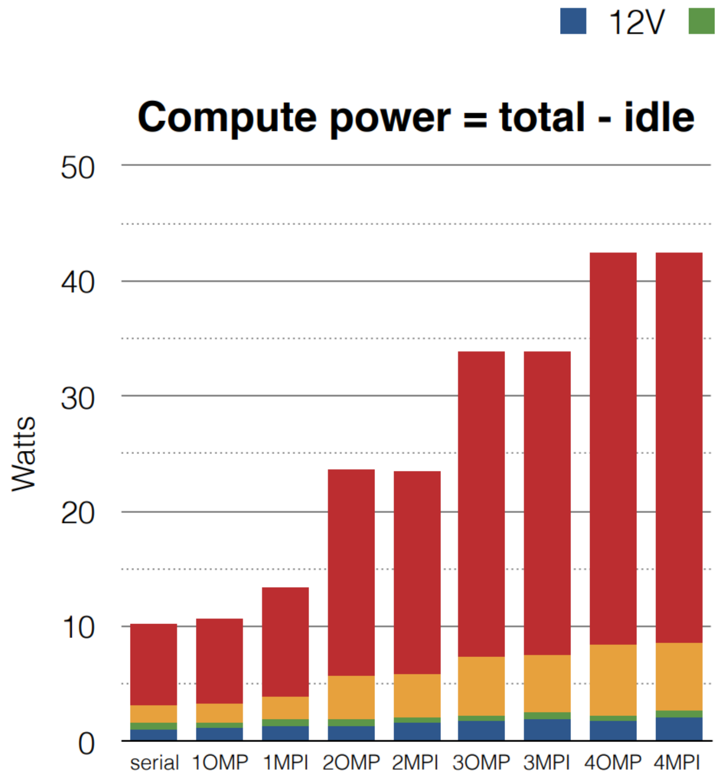


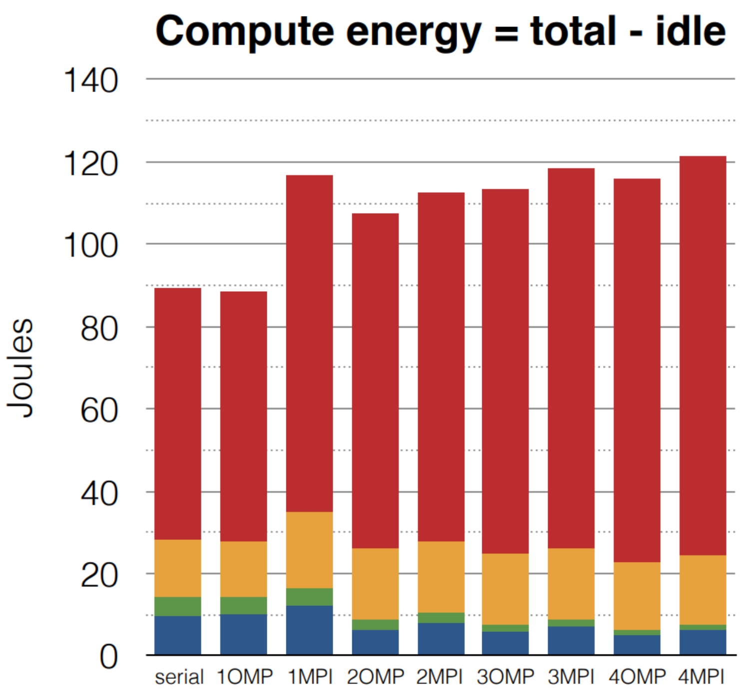
*Energy = Power * time*

Adept
addressing energy in parallel technologies

***Remember**: system consumes power even if it is not doing anything "useful"!*

idle power:
12V=3.916W
3V3=0.97W
5V=5.189W
CPU=9.508W



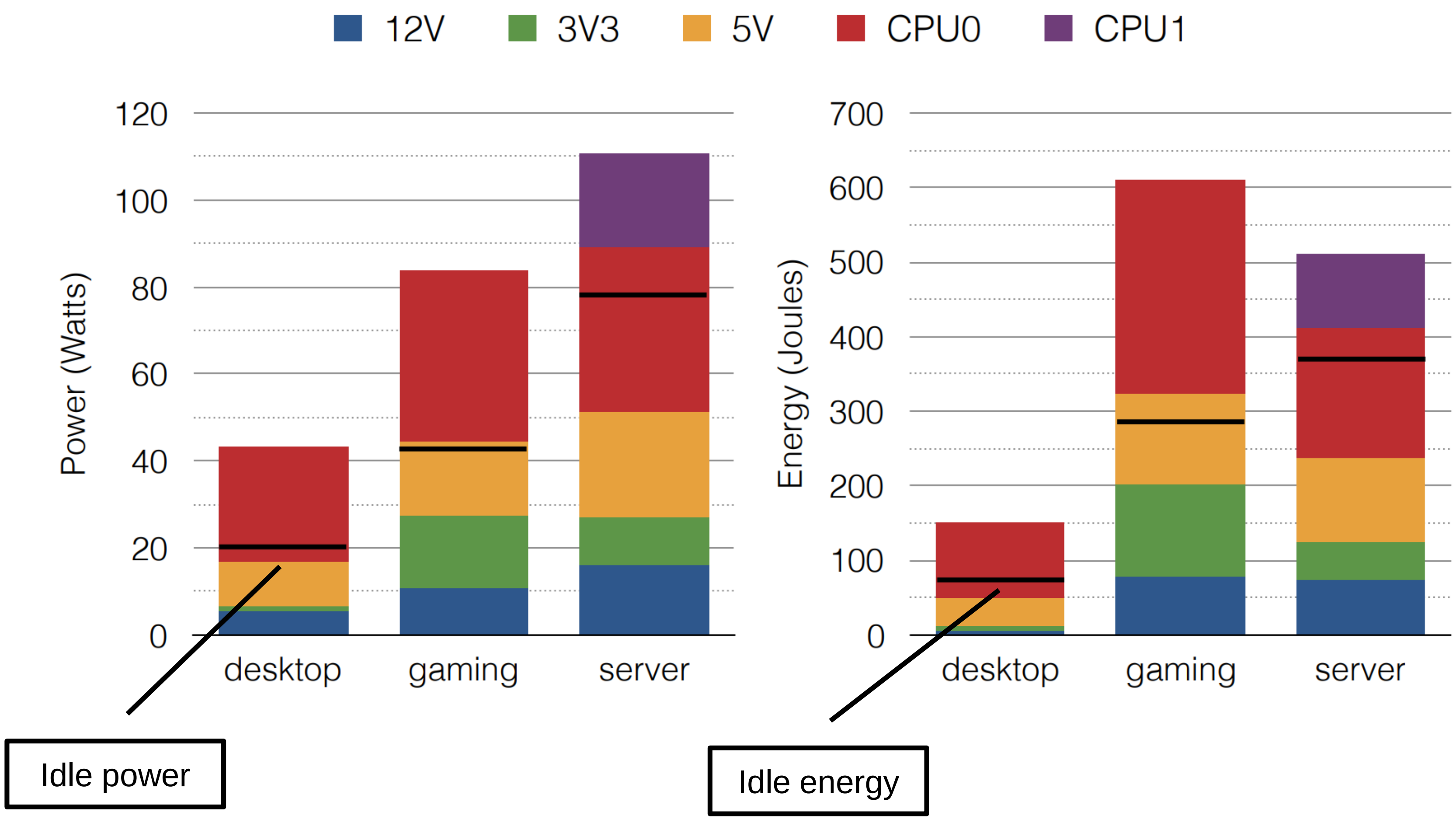


No amount of green software engineering can change the idle power/energy

$Power_{compute} = Power_{total} - Power_{idle}$

$Energy_{compute} = Power_{compute} * time$

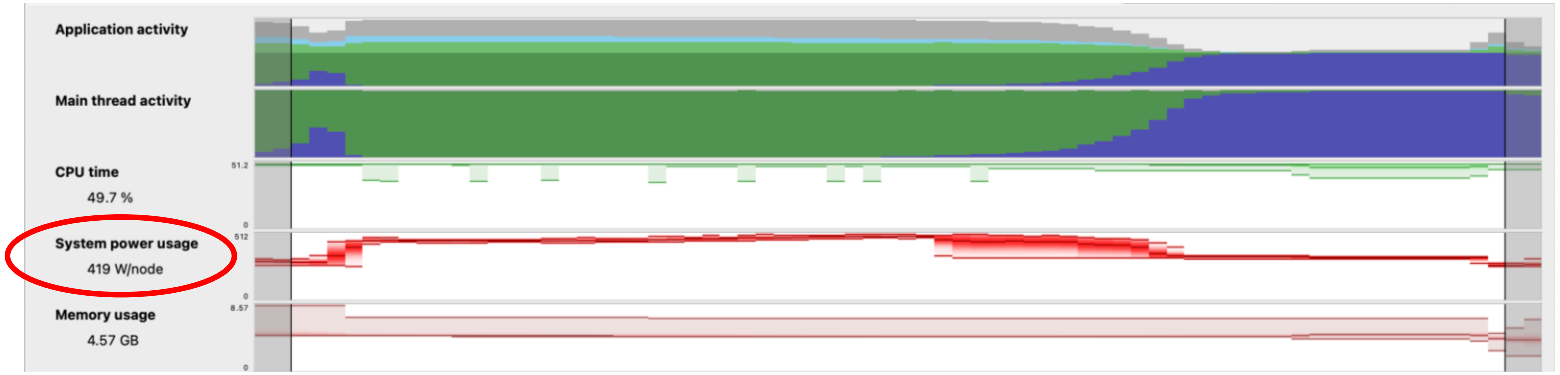
Comparing identical workload on different systems



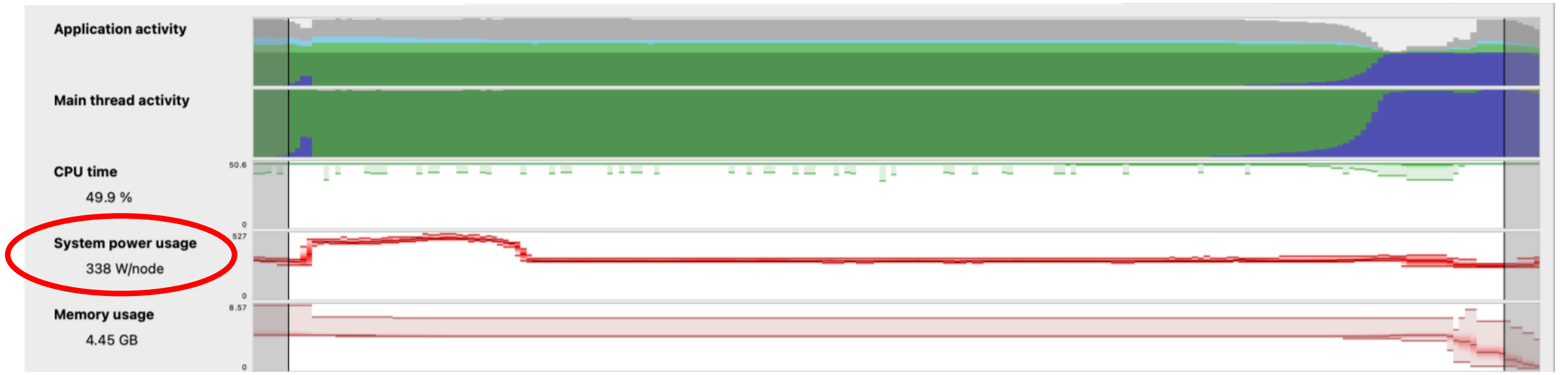
No amount of green software engineering can change the idle power/energy

Choice of algorithms

Reverse Cuthill-McKee



Space-filling curve (zcurve)



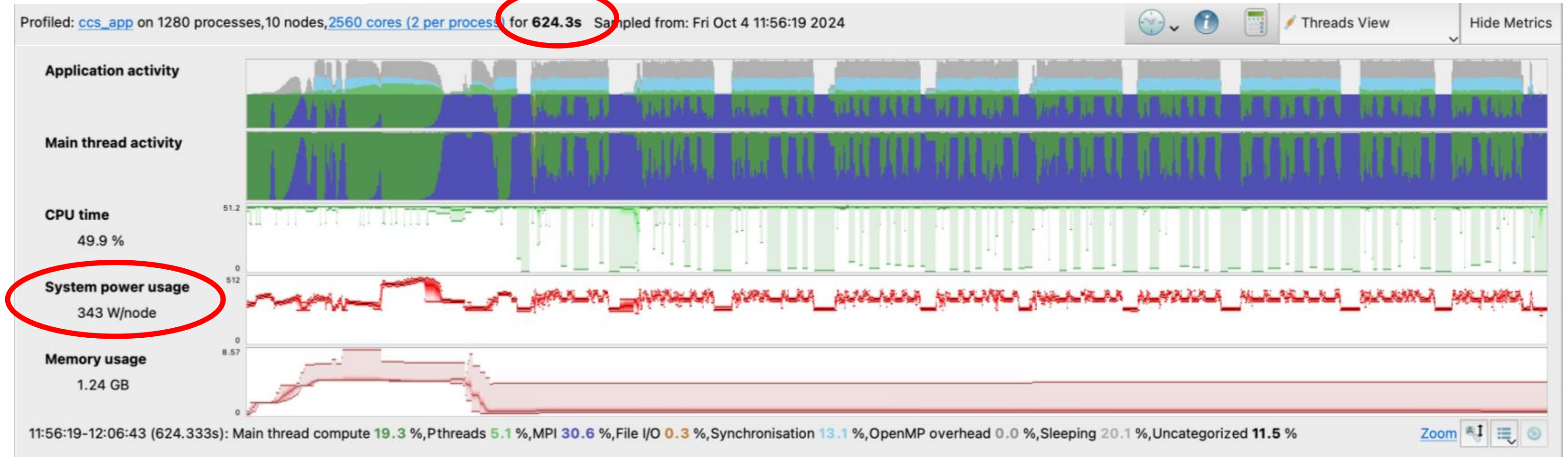
- CFD application – performs reordering on the mesh
- Taylor-Green Vortex on 400^3 mesh
  - 10 nodes of ARCHER2
- Different algorithms available for reordering
  - RCM
  - zcurve

Choice of algorithms – the full picture

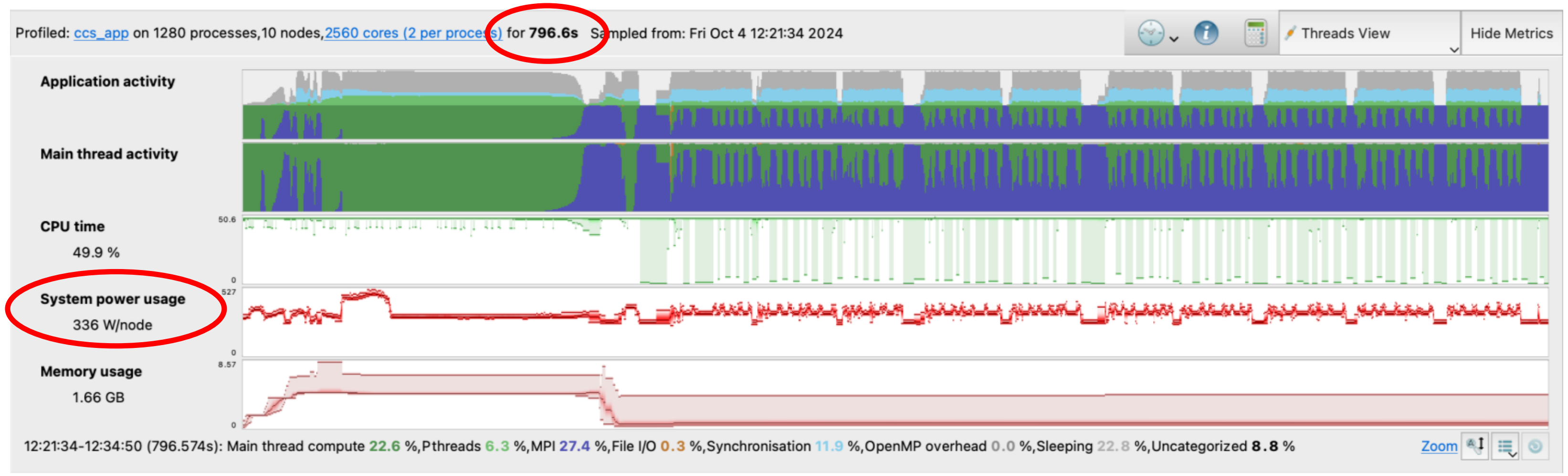
```
mwrr@ln04:> sacct -j 7741382 --format=JobID,JobName,ElapsedRaw,NNodes,ConsumedEnergy
JobID           JobName ElapsedRaw   NNodes ConsumedEnergy
------------ ---------- ---------- -------- --------------
7741382             tgv       1007       10          3.02M
7741382.bat+      batch       1007        1        304.71K
7741382.ext+     extern       1007       10          3.02M
7741382.0    forge-bac+        964       10          2.92M

mwrr@ln04:> sacct -j 7741587 --format=JobID,JobName,ElapsedRaw,NNodes,ConsumedEnergy
JobID           JobName ElapsedRaw   NNodes ConsumedEnergy
------------ ---------- ---------- -------- --------------
7741587             tgv       1196       10          3.57M
7741587.bat+      batch       1196        1        361.27K
7741587.ext+     extern       1196       10          3.57M
7741587.0    forge-bac+       1154       10          3.48M
```

Reverse Cuthill-McKee

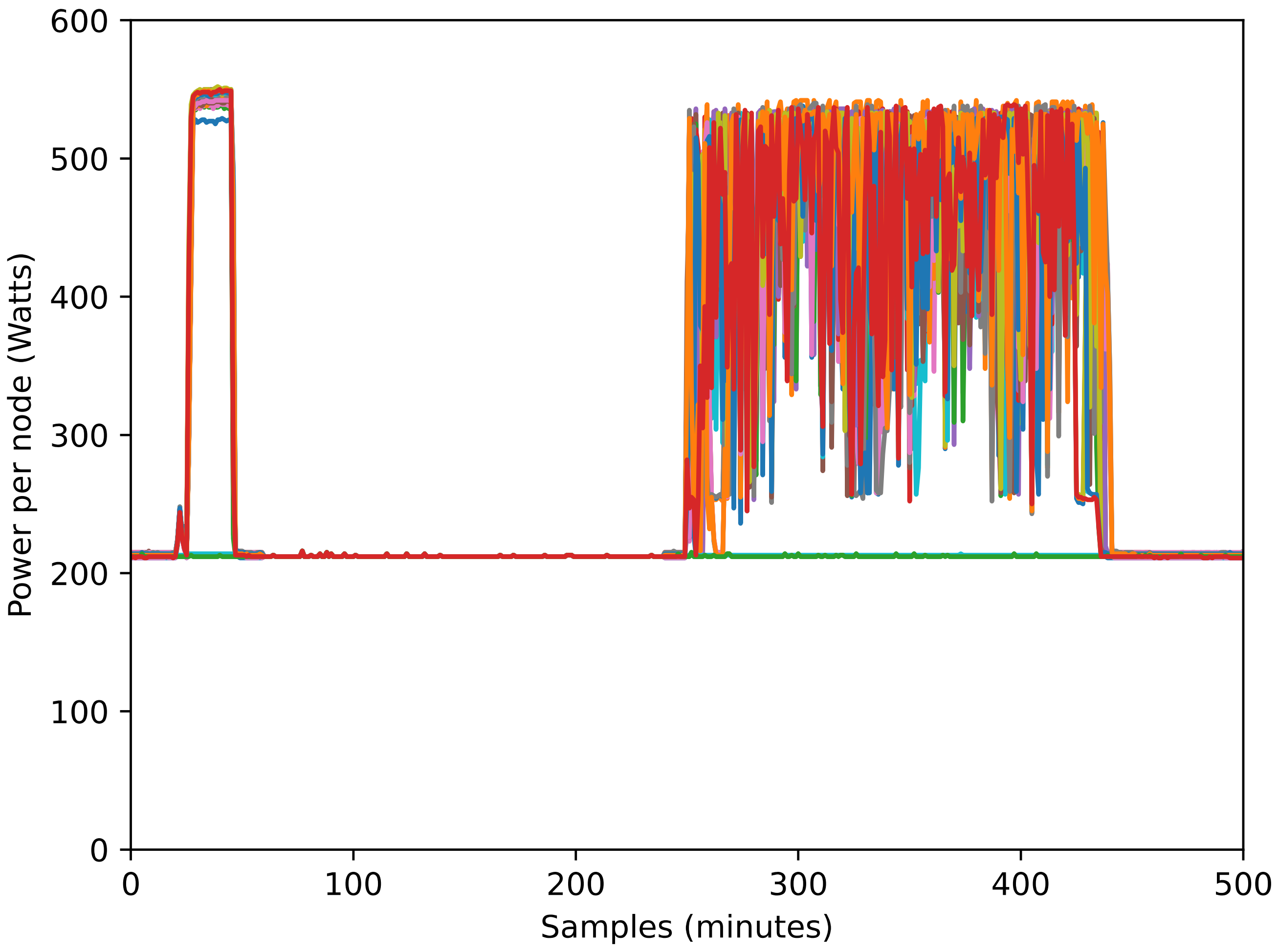


Space-filling curve (zcurve)



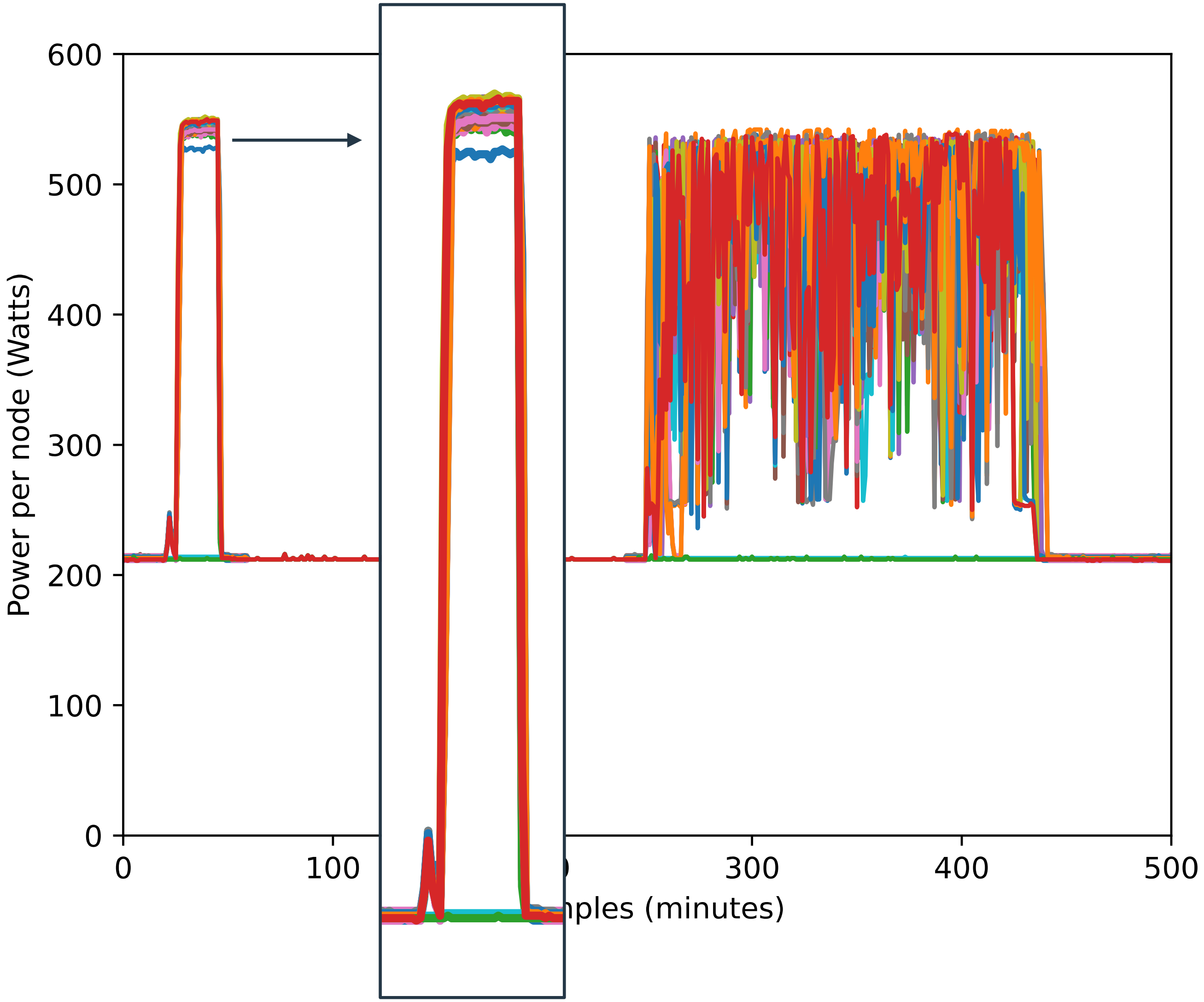
- RCM is much faster than zcurve
  - ~40s vs ~157s
  - Case dependent

Efficient software ≠ efficient use



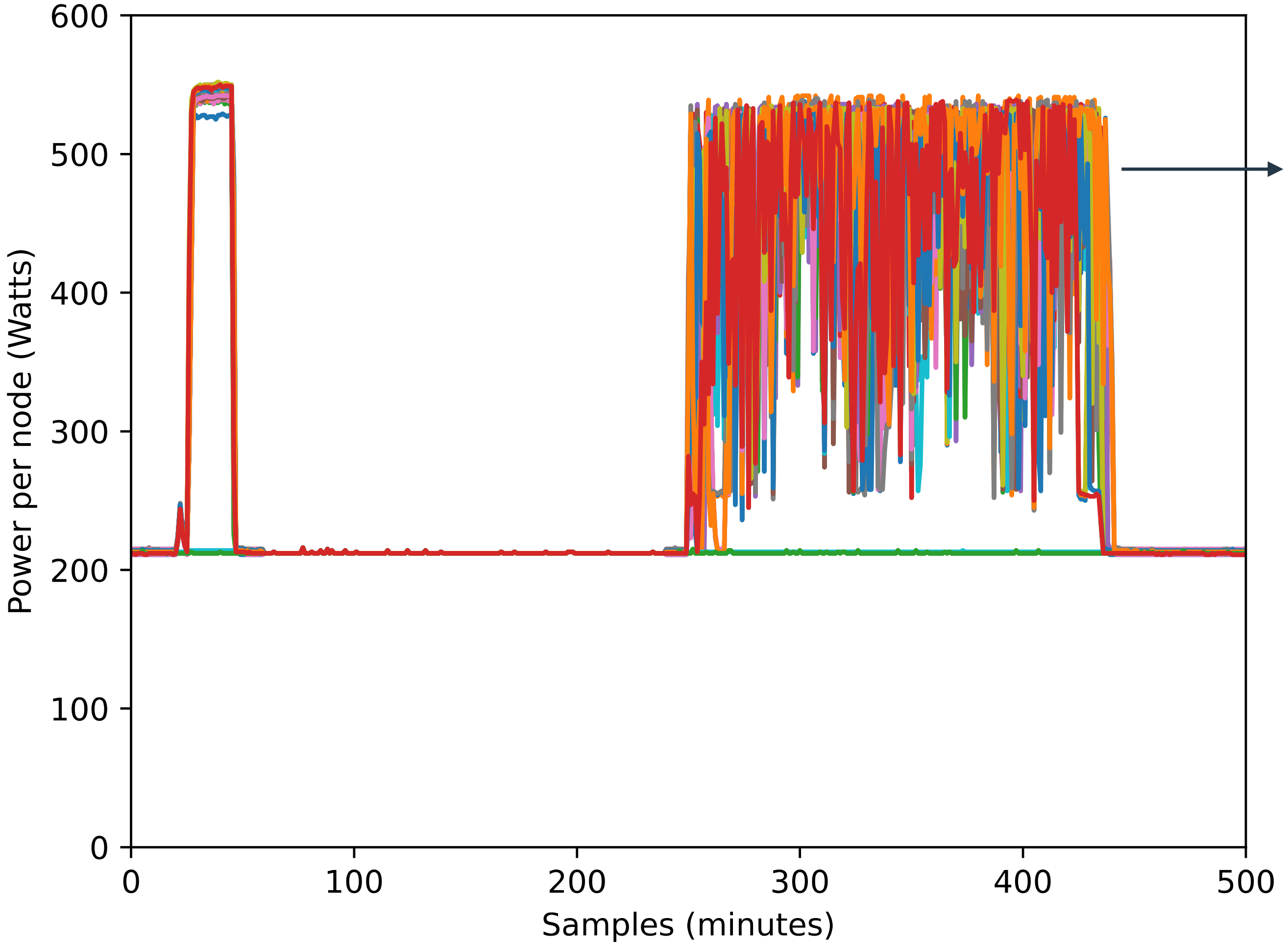
- Node-level power measurement
  - Each line represents power draw for 1 node
  - Full system, 34 nodes in total
  - Idle power draw: 213W
- Two identical aerodynamics simulations with OpenFOAM using 32 nodes
  - On the left: **no** I/O
  - On the right: **excessive** I/O

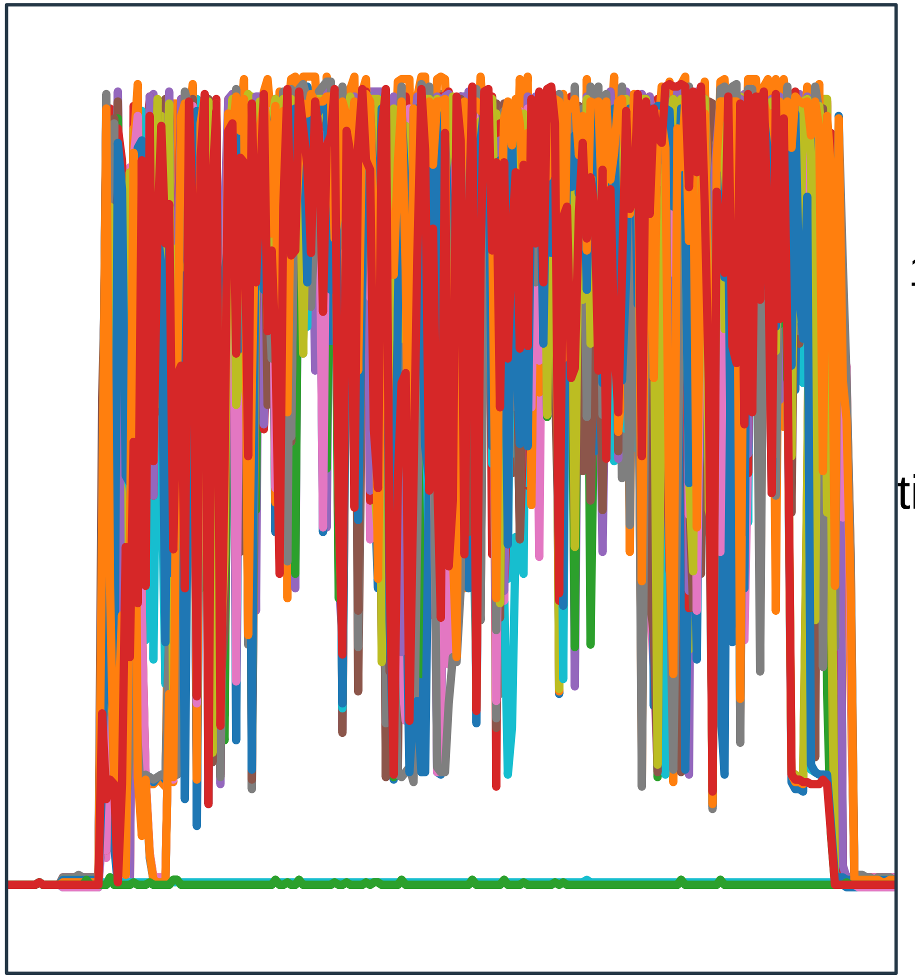
Efficient software ≠ efficient use



- Node-level power measurement
  - Each line represents power draw for 1 node
  - Full system, 34 nodes in total
  - Idle power draw: 213W
- Two identical aerodynamics simulations with OpenFOAM using 32 nodes
  - On the left: **no** I/O
  - On the right: **excessive** I/O

Efficient software ≠ efficient use

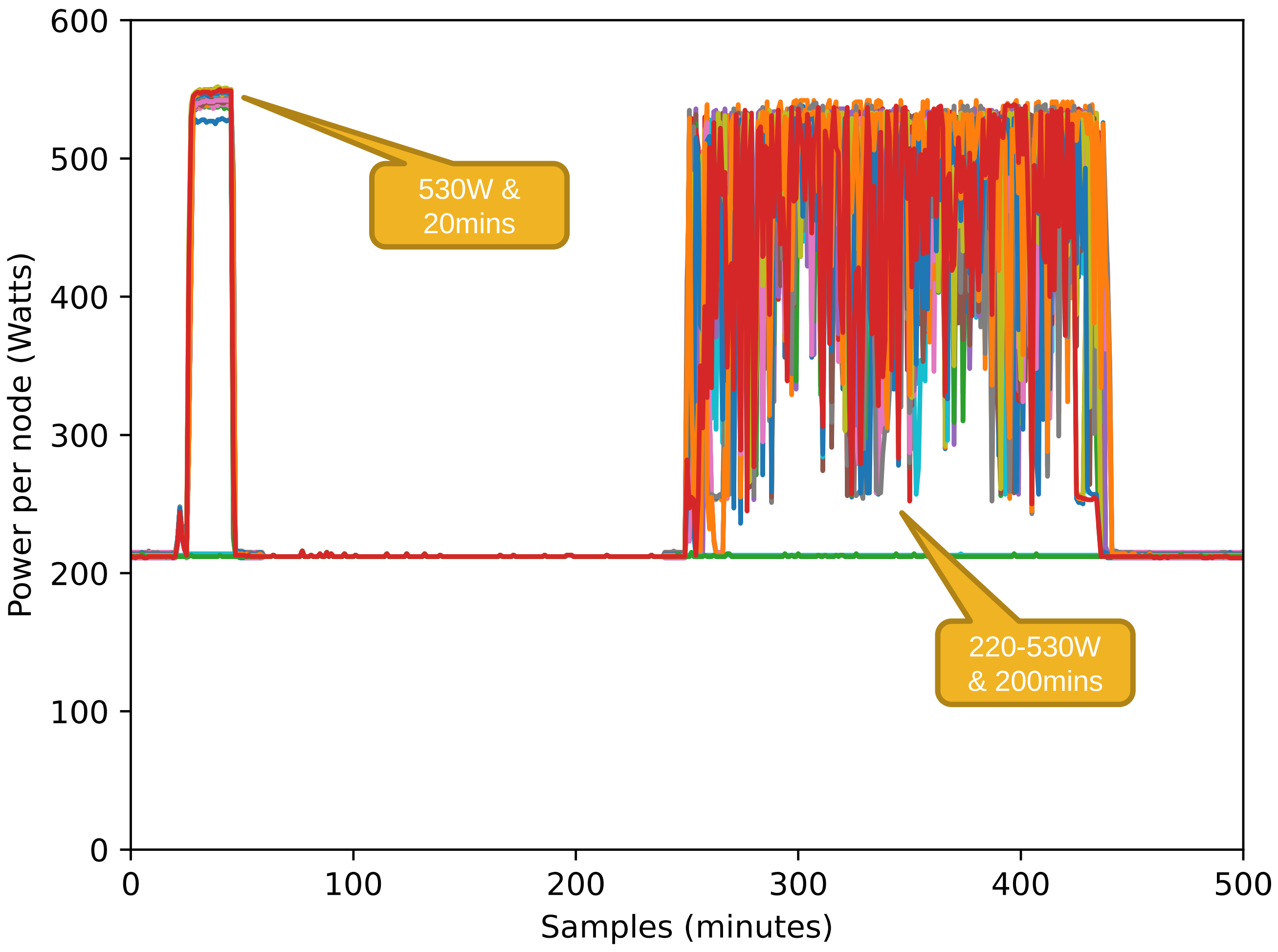




1 node

tions

Efficient software ≠ efficient use



- Node-level power measurement
  - Each line represents power draw for 1 node
  - Full system, 34 nodes in total
  - Idle power draw: 213W
- Two identical aerodynamics simulations with OpenFOAM using 32 nodes
  - On the left: **no** I/O
  - On the right: **excessive** I/O
- Excessive I/O means network contention & frequent stalling

Even highly efficient software can be misused to be extremely inefficient

epcc

Green software engineering - dos and don'ts

- **Do** capture requirements & write software that serves its intended purpose
- **Do** use CI systems and rigorous testing
- **Do** ensure users understand how to use your software correctly
- **Do** profile performance, find hotspots and fix them
- **Do** consider if algorithms are appropriate
- **Do** choose programming models based on performance, usability and maintainability
- **Do** design your code to be modular

- **Don't** jump on band wagons without justification
- **Don't** be afraid to test new/different techniques
- **Don't** forgo testing in favour of speed of development
- **Don't** forgo testing at scale because it uses compute cycles
- **Don't** believe software development for HPC is not a specialist skill
- **Don't** blindly use code generated by ChatGPT

epcc

Green software engineering - dos and don'ts

- **Do** capture requirements & write software that serves its intended purpose
- **Do** use CI systems and rigorous testing
- **Do** ensure users understand how t… your software correctly
- **Do** profile perf… them
- **Do** consider if al…
- **Do** choose progra… …odels based on performance, usa…y and maintainability
- **Do** design your code to be modular

- **Don't** jump on band w… …ns without justification
- … different
- … of speed of
- …orgo testing at scale because it uses compute cycles
- **Don't** believe software development for HPC is not a specialist skill
- **Don't** blindly use code generated by ChatGPT

None of this differs from good software engineering practice in general!

Final thoughts

Green software engineering is mostly just ***good*** software engineering

- Efficient, well written software that serves a purpose is inherently “green”
  - Survey of widely used applications?
- Education is key – targeting developers and users alike

HPC systems are **scientific instruments** that are used to find solutions to many of the problems humanity faces

→ to discover new vaccines

→ to design new renewable energy solutions

→ to model the climate, in order to more accurately predict climate change & its impact

Significantly reducing scientific throughput is a false economy

“Green” software engineering therefore **must** target **maximum throughput**!