User-level Power Monitoring and Application Performance on Cray XC30 Supercomputers

Alistair Hart, Harvey Richardson
(Cray Exascale Research Initiative Europe)

Jens Doleschal, Thomas Ilsche, Mario Bielert
(Technische Universität Dresden)

Matthew Kappel
(Cray Inc.)

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Contents

- Motivation and Introduction
- The hardware and the counters
- Running the NAS Parallel Benchmarks
- Score-P and Vampir
- Conclusions

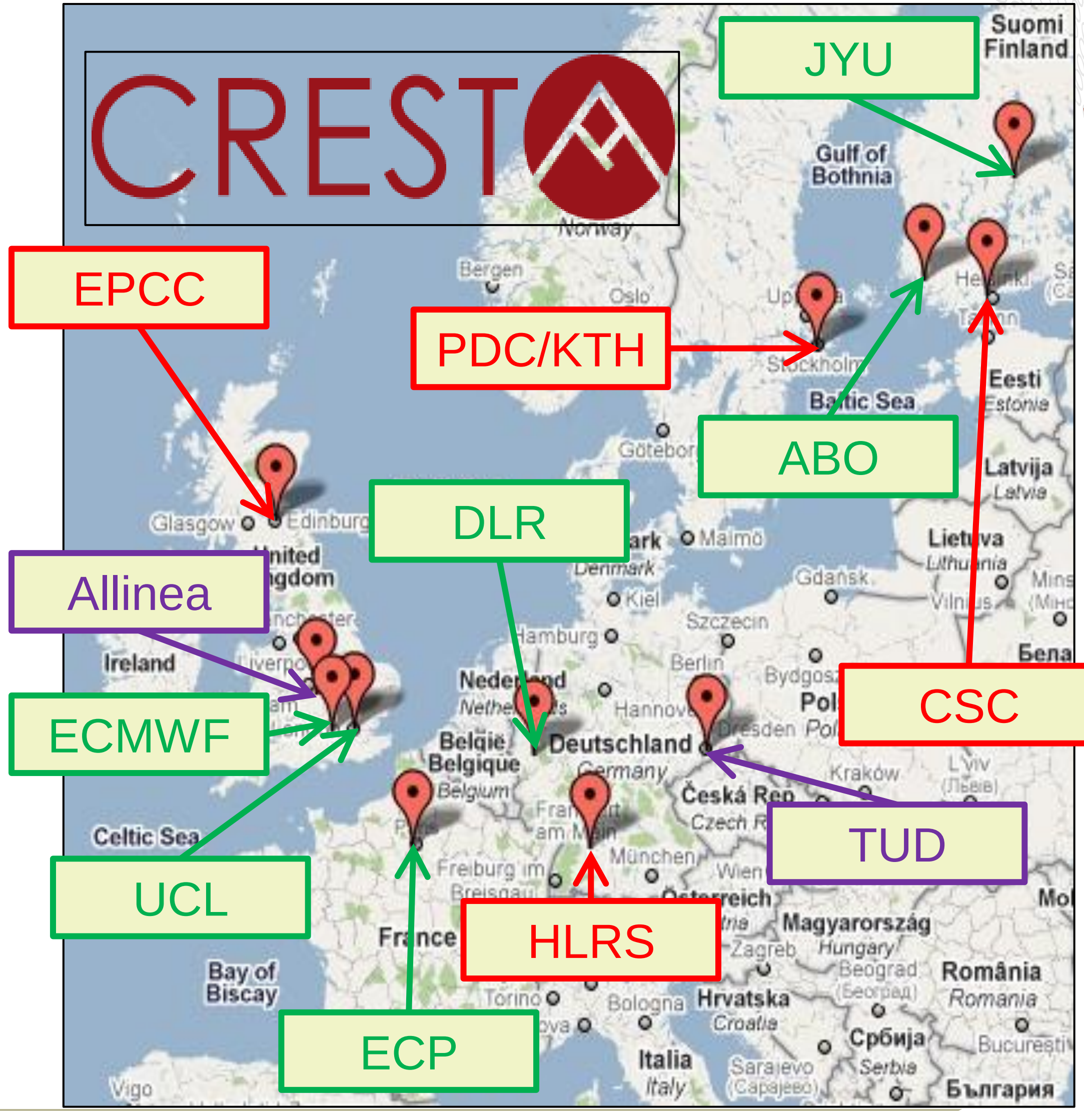
- **Related papers at CUG 2014:**
  - Fourestey, Cumming, Gilly, Schulthess:
    - *First Experiences with... the Cray Power Management Database Tool*
  - Martin, Kappel:
    - *Cray XC30 Power Monitoring and Management*
  - Poxon:
    - *New Functionality in the Cray Performance Analysis and Porting Tools*

Collaborative **R**esearch into **E**xascale **S**oftware, **T**ools and **A**pplications

- EU FP7-funded project
- Runs until Dec. 2014

- **Consortium includes**
  - Cray
  - EU HPC centres
  - Tools developers
  - Codesign app. owners

- **Codesign approach**
  - GROMACS, IFS,
  - Nek5000, HemeLB,
  - OpenFOAM, Elmfire



Reducing energy & power usage

- **A hot topic in HPC**
  - energy: reduce "carbon footprint"
  - power: balance demand
  - rise of the Green500 list
- **Importance will grow in the future**
  - US DOE: exascale system for 20MW
- **How can we get there?**
  - Hardware design will help
  - But software engineering will also be needed

Visibility of energy/power consumption

- **Feedback is needed to inform these decisions**
  - fine-grained information about how energy is consumed
    - fine-grained spatially (across components)
    - fine-grained temporally (across application execution phases)
- **Non-privileged, real-time access is needed**
  - where possible, information should be visible to users
    - not just to system administrators (e.g. on the SMW)
  - information should be accessible as the application executes
    - not just in post-processed log files

Cray XC30

- **The Cray XC30 offers a range of power-monitoring**
  - system-side:
    - via the PMDB power-monitoring database (on the SMW)
  - also user-side:
    - node-level
- **In this talk, we investigate the user-side options**

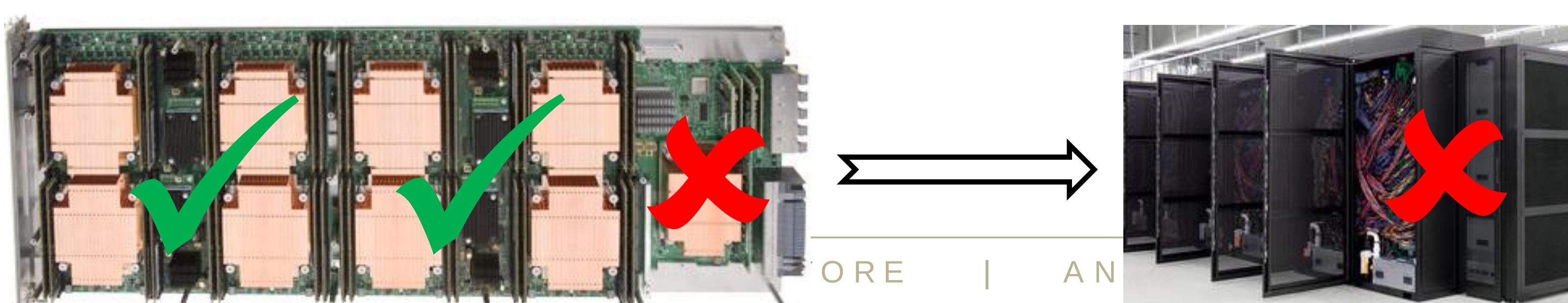
The hardware we studied

- **We look at two configurations of the Cray XC30**
- **"Marble", pure CPU:**
  - e.g. Archer, ECMWF systems in the UK
  - dual 12-core "Ivybridge" CPUs
    - Intel Xeon E5-2697, 2.7GHz

- **"Graphite", CPU/GPU hybrid:**
  - e.g. Piz Daint system at CSCS
  - single 8-core "Sandybridge" CPU
    - Intel Xeon E5-2670, 2.6GHz
  - plus Nvidia Kepler K20x GPU

Node-level power counters

- **The user-accessible power counters are node-specific**
  - One set of counters per node
  - Measure the power consumption of the node-specific components
  - This includes:
    - CPU, memory system, PCI bus, node-level hardware
  - Does not include shared components:
    - Aries interconnect (shared between 4 nodes on a blade)
    - Cabinet-level infrastructure (shared between 192 nodes)
- **So we do not get a full picture**
  - but it is harder to disentangle consumption by shared components



Power counters

- **The node-level energy/power counters are visible via a set of files in the /sys filesystem**

```
$ ls /sys/cray/pm_counters
power
energy
freshness
generation
power_cap
startup
version
```

- **power**
  - instantaneous power draw
    - in Watts
- **energy**
  - cumulative consumption
    - in Joules
    - relative to some time in the past
- **freshness**
  - label (integer-valued)
  - changes when counters update
    - every 100ms

```
$ cat /sys/cray/pm_counters/power
37 W
$ cat /sys/cray/pm_counters/energy
58062241 J
$ cat /sys/cray/pm_counters/freshness
12933225
```

Reading the counters

- **The counters update "atomically" (at the same time)**
- **But...**
  - if you read multiple counters sequentially...
  - ... you need to ensure that you have a consistent set of readings
  - the freshness counter helps ensure this

- **Procedure for reading multiple counters:**
  1. Read the freshness counter
  2. Read the relevant counters (power, energy, ...)
  3. Read the freshness counter again
  - If the initial and final freshness values are the same...
    - we have a consistent set of readings
  - ... otherwise:
    - repeat steps 1 to 3

Reading the power counters (API)

- **Can read counters from job script**
  - but we often want better temporal resolution than job level
- **In CRESTA, developed simple pm_lib library**
  - Reads a user-defined set of counters "consistently"
  - User can insert API calls in application
    - to instrument particular phases of the application execution
  - pm_lib can be called from Fortran or C/C++

```fortran
! Fortran
use pm_lib
type(pm_counter) :: counters(3)=[ &
    PM_COUNTER_FRESHNESS, &
    PM_COUNTER_ENERGY, &
    PM_COUNTER_ACCEL_ENERGY ]
integer(kind=i64) :: values(3)

call pm_init
nc=pm_get_counters(3,counters,values,1)
```

```c
// C/C++
#include "pm_lib.h"
pm_counter_e counters[3]={ \
   PM_COUNTER_FRESHNESS, \
   PM_COUNTER_ENERGY, \
   PM_COUNTER_ACCEL_ENERGY };
pm_counter_value values[3];

pm_init();
nc=pm_get_counters(3,counters,values,1);
```

Changing p-states

- **You can change the p-state without rebooting**
  - Linux OS (e.g. CLE) provides CPUFreq governor

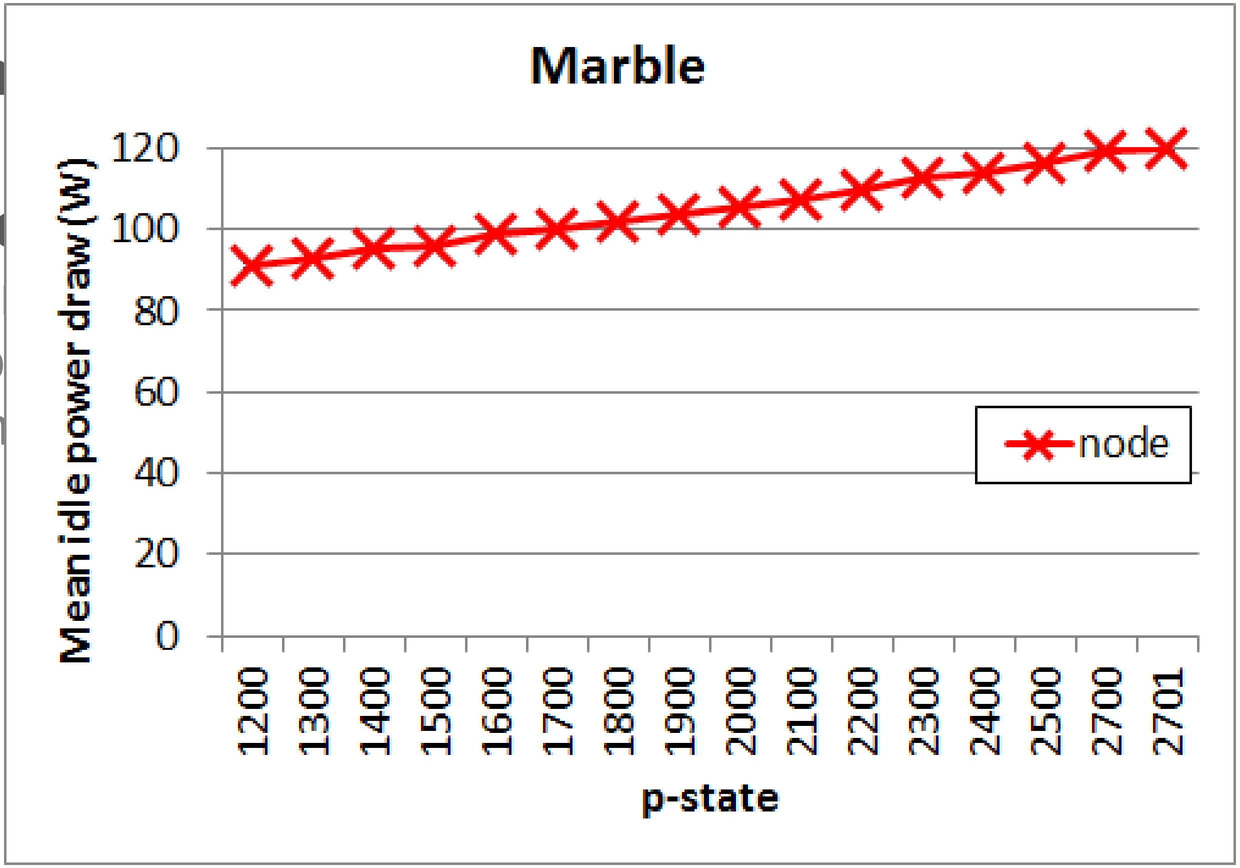
- **Default Cray XC30 setting:**
  - "performance", fixed to highest p-state

- **Instead can fix frequency to specified value**

- **Cray ALPS provides way to do this**
  - at application launch
  - `aprun --p-state <value>`

Idle power draw

- **How much of the power draw is due to the application?**
- **We need a baselin**
- **Simple to measur**
  - Execute a sleep co
  - Measure energy co
  - Calculate the mean



NAS Parallel Benchmarks

- **NPB provides a suite of parallel benchmarks**
  - each replicates single phase of a representative CFD application

- **Results reported here use 4 nodes**
  - CLASS=C is appropriate problem size

- **Compilation**
  - Used Cray Compilation Environment (CCE) version 8.2
  - Default optimisation -O2 used, unless otherwise specified

Questions we seek to answer

- Is optimising for energy consumption different to runtime?
- Is hyperthreading more energy-efficient?
- Is it "green" to optimise your code?
- Is a hybrid MPI/OpenMP code more energy-efficient?
- Do GPUs save energy?

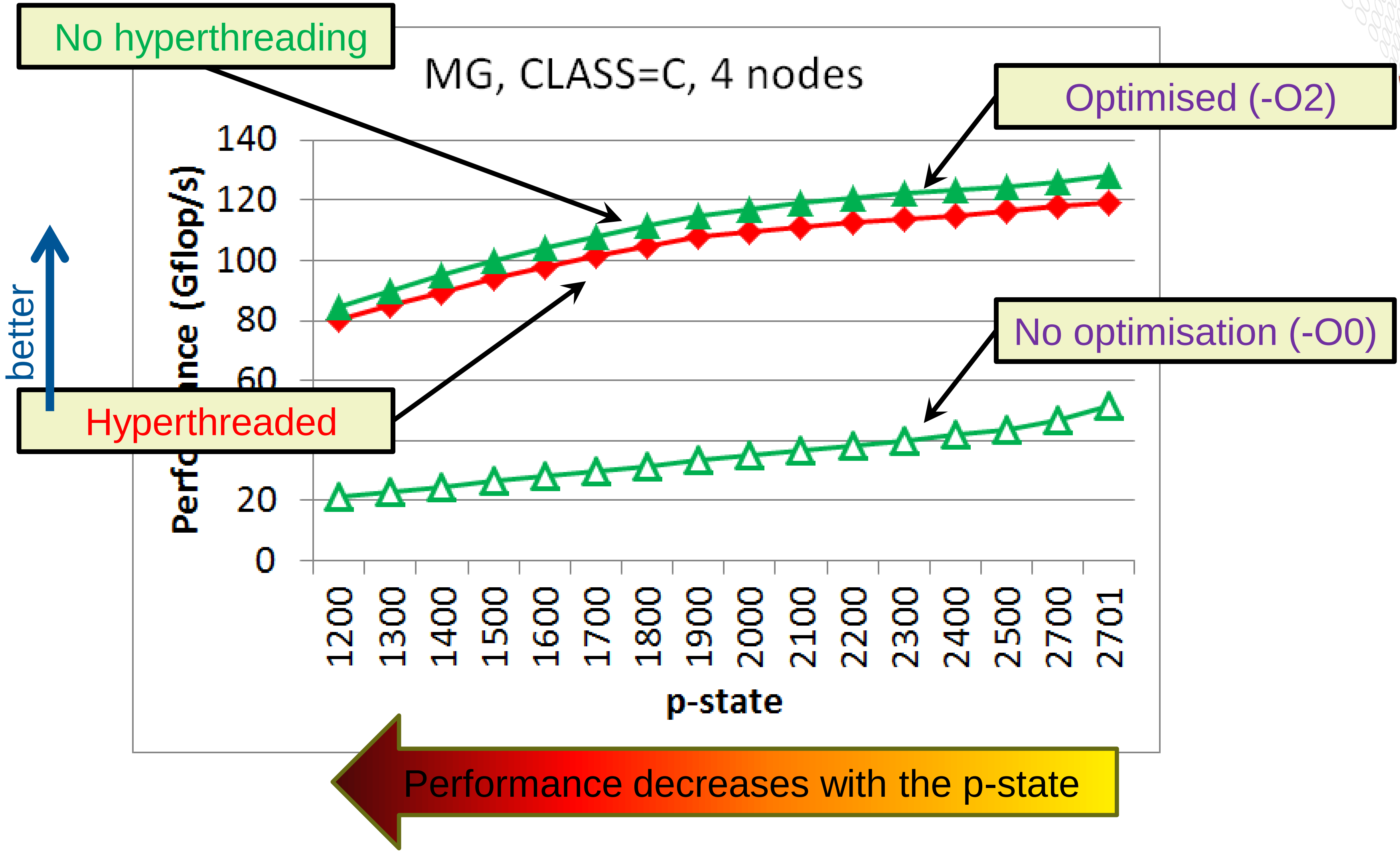
Performance metrics

- **Runtime performance**
  - flop/s
- **Total energy consumed during the calculation**
  - Joules

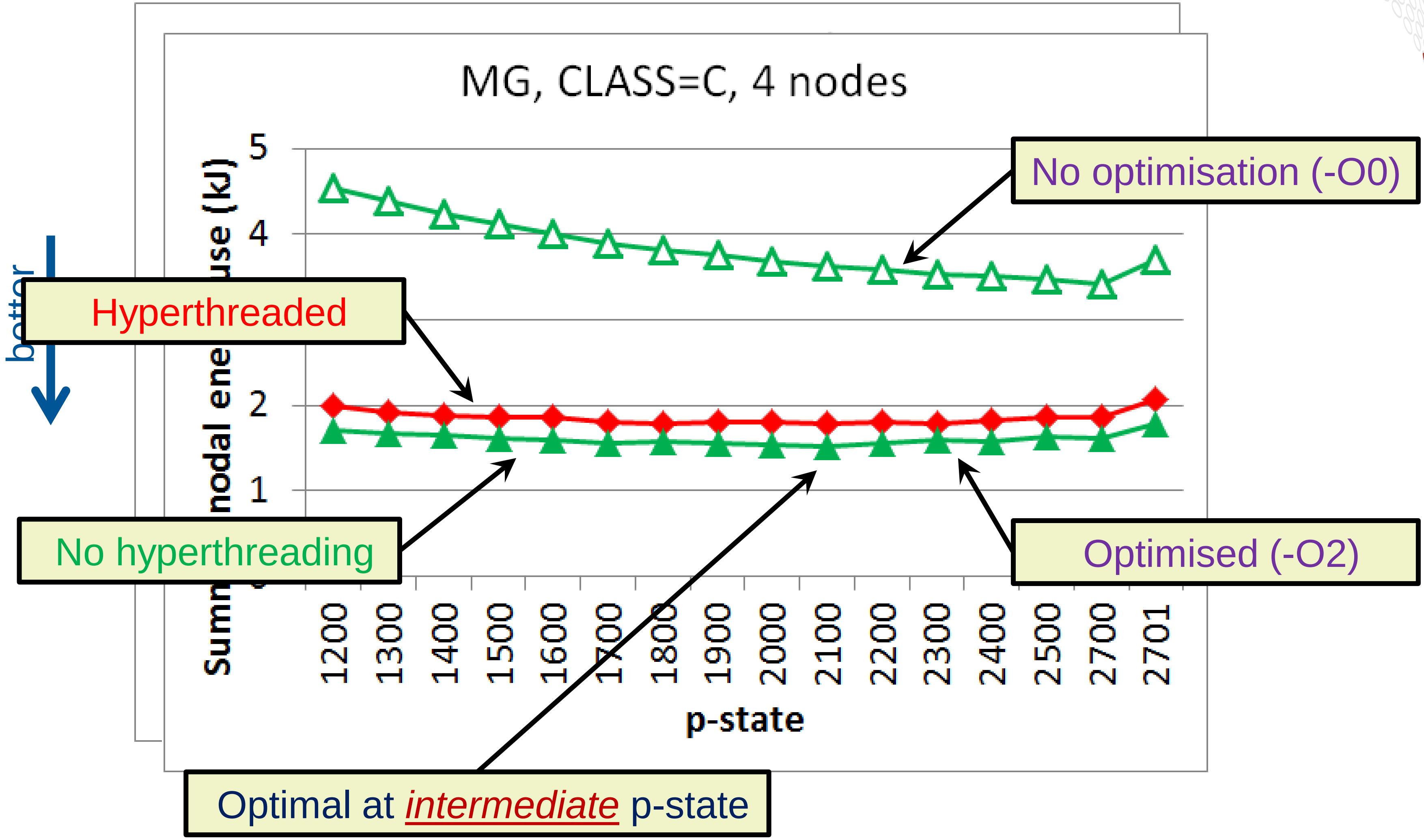
- **Mean node power draw during calculation**
  - Watts

- **Performance:power ratio**
  - flop/s per Watt (or flop/J)

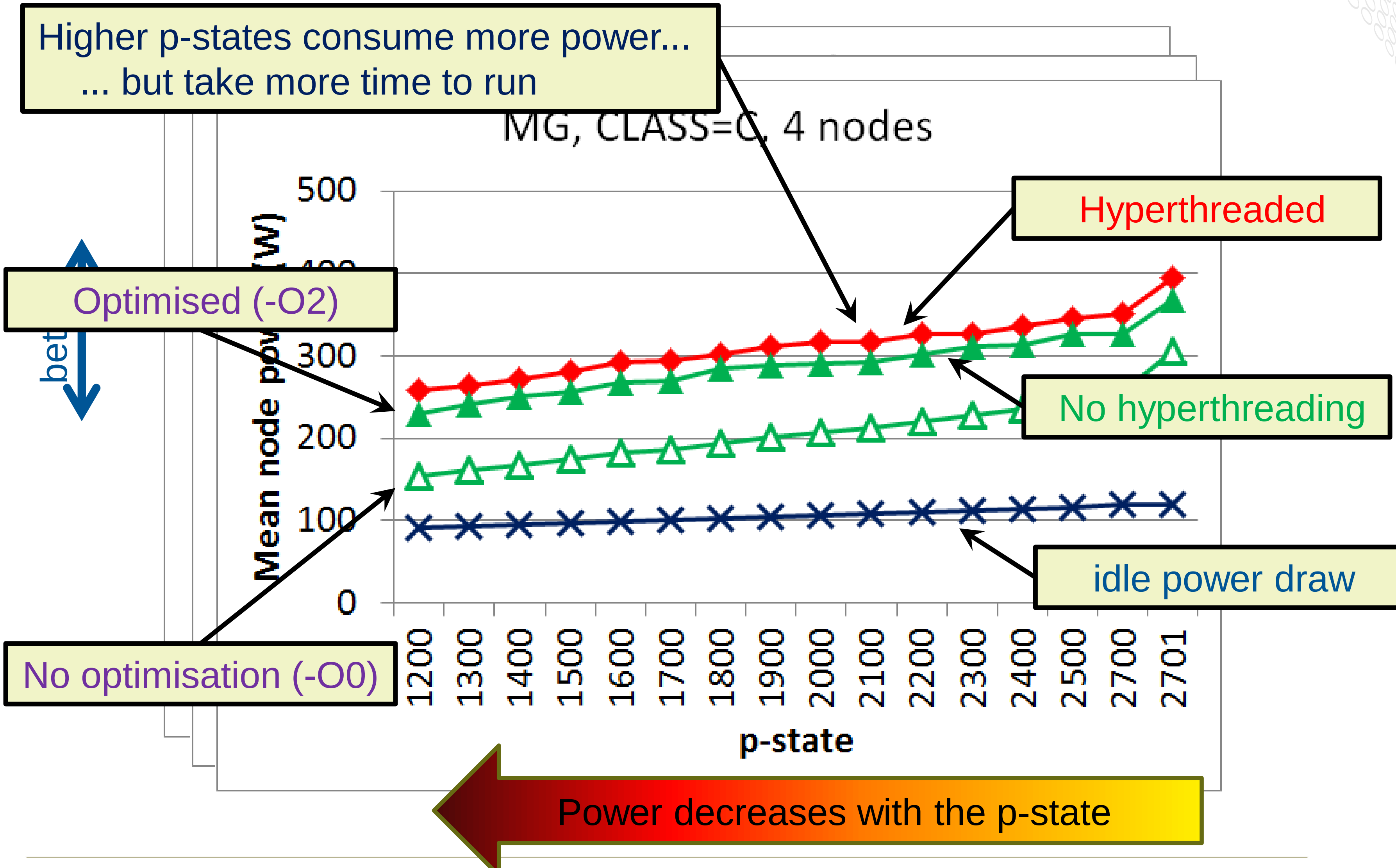
MG on Marble: runtime performance



MG on Marble: energy consumption



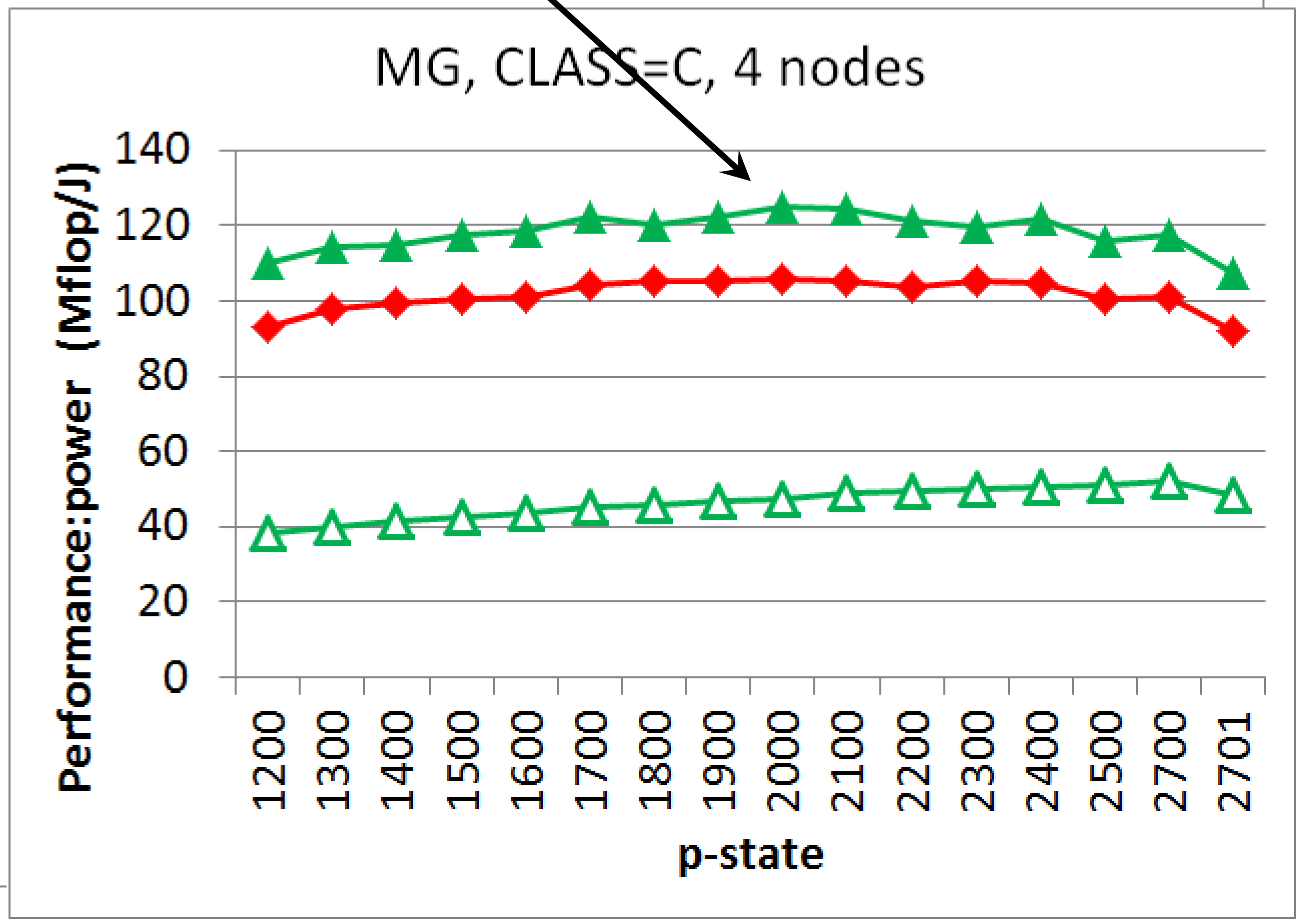
MG on Marble: mean power consumption



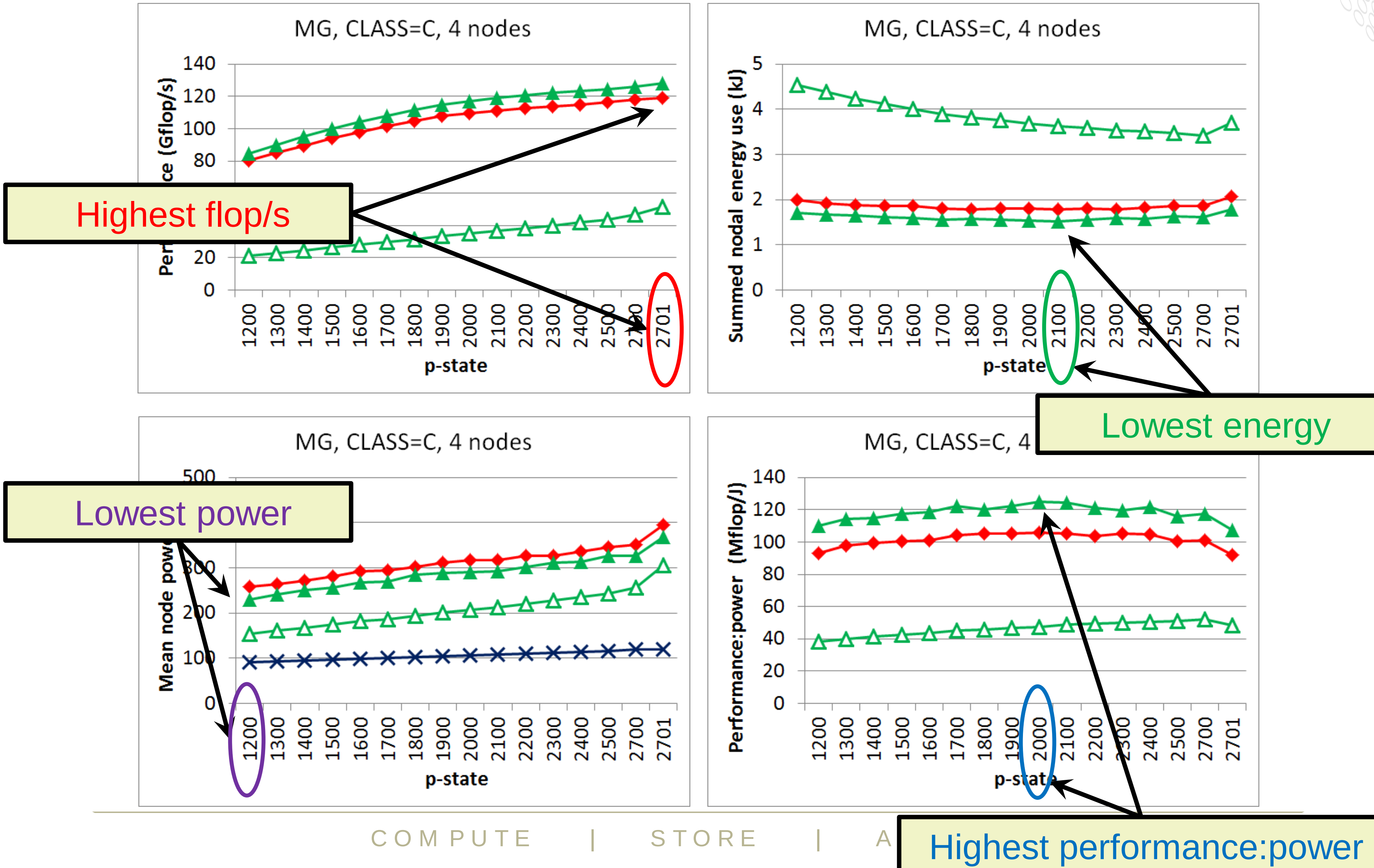
MG on Marble: performance:power ratio

Optimal at *intermediate* p-state

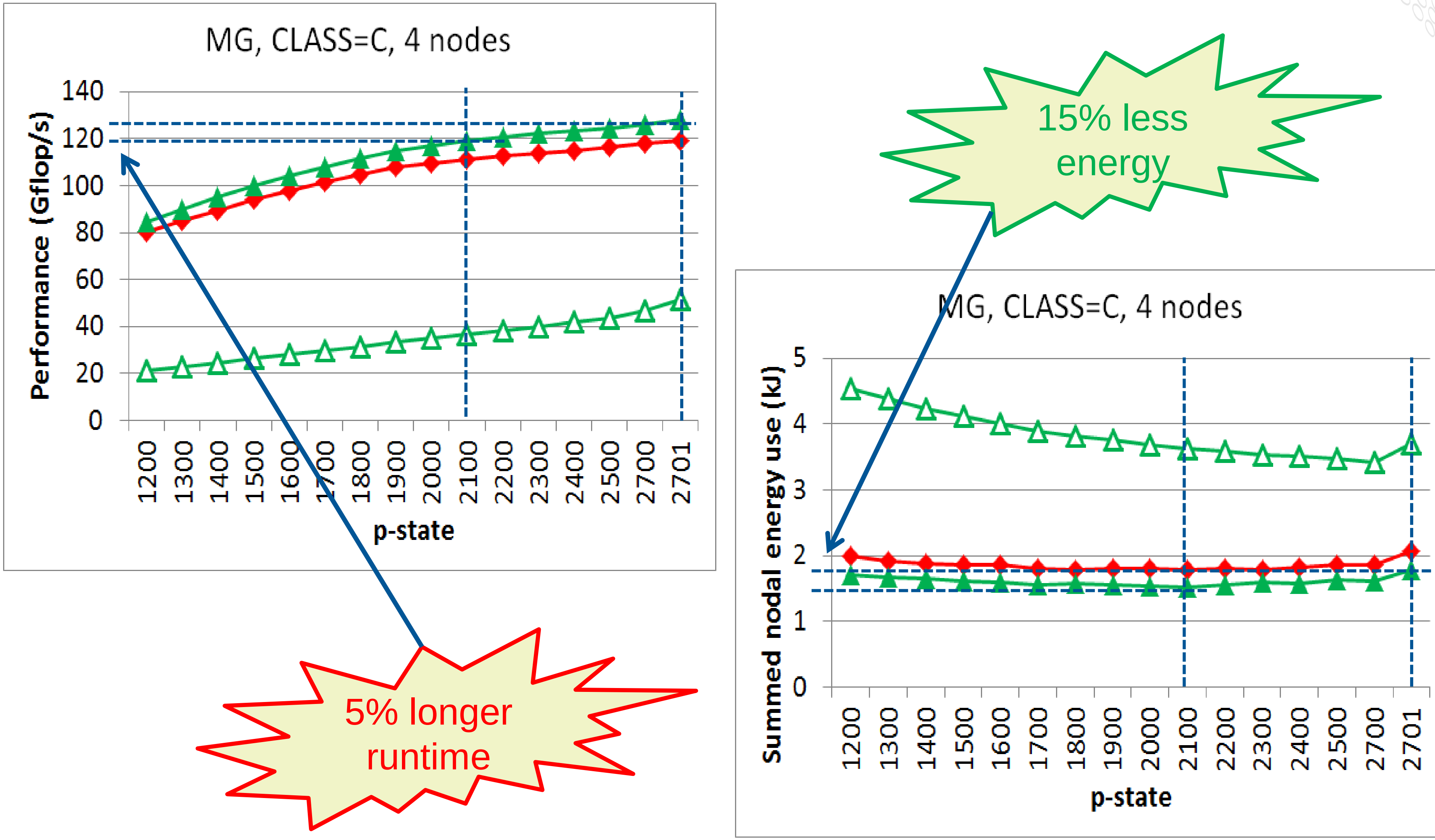
better



MG on Marble: summary



MG on Marble: performance vs. energy



Other benchmarks

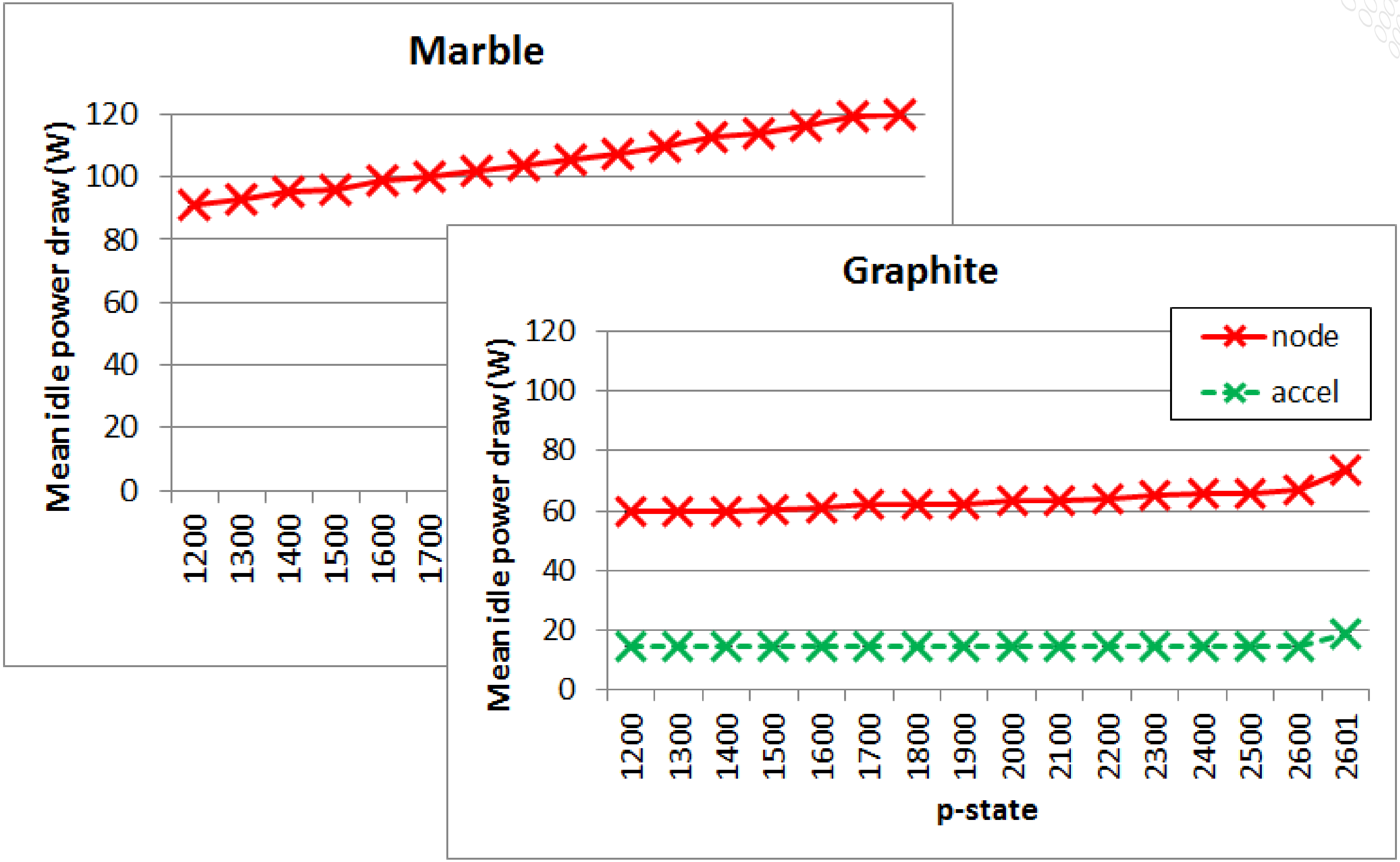
- **MG is representative of "computationally heavy" BMs**
  - CG and FT behave similarly
- **EP and IS show a different pattern of behaviour**
- **Same patterns seen using CPUs of Graphite nodes**

Accelerated nodes

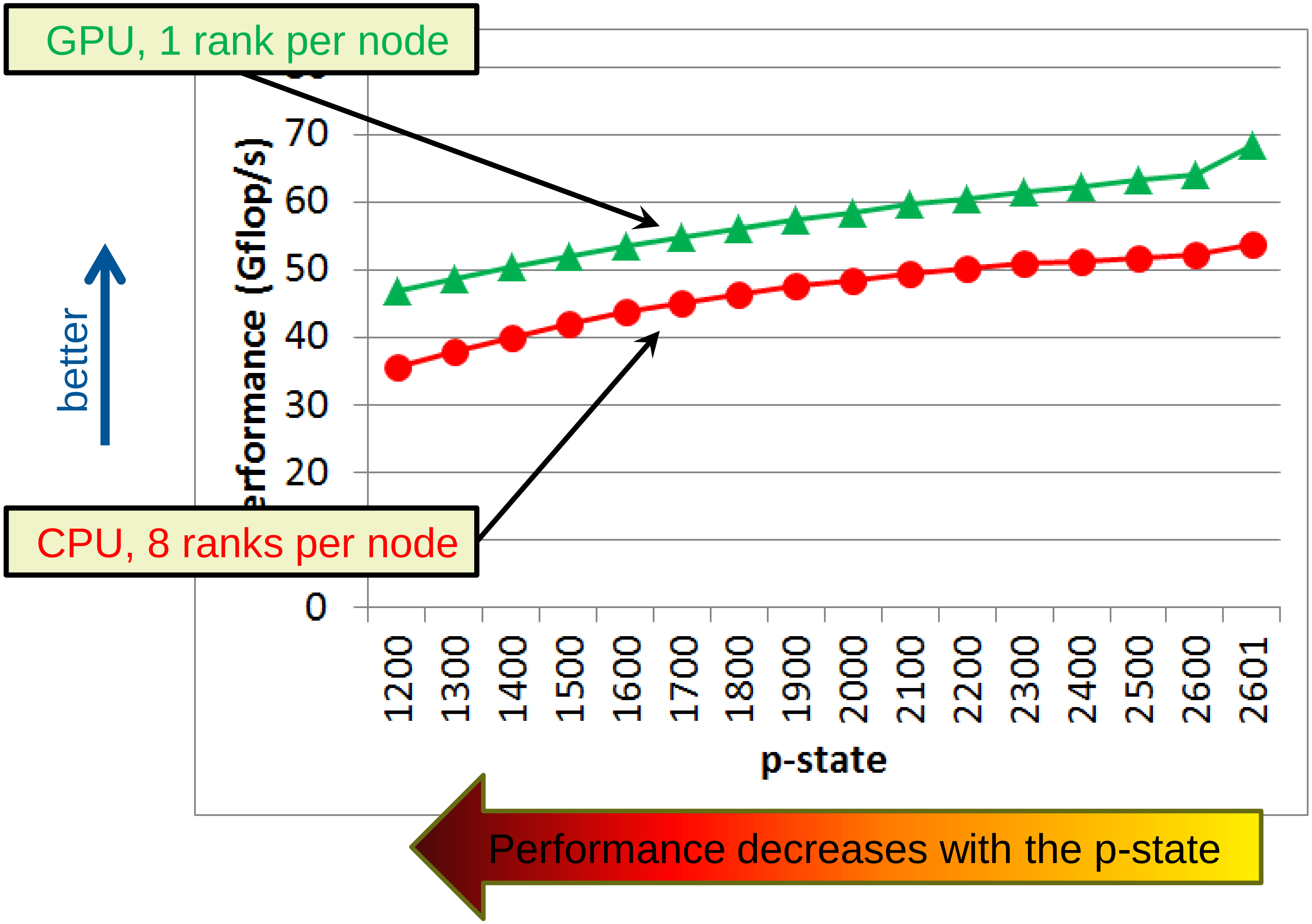
- **Previous counters continue to measure entire node**
  - including the accelerator (Nvidia GPU or Intel Xeon Phi)
  - power, energy, freshness
- **Two new counters available**
  - measure just the accelerator
- **accel_power**
  - instantaneous power draw (in Watts)
- **accel_energy**
  - cumulative consumption (in Joules)

```
$ ls /sys/cray/pm_counters
power
accel_power
energy
accel_energy
freshness
generation
power_cap
startup
version
```

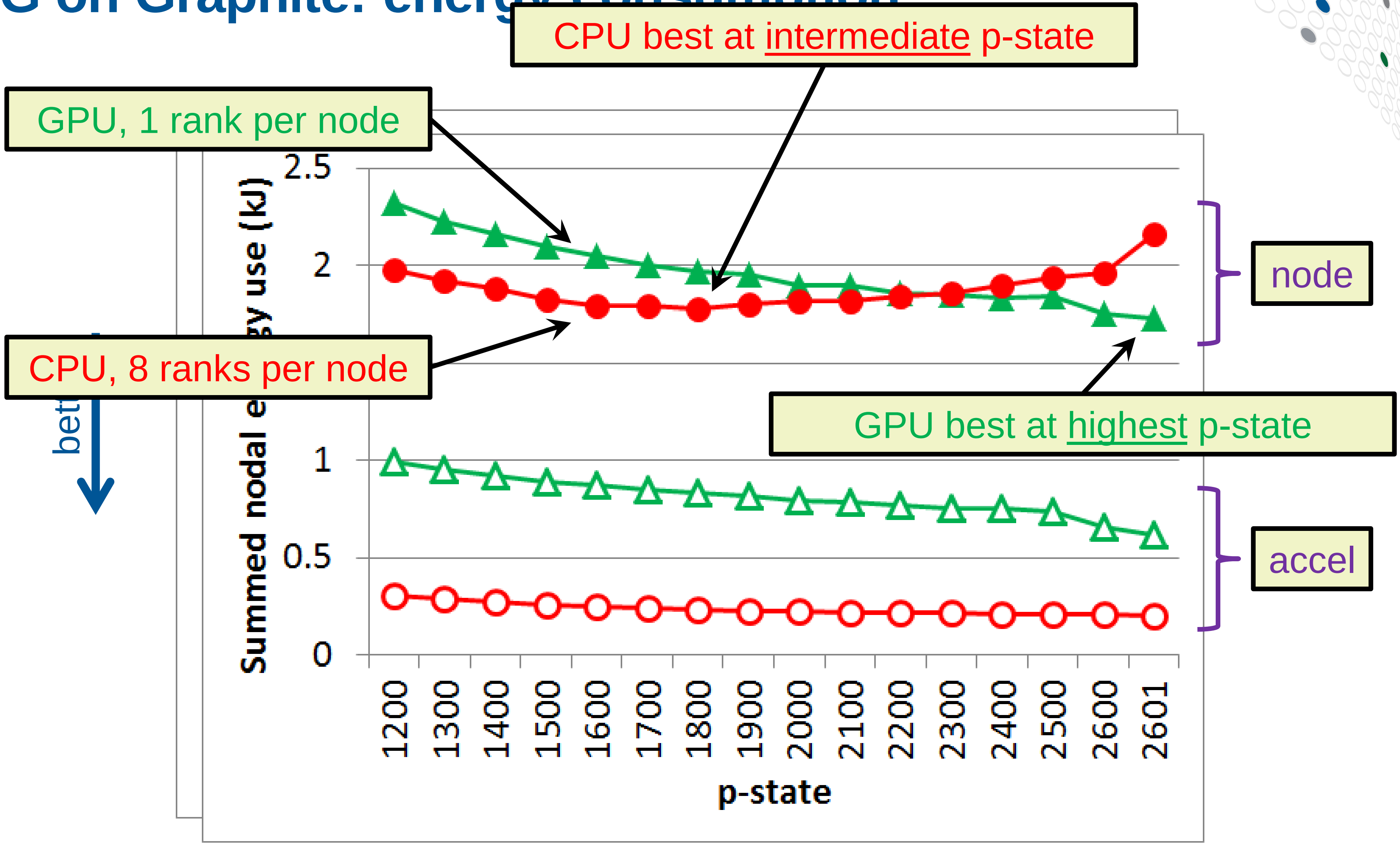
Idle power draw



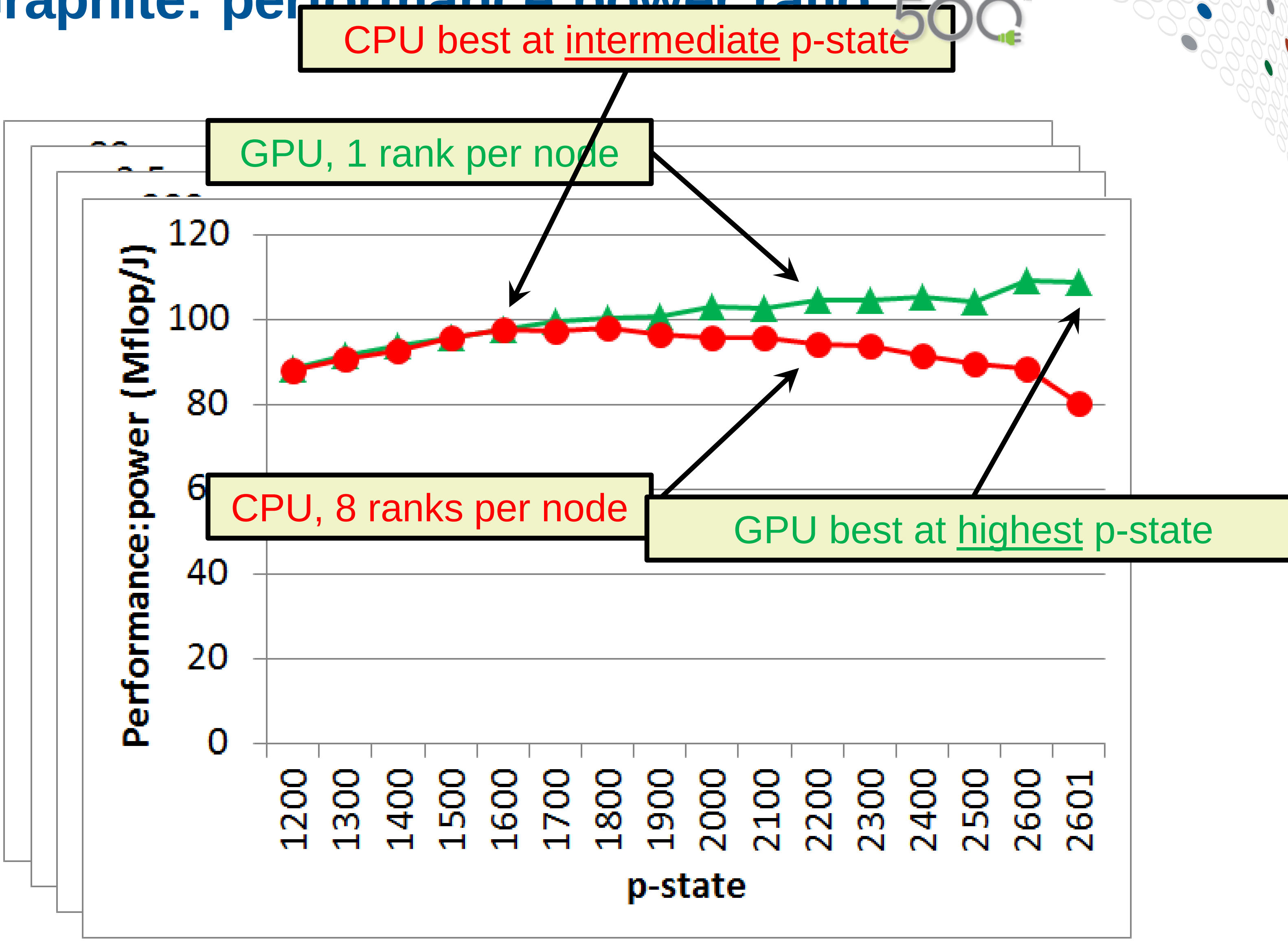
MG on Graphite: runtime performance



MG on Graphite: energy consumption



MG on Graphite: performance:power ratio



GPU and CPU

- **We would like to use both GPU and CPU**
  - both have considerable computational power
  - both have a base power consumption

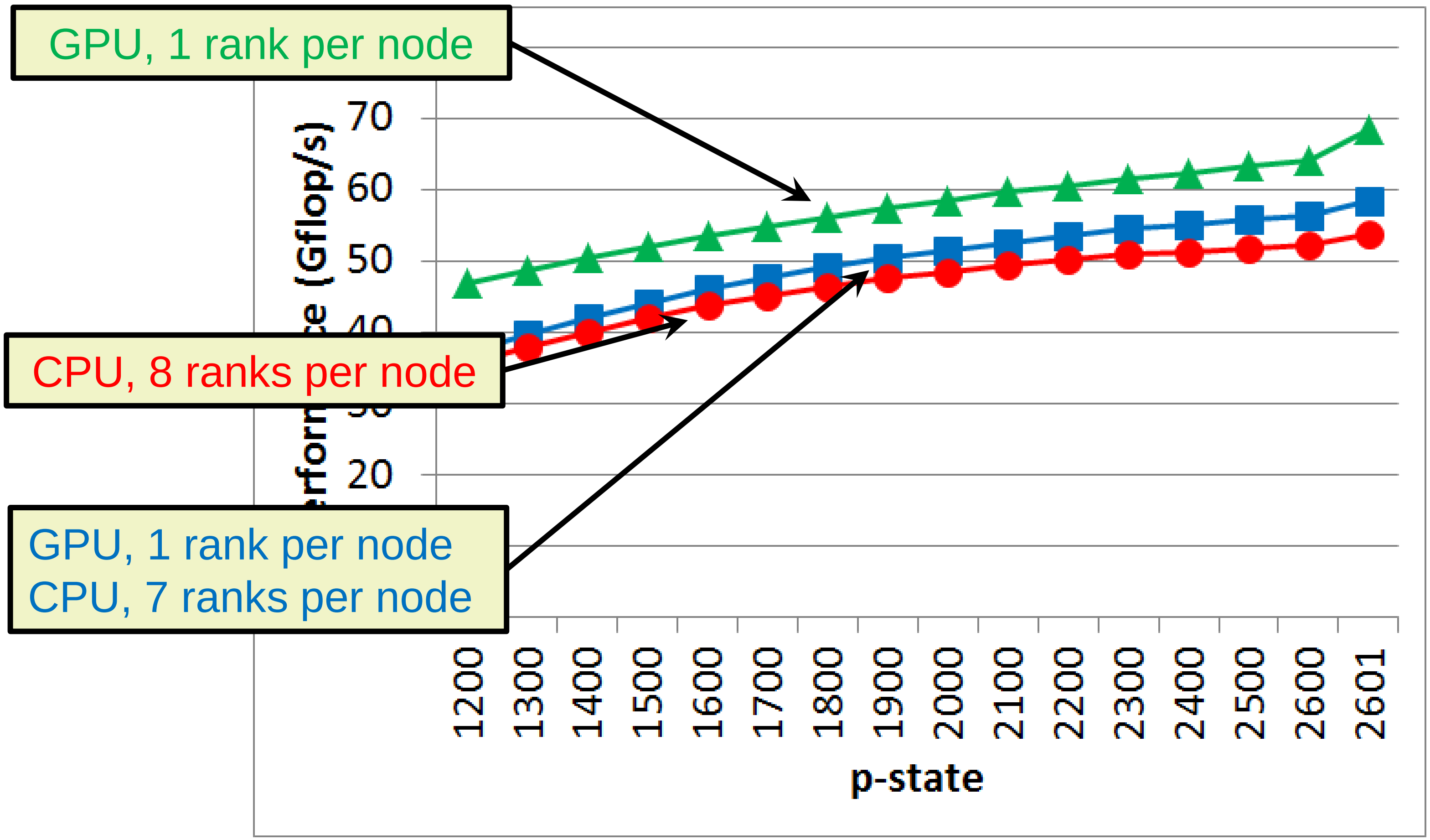
- **We would like to do this transparently**
  - rather than having to recode the application

- **Cray ALPS provides a way to do this**
  - Launch multiple binaries on the same node
  - Some targetting the GPU, some targetting the CPU
  - Sharing MPI_COMM_WORLD
  - Using new `PMI_NO_FORK` environment variable

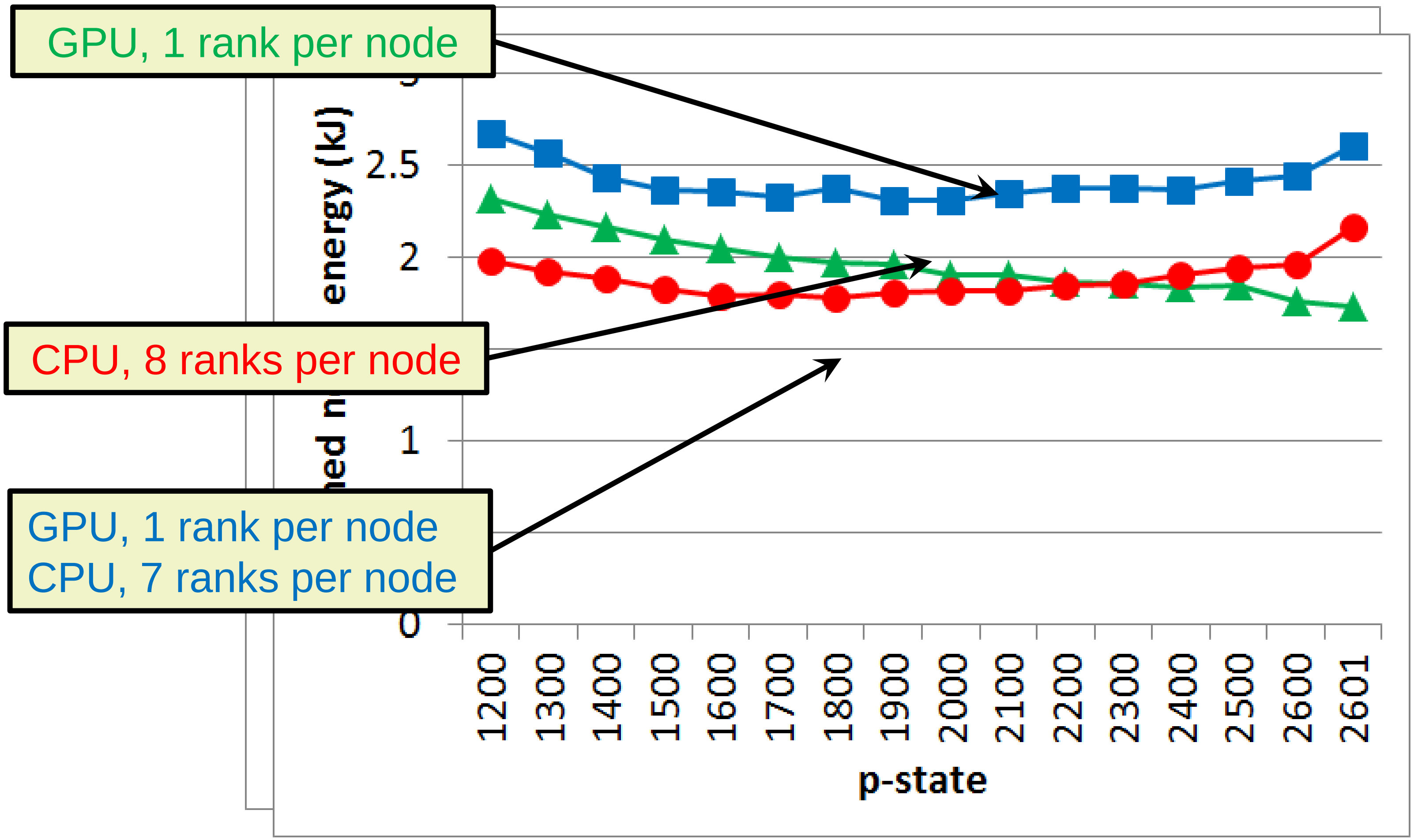
What we do

- **Compile the application twice using 32 ranks**
  - Once to run on the CPU
  - Once to run on the GPU, using OpenACC
- **Run the application as follows**
  - 4 ranks (1 per node) will use the GPU
  - 28 ranks (7 per node) will use the CPU
- **All ranks share the same MPI_COMM_WORLD**
- **Each rank computes the same-sized local problem**

CPU/GPU MG: runtime performance



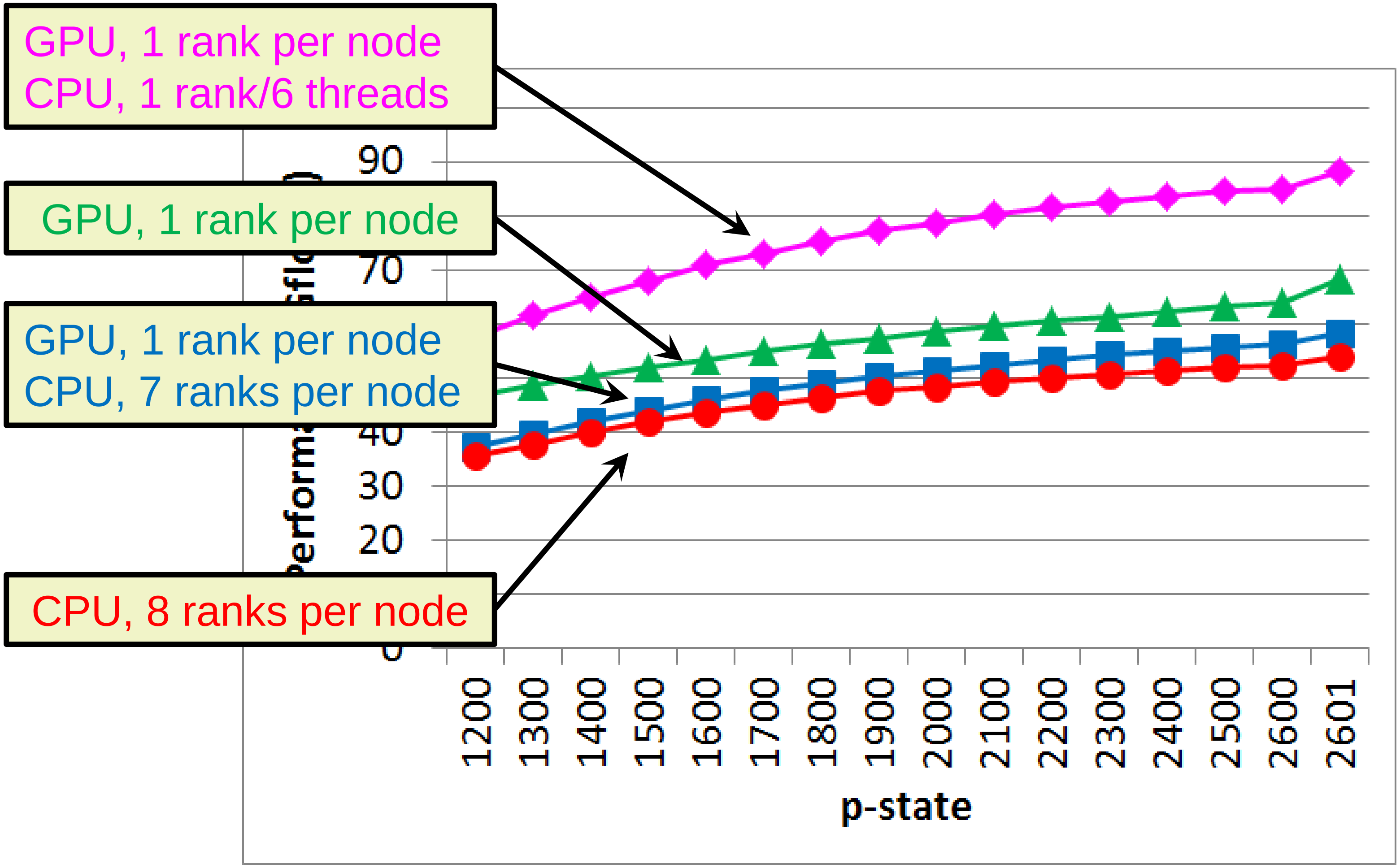
CPU/GPU MG: energy consumption



Correcting load balance

- **Load balance is the main problem**
  - Each rank processes the same amount of data
    - GPU ranks have a whole accelerator
    - CPU ranks have just a single core
- **Need to improve computing power/rank on the CPU side**
- **Introduce OpenMP threading on the CPU**

hybrid MG: runtime performance



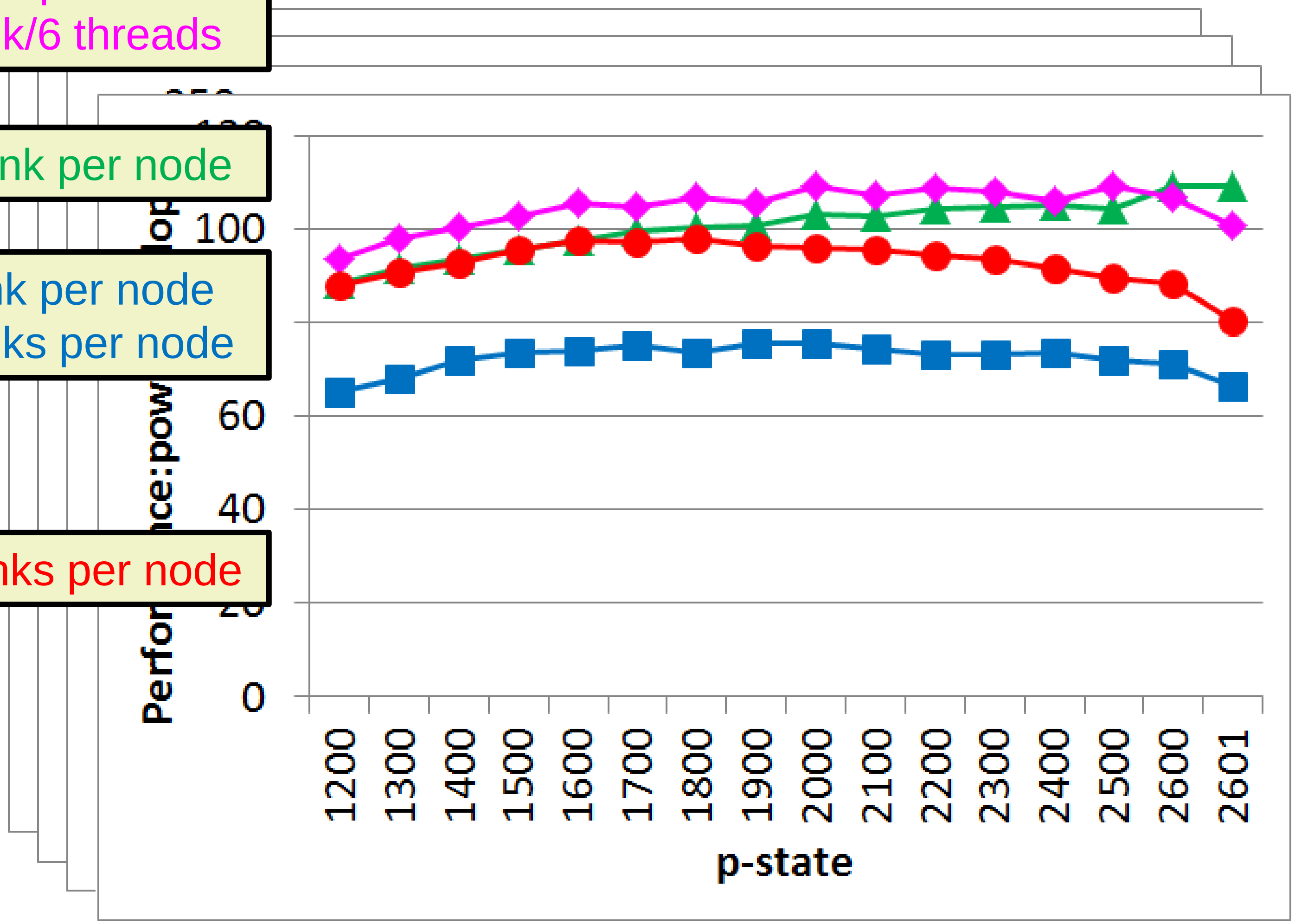
hybrid MG: performance:power ratio

GPU, 1 rank per node
CPU, 1 rank/6 threads

GPU, 1 rank per node

GPU, 1 rank per node
CPU, 7 ranks per node

CPU, 8 ranks per node

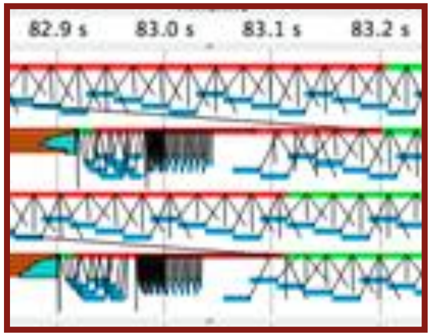


Performance Measurement and Visualisation Infrastructure

Score-P

- performance measurement infrastructure
- enables profiling and tracing of parallel applications
- scalable to high core counts
- Interface for new metric plugins
  - to record external, generic, hierarchical performance counters

Vampir



- Performance visualizer
- displays the behaviour of the application over time
- colour-coded timelines and other special displays

Adding performance measurement

Score-P

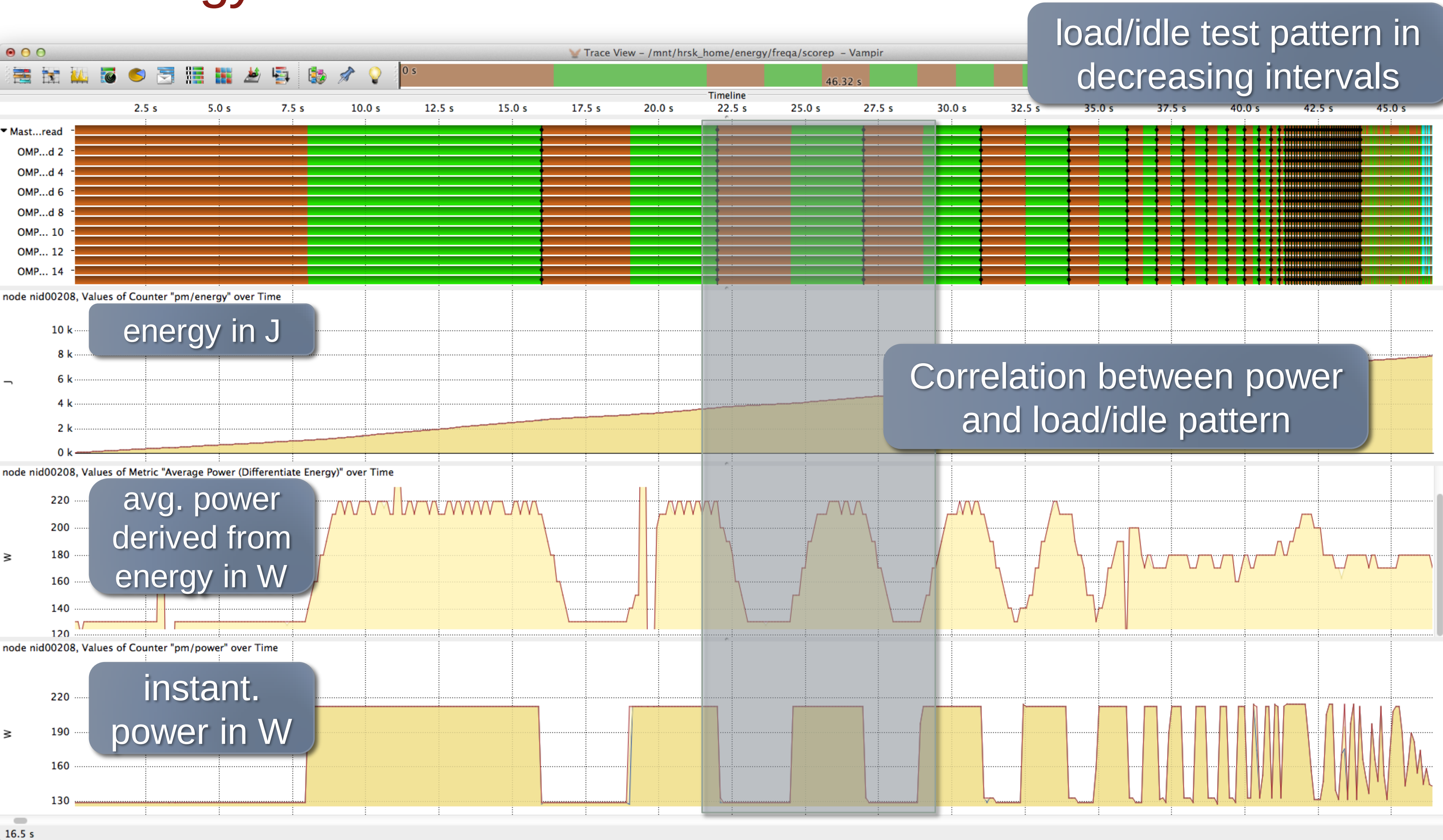
- Built a new PM-Power metric plugin
  - records Cray XC30 energy and power counters at application runtime
  - Using user-readable counters (/sys/cray/pm_counters)
- Counters collected asynchronously to the application events
  - measured at predefined frequency (~100Hz)
  - stored as timestamp-value pair
- At the end of the measurement
  - Timestamp-value pairs linked with application monitoring information

Vampir

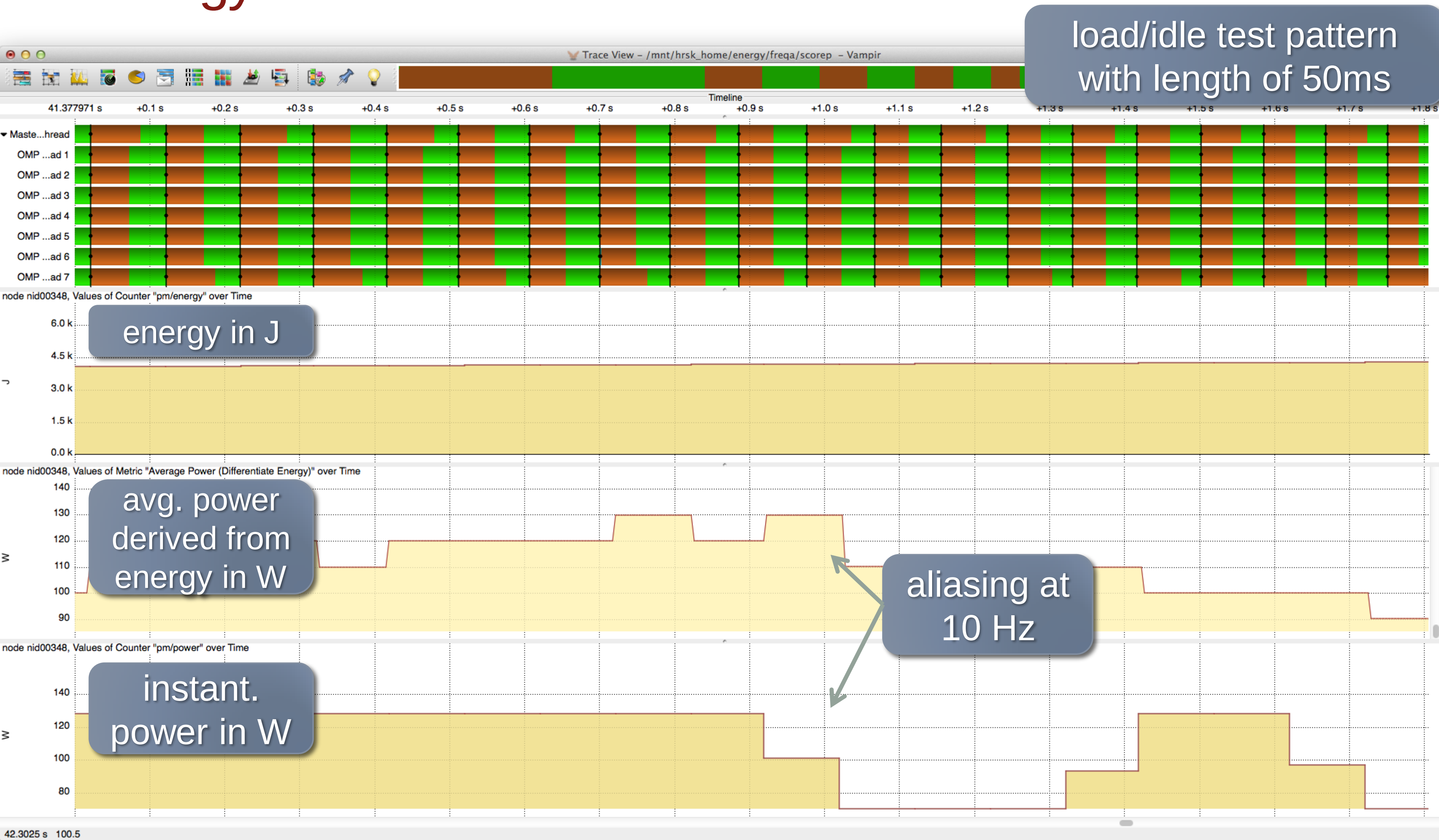
- Modified to display power, energy using colour-coded timelines

CREST

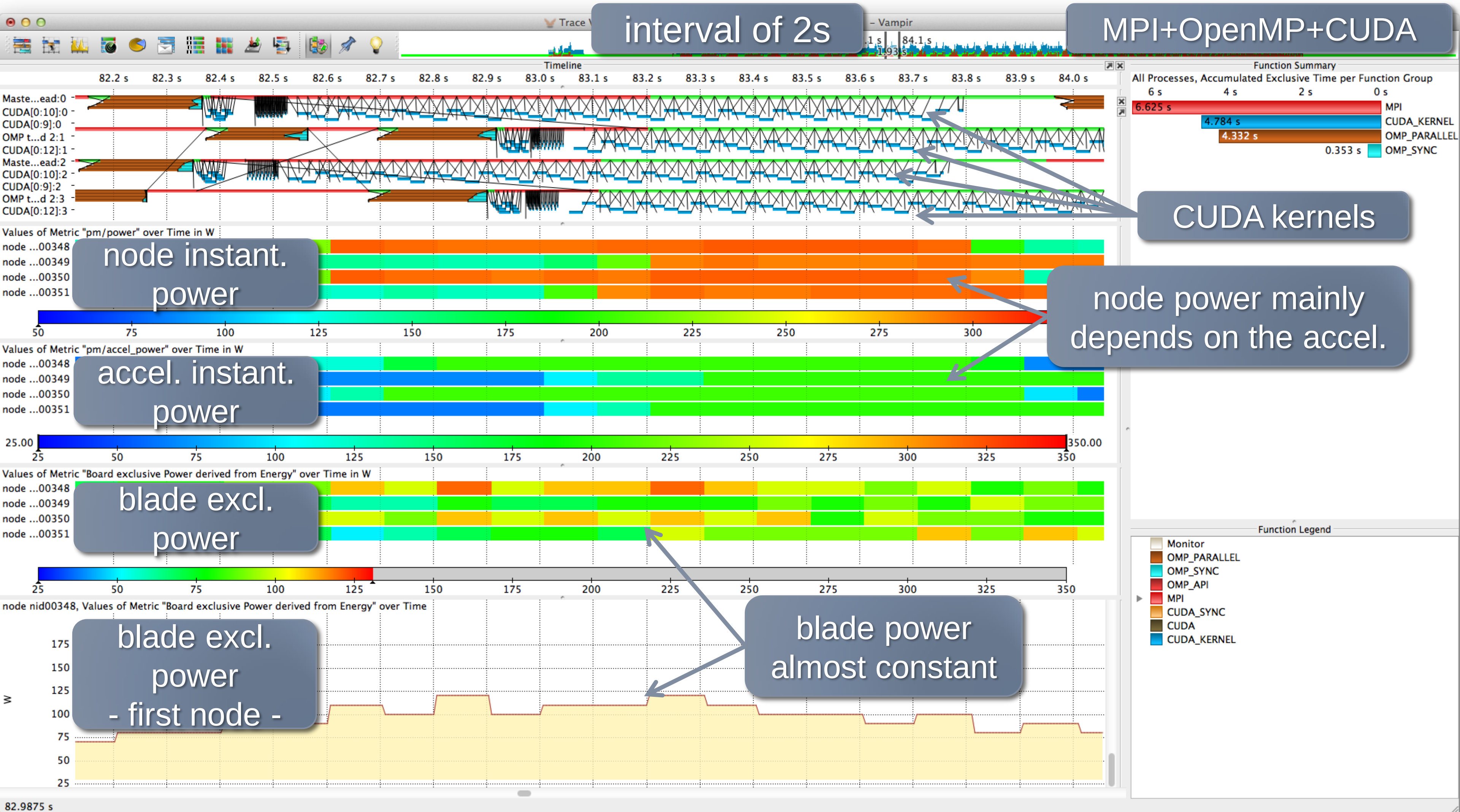
Energy Measurement – Test Patterns



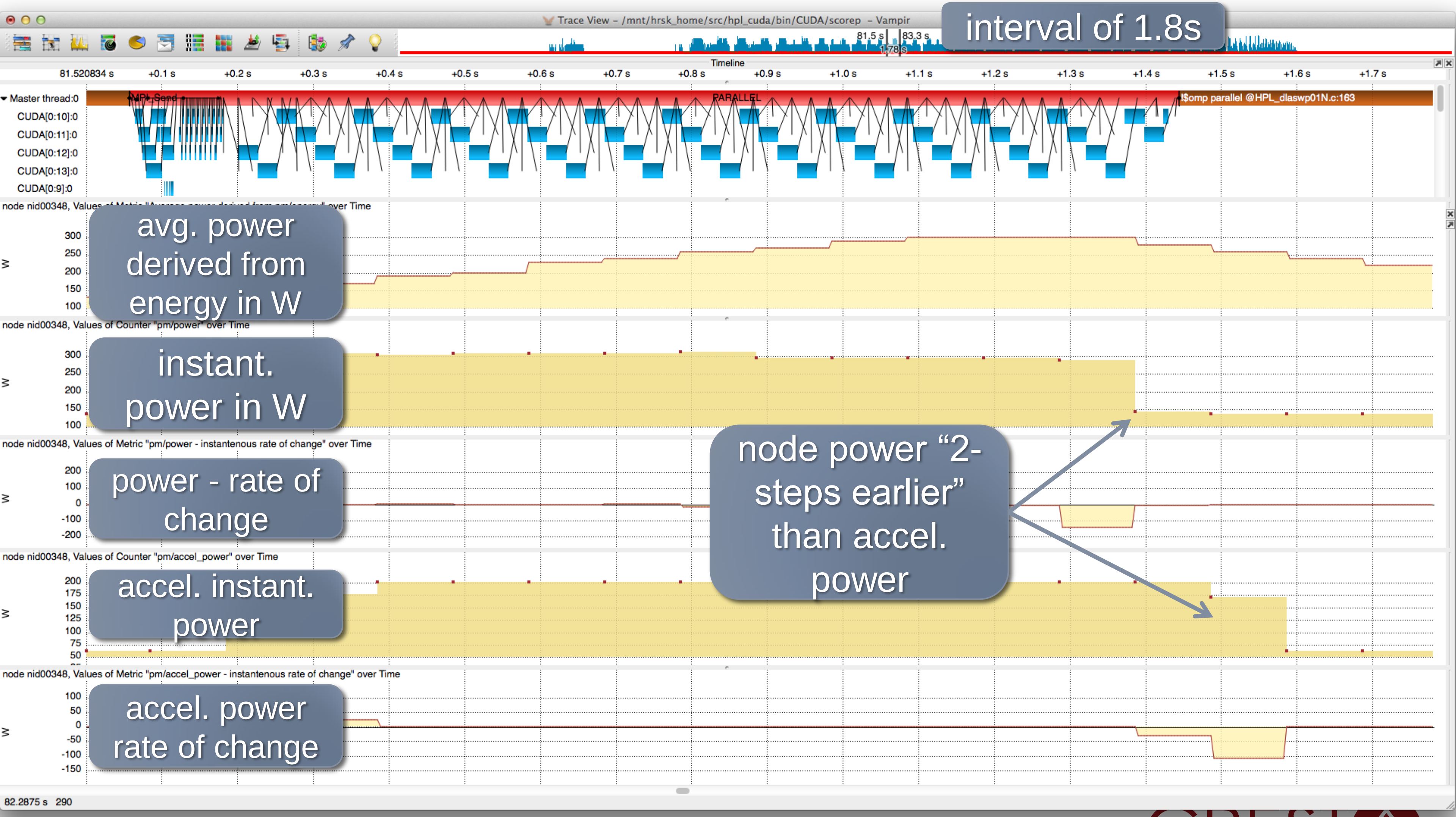
Energy Measurement – Test Patterns – zoomed in



Energy Measurement – HPL CUDA – four nodes

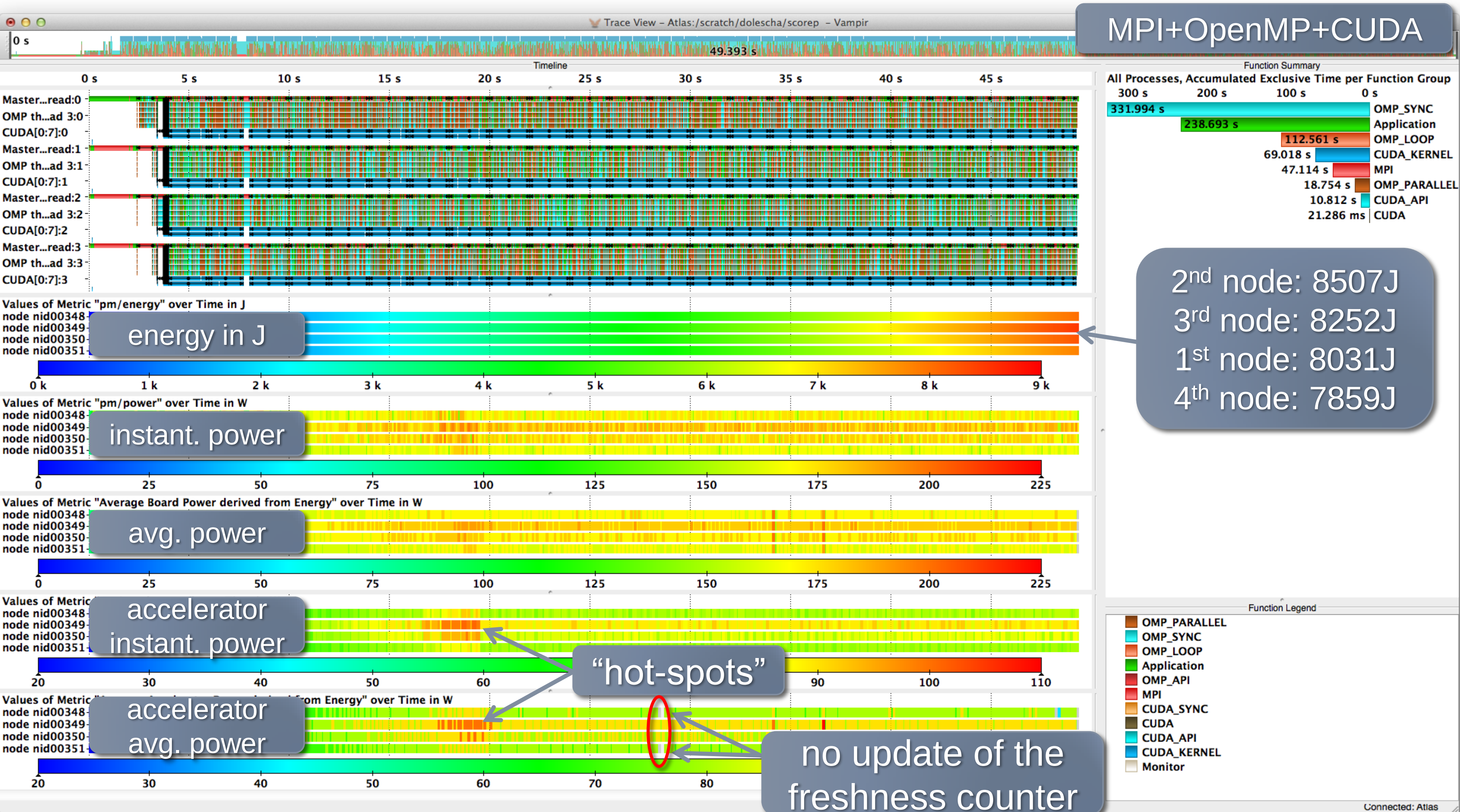


Energy Measurement – HPL CUDA – first node

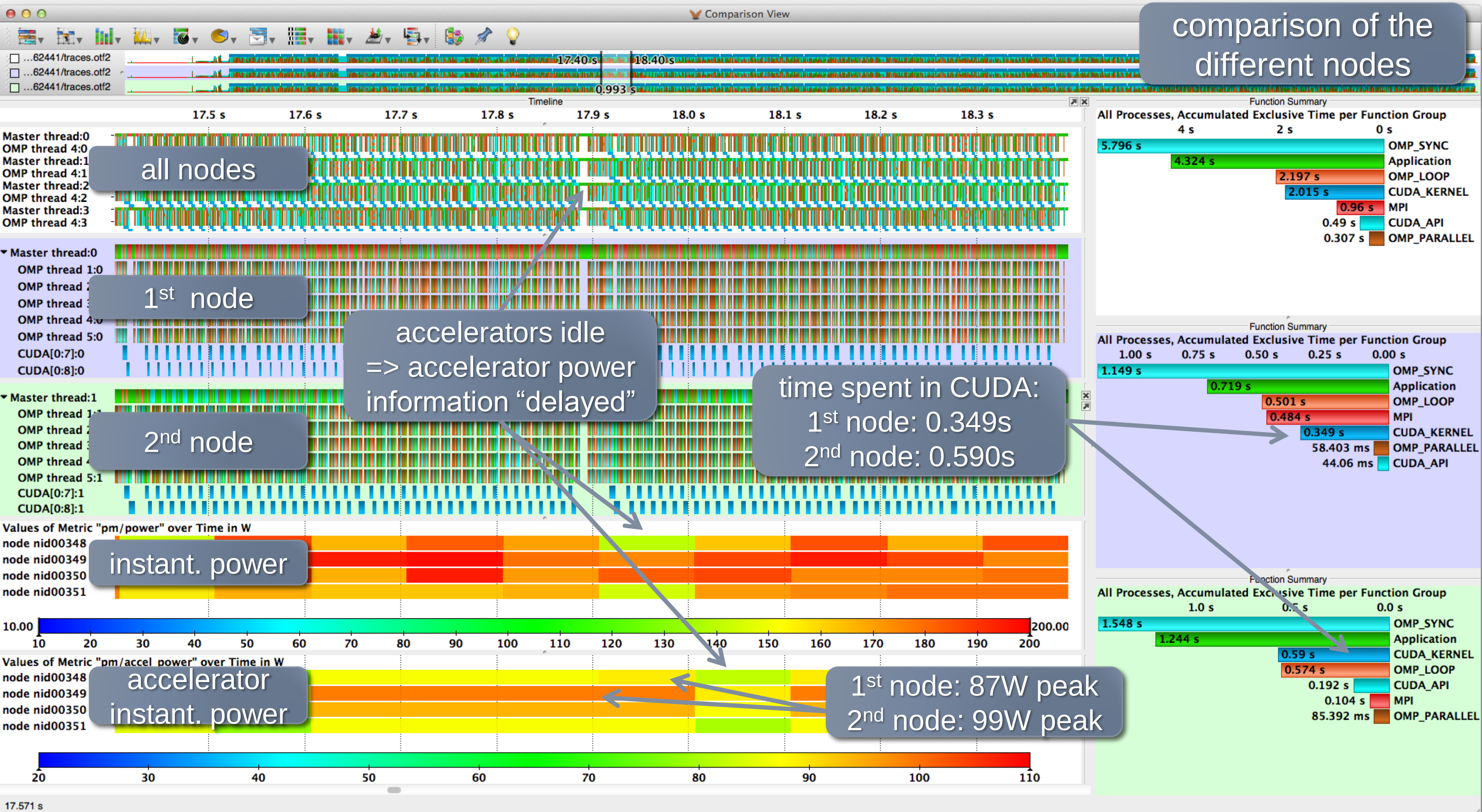


CRESTA

Energy Measurement – Gromacs – four nodes – 4k iterations



Energy Measurement – Gromacs – 80 iterations zoomed in



A performance model

- **Reducing the p-state lowers the energy consumption**
  - By a factor of $E \leq 1$
- **At the cost of an increased runtime**
  - By a factor of $R \geq 1$

- **Is this a good thing?**
  - That is, economically viable

- **Energy is only one part of the system cost**
  - Energy costs C per year
  - The initial capital cost is S
    - depreciates over system lifetime T
    - Typically, $^{C}/_{S} \approx 5\%$
  - Would normally run N jobs in this time
    - Increased runtime means fewer jobs run

A performance model

- **A simple model captures this:**
  - Running at the highest p-state, cost per job:

$$K_0 = \frac{S}{NT} + \frac{C}{N}$$

  - Running at a reduced p-state, cost becomes:

$$K_1 = \frac{SR}{NT} + \frac{CE}{N}$$

  - Payback time is when K_1 becomes less than K_0:

$$T_{pb} \geq \frac{R-1}{1-E} \times \frac{S}{C}$$

- **For the MG benchmark, $R \approx 1.05$ and $E \approx 0.85$**
  - So $T_{pb} \approx 6.7$ years
    - much longer than typical system lifetime
  - So reducing the p-state is not yet worth it
    - but hardware will improve and energy prices may rise

Conclusions

- **Users can now measure energy consumption of their applications on Cray XC systems**
  - At least for node-specific components
- **Tuning for different metrics is (quasi-) independent**
  - Runtime; Energy consumption; Power; Performance:power ratio
- **Real applications need energy/power tracing**
  - Score-P/Vampir has been adapted to do this
- **Current CPUs don't justify using a slower clockspeed**

Do you have any questions?

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.