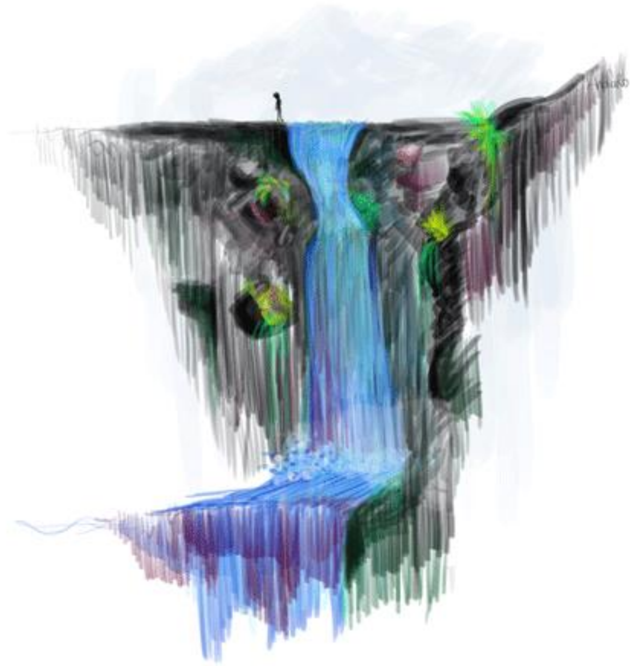
Power Update
x86 vs. ARM

Recent Updates

- ❖ Re-done the IPMI validation measurement (IPMItool vs. metered plug)
- ❖ Completed a few more comparison between the **x86** (w/out HT) & **arm:**
  - Idle measurements,
  - HEP-Score workloads available so far,
  - Thread-Scan.
- ❖ Testing the HEP-Score suite 2023 (and result submission)
- ❖ Developing an Energy plug-in for the HEP-Score suite
- ❖ Upcoming ARM cluster at Glasgow site
- ❖ …



ScotGrid @ Glasgow: Emanuele Simili, Gordon Stewart, Samuel Skipsey, Dwayne Spiteri, Albert Borbely, David Britton

Available Hardware

Test machines: we have two almost identical servers of comparable price, one with an AMD **x86_64** CPU (48c/96t), the other with an Ampere **arm64** CPU (80c):

x86_64: Single AMD EPYC 7003 series (GigaByte)

CPU:	AMD EPYC 7643 48C/96T @ 2.3GHz (TDP 300W)
RAM:	256GB (16 x 16GB) DDR4 3200MHz
HDD:	3.84TB Samsung PM9A3 M.2 (2280)

arm64: Single socket Ampere Altra (GigaByte)

CPU:	ARM Q80-30 80C @ 3GHz (TDP 210W)
RAM:	256GB (16 x 16GB) DDR4 3200MHz
HDD:	3.84TB Samsung PM9A3 M.2 (2280)

The **x86_64** CPU can run with Hyper-Threading (96 HT cores), or without (48 physical cores). Hyper-threading does not double performances, but adds 10-20% (roughly: 1 ht core ~ 55-60% of 1 physical core, depending on the task).
The **arm64** CPU has no such feature, therefore it can only run with its 80 physical cores.

In the following tests we have compared:
AMD hyperthreaded (**x86 HT**), AMD non hyperthreaded (**x86 noHT**), and ARM (**arm**).

IPMItool validation

IPMI validation

We have re-done a validation of the IPMI readings using our relatively cheap metered plugs (~ 30£ each):

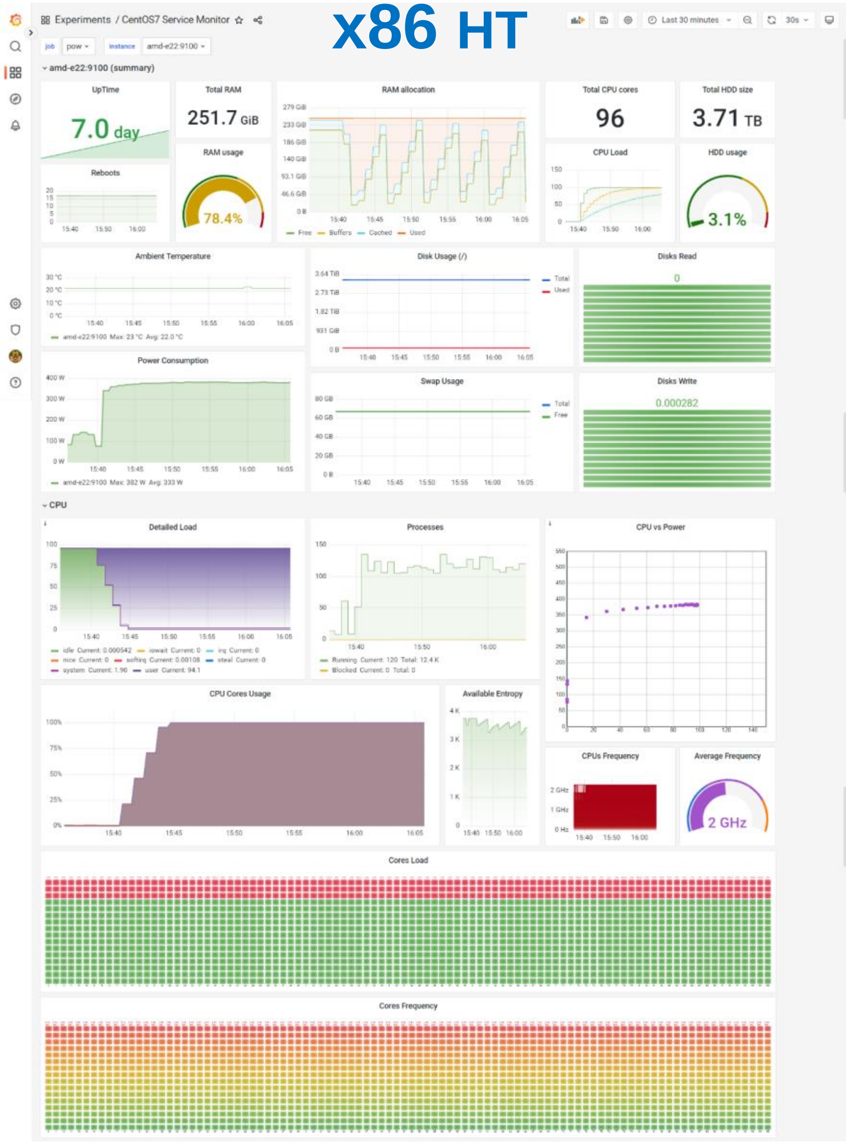
- ❖ Instantaneous power is impossible to compare, as the number changed too quickly on the metered plugs and they almost never matched the IPMI readings from the machine.
- ❖ We did an integrated measurement of the total energy for a fixed time (1 hour) of min. (**idle**) and max. (**stress**) power usage.
- ❖ The total energy calculated by integrating IPMI readings (1 Hz frequency) was compared to the kWh reading from the metered plug by resetting the meter on start and taking readings after 1h.

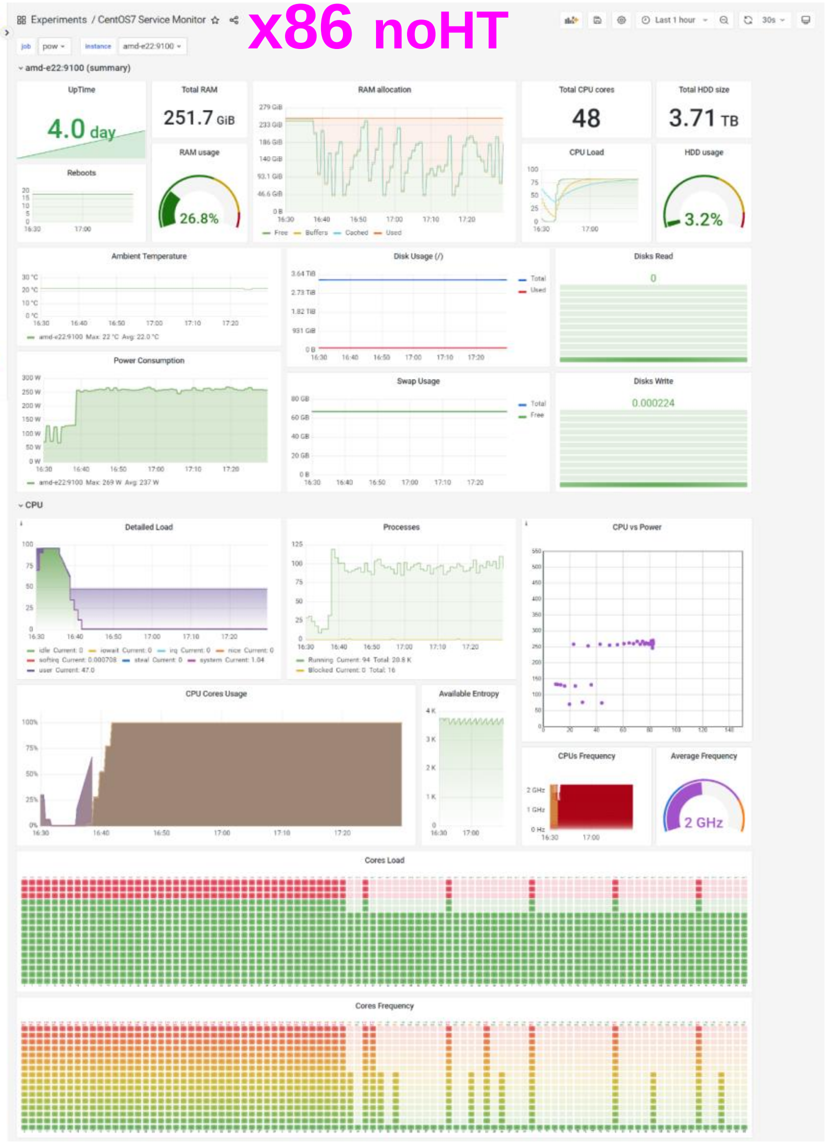
Each server has two redundant power supplier, therefore we tried 3 different settings:

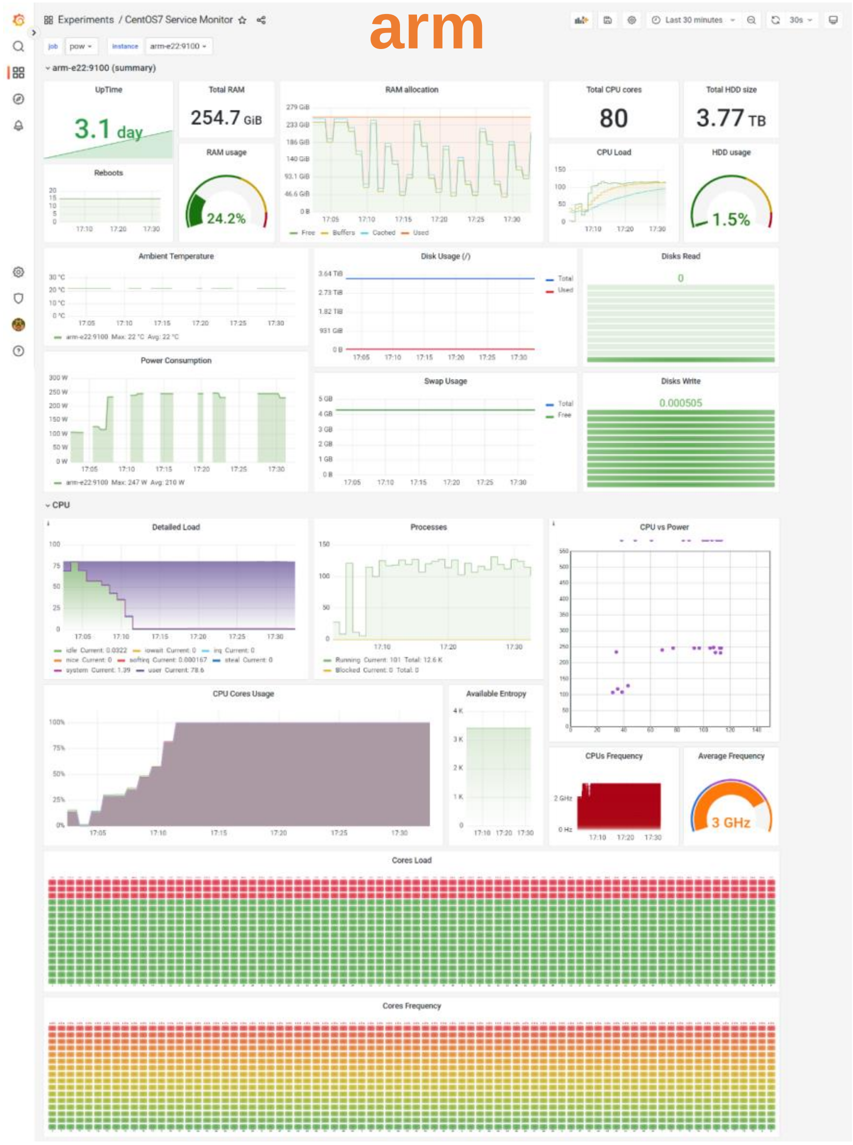
- ✓ 1 power supplier to 1 metered plug (the other supplier was disconnected),
- ✓ 2 power supplier to 1 metered plug (using a split plug),
- ✓ 2 power supplier to 2 metered plugs (and readings were added up).

Monitoring Power

Grafana dashboard visualizing the execution of a *stress* test on the **arm** and **x86**, (w/o hyper-threading):



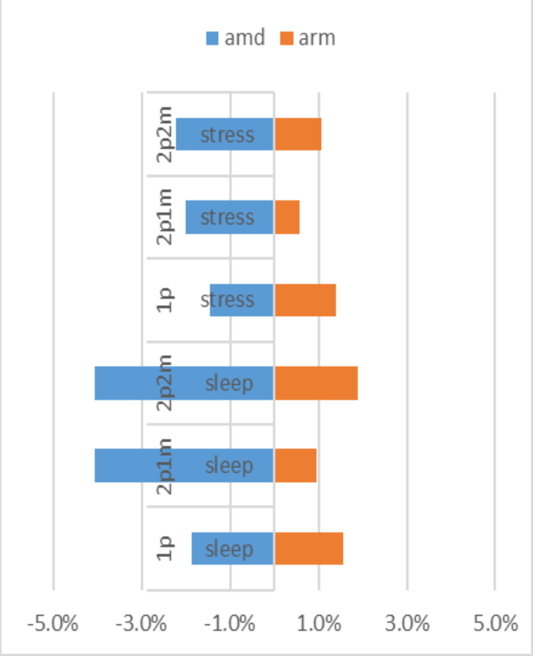




Validation Results

Results were a bit confusing **(*)**, with the discrepancy changing sign between **arm** & **x86** …
However, the error is small enough, with the <u>highest discrepancy being ~ **4%**</u>, and comes from idle!

					IPMI exporter					Manual read (1h)			diff (IPMI - Plug)	
job	machine	HT	power(s)	meter(s)	Time (s)	Energy (kWh)	Min (W)	Max (W)	Avg (W)	Energy (kWh)	Min (W)	Max (W)	Energy	% diff/IPMI
sleep	amd	HT	1	1	3601	0.086	60	148	86	0.088	82	135	-0.002	@ -1.9%
sleep	amd	HT	2	1	3601	0.093	72	150	93	0.097	87	156	-0.004	-4.1%
sleep	amd	HT	2	2	3600	0.092	72	163	92	0.096	89	154	-0.004	-4.1%
stress	amd	HT	1	1	3600	0.291	68	296	291	0.295	267	300	-0.004	# -1.5%
stress	amd	HT	2	1	3630	0.379	76	446	376	0.387	334	396	-0.008	-2.0%
stress	amd	HT	2	2	3600	0.372	77	377	372	0.380	334	388	-0.008	-2.2%
sleep	arm	//	1	1	3600	0.095	93	114	95	0.094	94	109	0.001	1.6%
sleep	arm	//	2	1	3599	0.104	87	131	104	0.103	102	116	0.001	1.0%
sleep	arm	//	2	2	3599	0.104	52	123	104	0.102	102	118	0.002	1.9%
stress	arm	//	1	1	3599	0.236	94	244	236	0.233	197	240	0.003	1.4%
stress	arm	//	2	1	3600	0.241	92	248	241	0.240	105	246	0.001	0.6%
stress	arm	//	2	2	3600	0.239	93	247	239	0.236	207	246	0.003	1.1%
sleep	amd	noHT	1	1	3601	0.134	116	151	134	0.135	133	143	-0.001	@ -0.7%
stress	amd	noHT	1	1	3600	0.259	131	269	259	0.263	241	272	-0.004	# -1.4%



There is some idea about fitting the data separately for the two servers, with a slope and an intercept to model the efficiency and power lost of each power supplier…

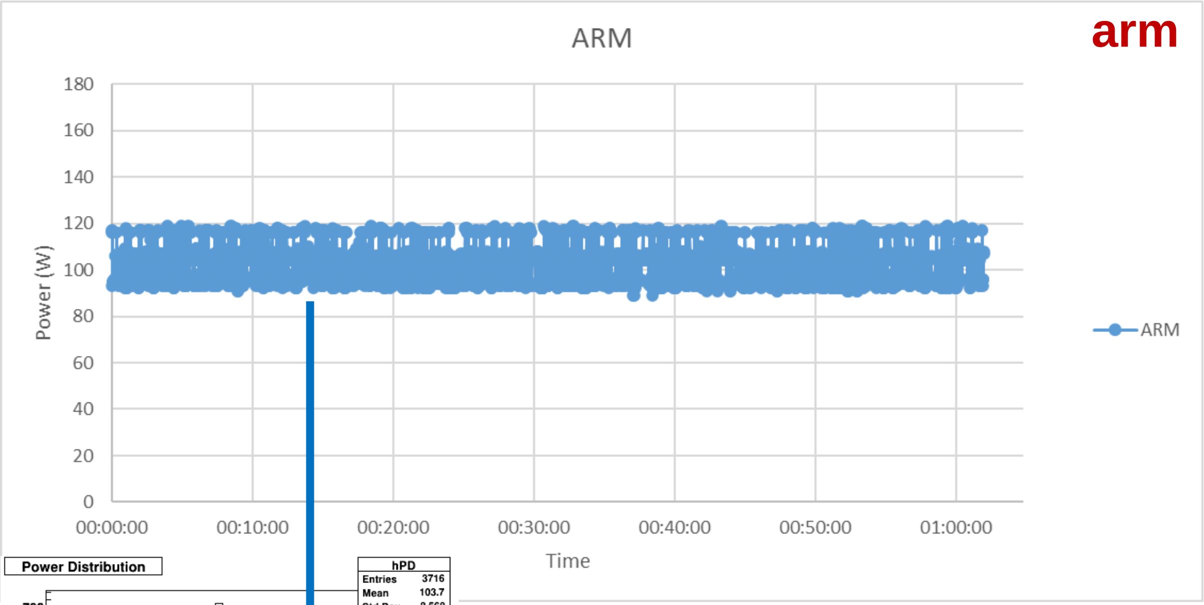
… but I should to collect more data, as an interpolation over 2 points is not ideal. For instance, repeat the measurement for various level of power usage (~ % CPU usage).

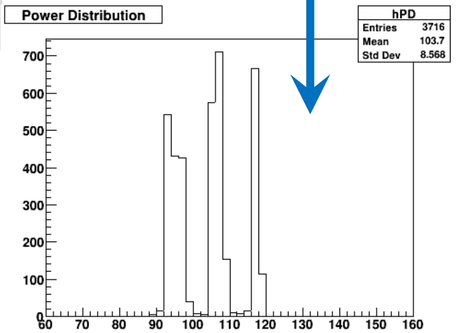
(*) the precision of the relatively cheap metered plug might be questionable, as well as the human error in starting/stopping the measurement (±1sec ?)

idle

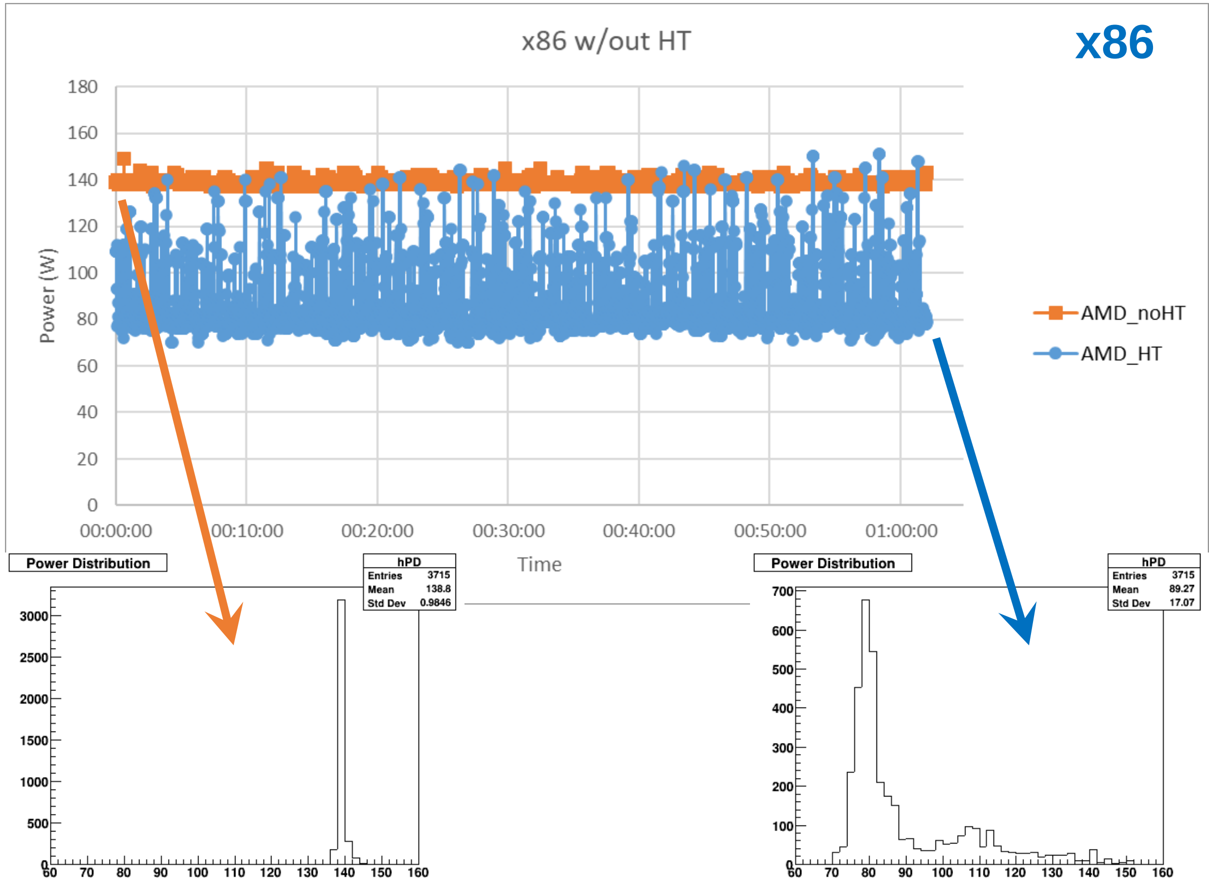
Measuring Idle

Idle measurements show that machines oscillate between different power states when idling. Also, each machine seems to have its own peculiarity …





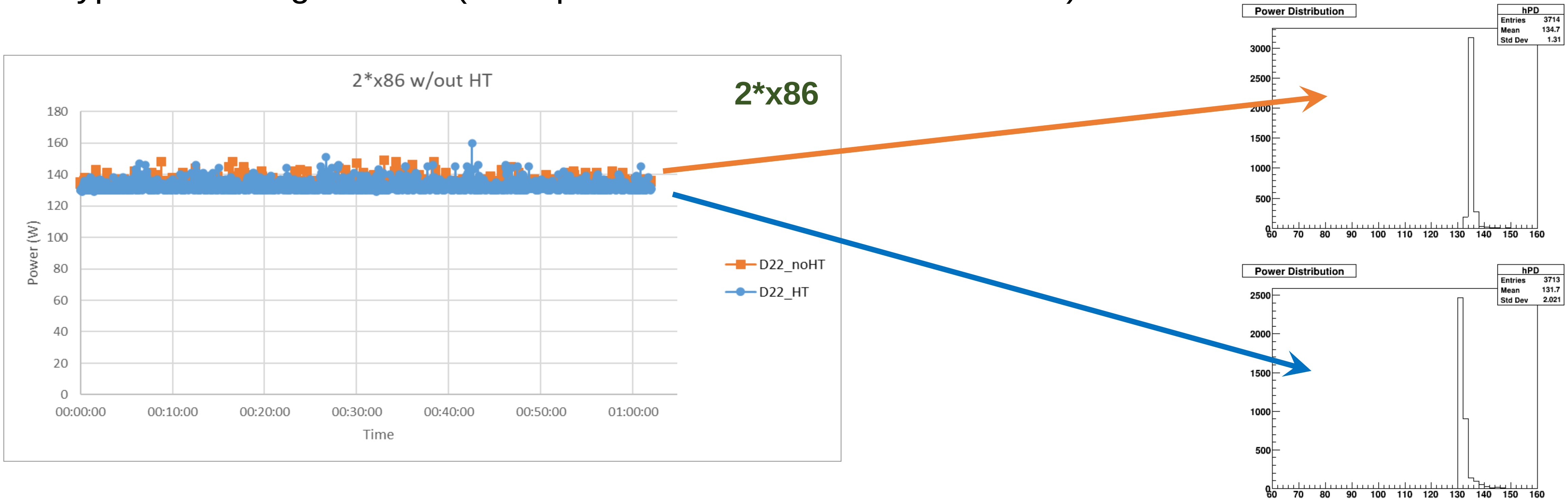
The **arm** oscillates between 3 power states while in idle, averaging at ~104 Watts.



The **x86** seems to use more idle power when hype-threading is turned off (139 – 89 = 40 Watts). It is possible that this particular chip enables some sort of power optimization together with hyper-threading … however this only affects the idle state. Peak power usage does not change with HT.

Measuring Idle

By comparison, another **x86** machines shows a less significant change in idle power usage (~ 3 W). when hyper-threading is on/off (example from a standard workernode):



2*x86_64: Dual AMD EPYC 7513 series Processors (DELL)

CPU: 2 * AMD EPYC 7513, 32C/64T @2.6GHz (TDP 200W)
RAM: 512GB (16 x 32GB) DDR4 3200MHz
HDD: 3.84TB SSD SATA Read Intensive

* this machine is part of a 2 unit / 4 node chassis.

thread-scan

Thread-Scan

I have used the latest HEP-Score containers available for **arm** & **x86** to characterize performance and power consumption with respect to the number of threads.

This was done by fixing the number of threads per copy (-t 1 or 4 depending on the experiment), and running an increasing number of copies (-c *N*) till saturating the CPU.

```
for NCP in 1 8 20 24 40 48 60 80 96 100
do
  singularity run -B ${EXEDIR}:${RESDIR} oras://${CONTAINER} -c ${NCP} -t ${NTH} -e ${NEVTS}
done
```

List of HEP-Score containers used:

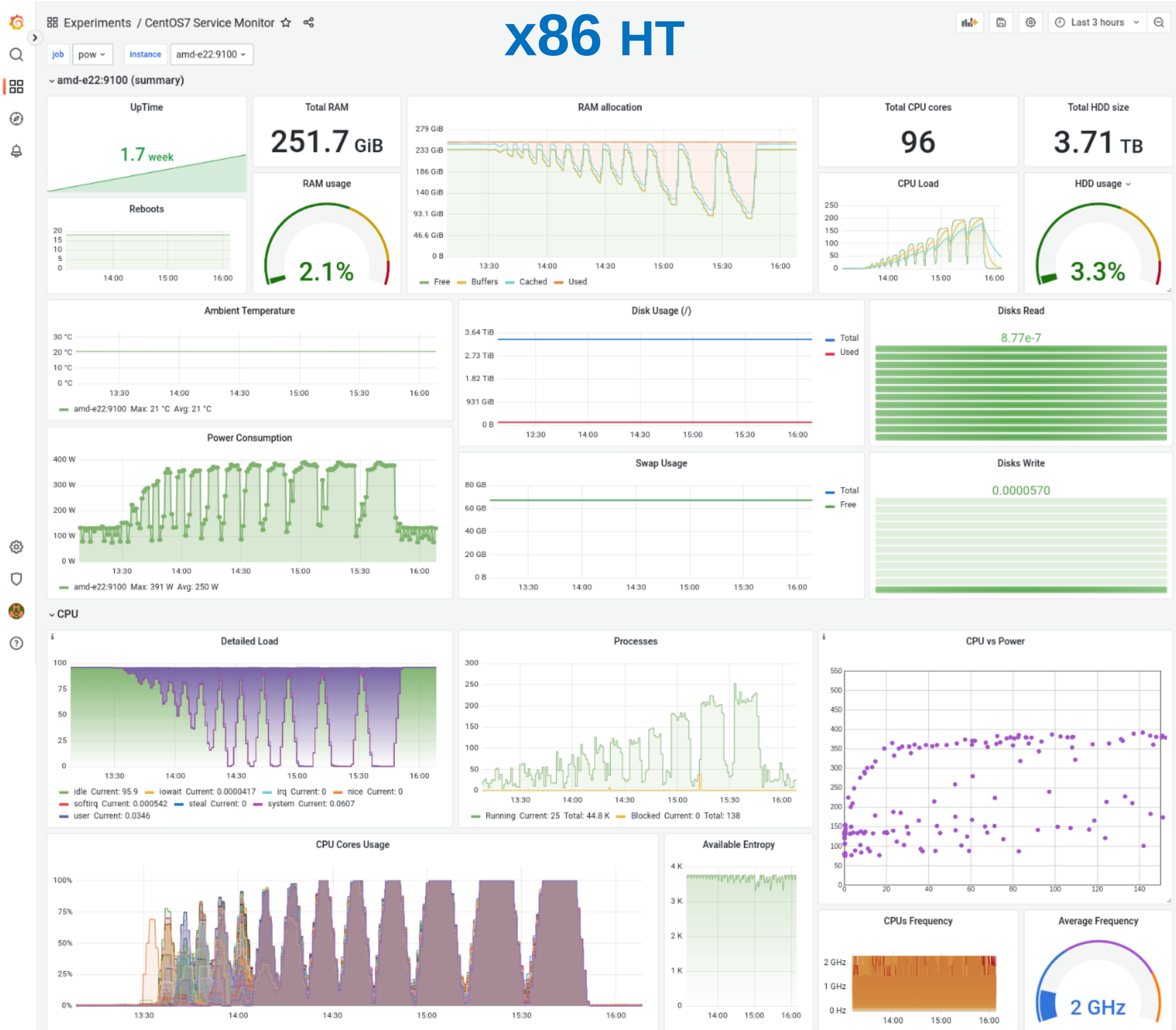
(except the **ATLAS Sim**, all these 7 containers are included in the 2023 release of HEP-Score ~v2.1)

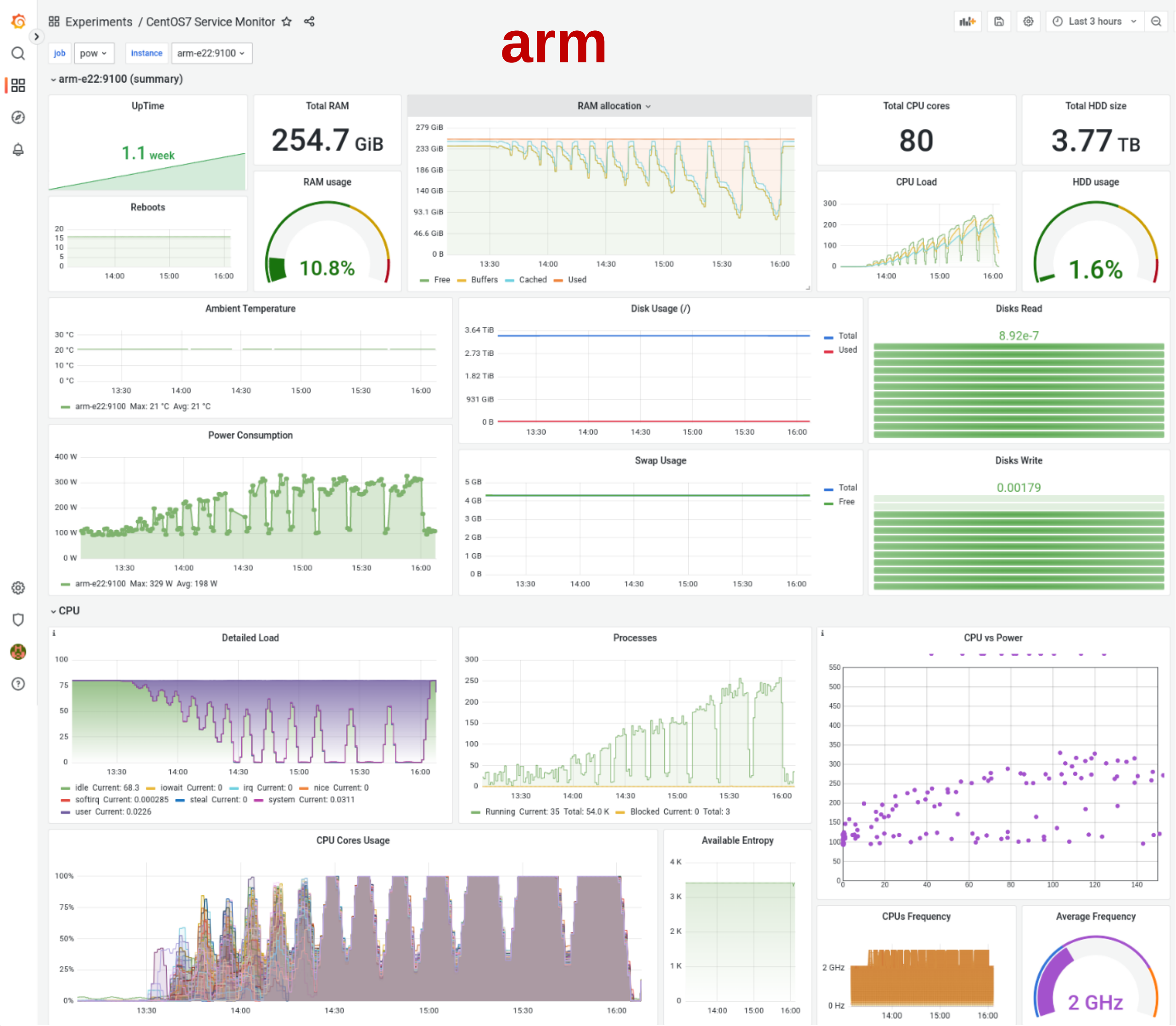
WorkLoad	Container	thr/cp	evt/thr
ATLAS Sim	atlas-sim_mt-ma-bmk	4	20
CMS Gen-Sim	cms-gen-sim-run3-ma-bmk	4	100
Belle2 Gen-Sim-Reco	belle2-gen-sim-reco-ma-bmk	1	50
ALICE Deigi-Reco	alice-digi-reco-core-run3-ma-bmk	4	3
ATLAS Gen Sherpa	atlas-gen_sherpa-ma-bmk	1	500
ATLAS Reco	atlas-reco_mt-ma-bmk	4	100
CMS Reco	cms-reco-run3-ma-bmk	4	100
LHCb Sim	lhcb-sim-run3-ma-bmk	1	5

See: https://gitlab.cern.ch/hep-benchmarks/hep-workloads-sif/container_registry

Example Job Profile

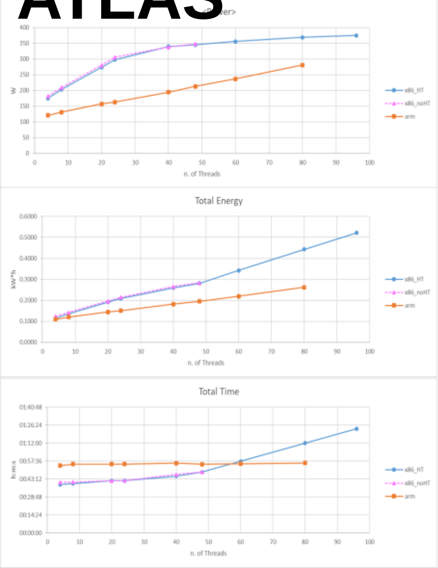
Grafana runtime profiles of the **LHCb** workload (from HEP containers). The workload was executed ten times, increasing the number of copies at each run to progressively fill the CPU …



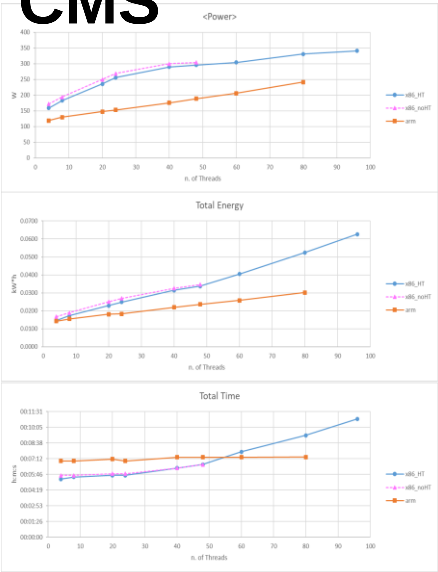


Thread Scan (8x)

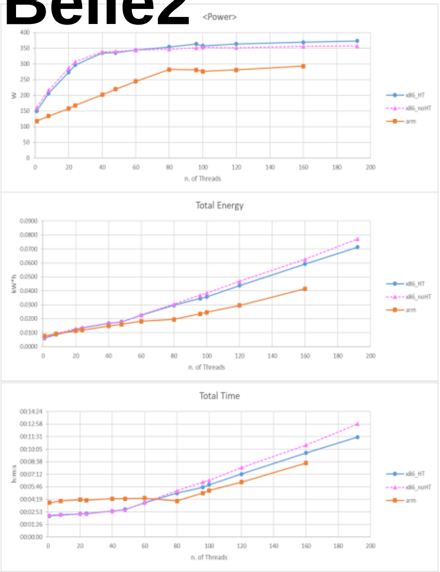
ATLAS



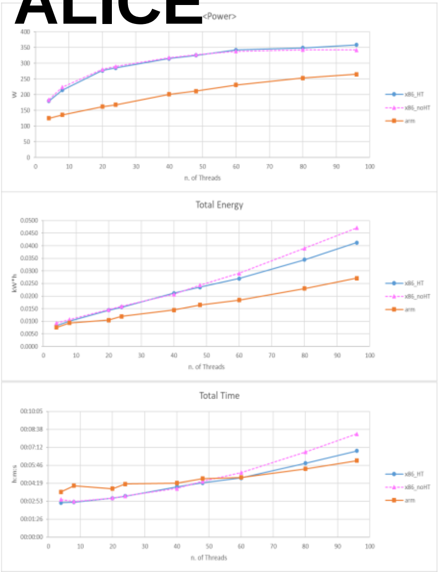
CMS



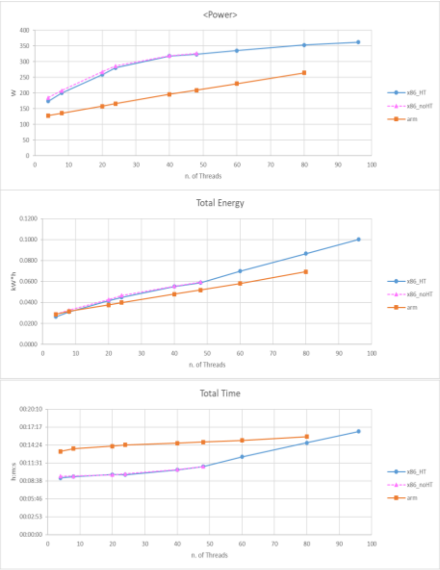
Belle2



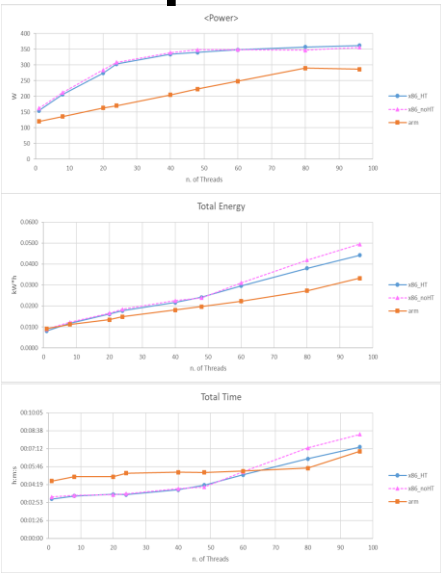
ALICE



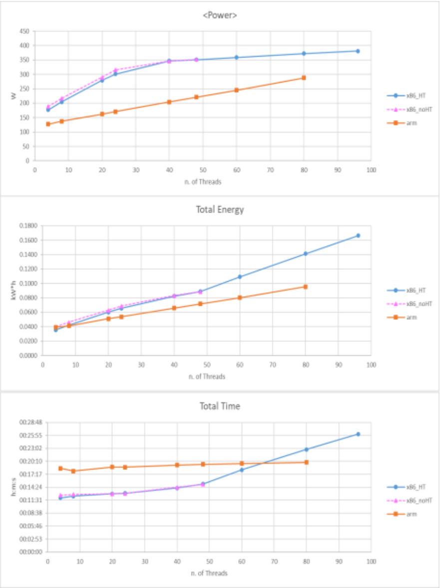
ATLAS reco



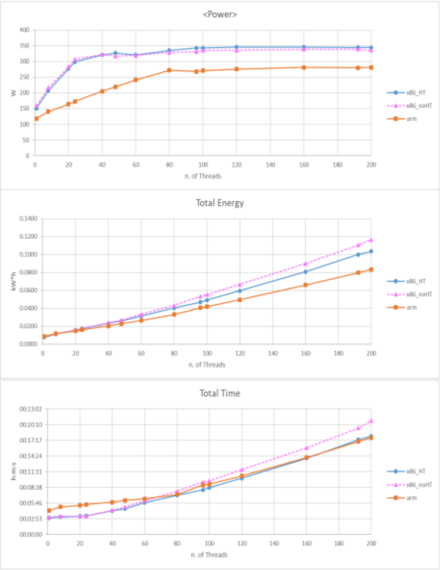
Sherpa



CMS reco



LHCb



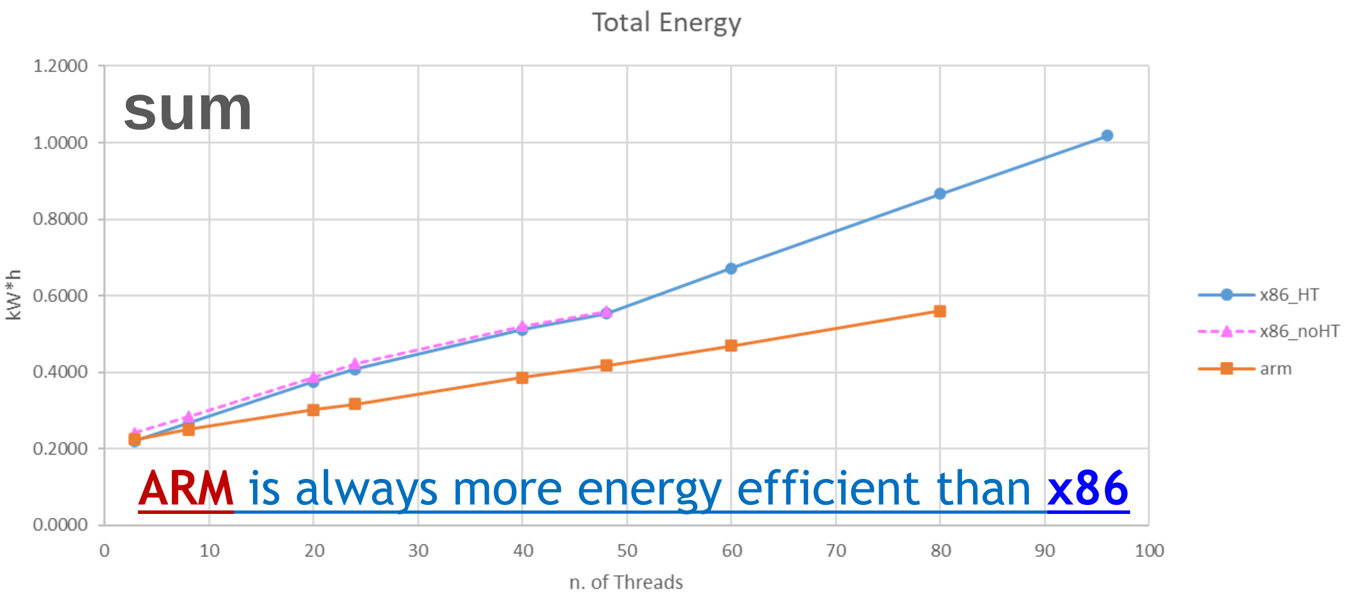
On **ARM**, the Energy (Power) increase linearly with the n. of threads, on **x86** saturates once hyper-threading starts.

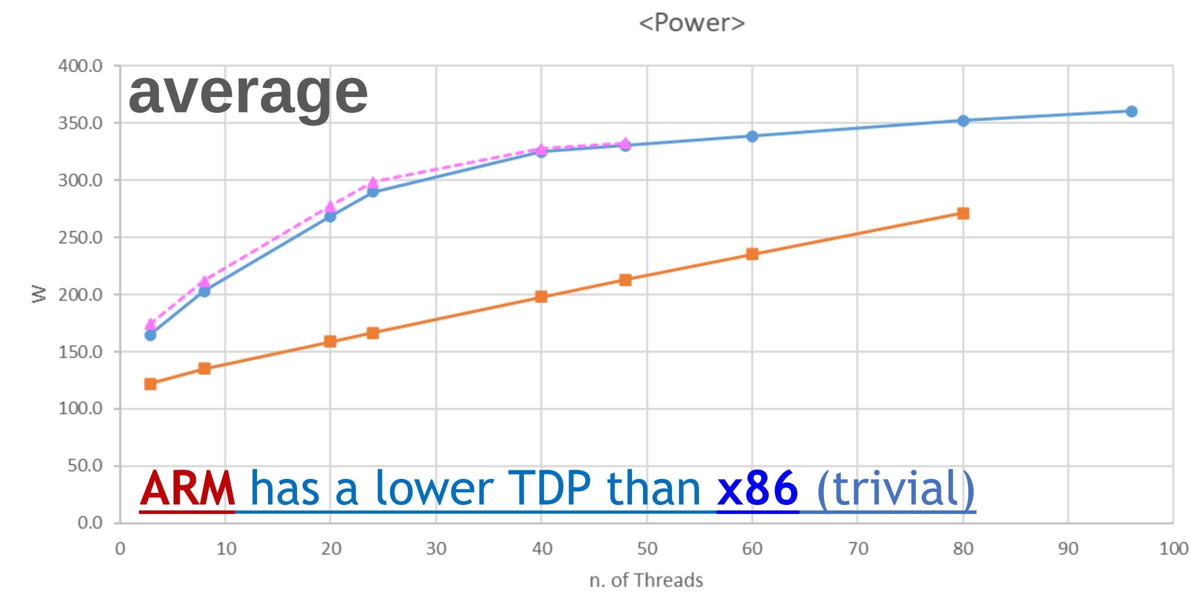
W.r.t. the n. of threads, the execution time is constant on **ARM**, while it increases on **x86** once hyper-threaded.

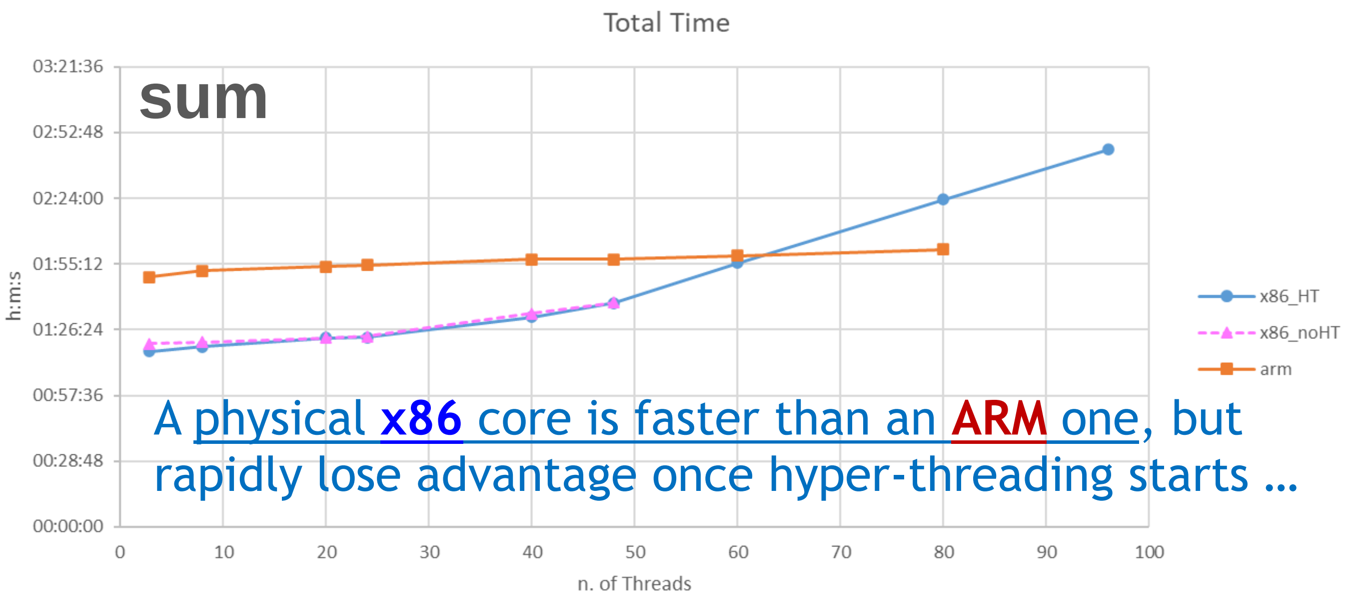
Note:

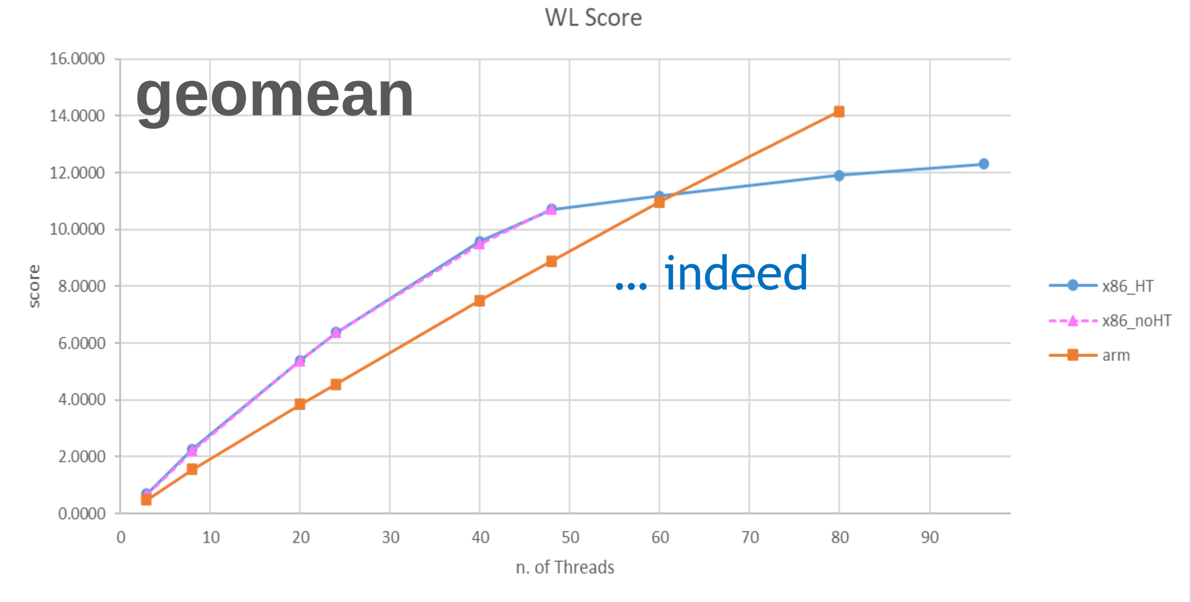
the 1st bin is different depending on the work-load, some use 1 thread (**belle2-gen-sim, atlas-gen_sherpa, lhcb-sim**), some use 4 (**atlas-sim, cms-gen-sim, alice-digi-reco, atlas-reco,cms-reco**) …

Thread Scan (averages)





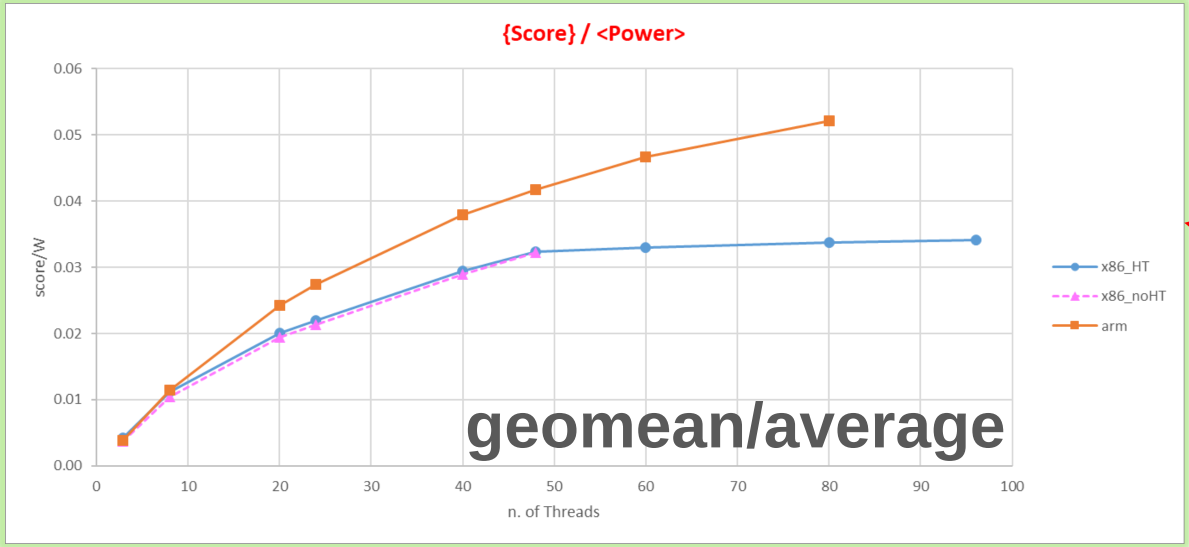


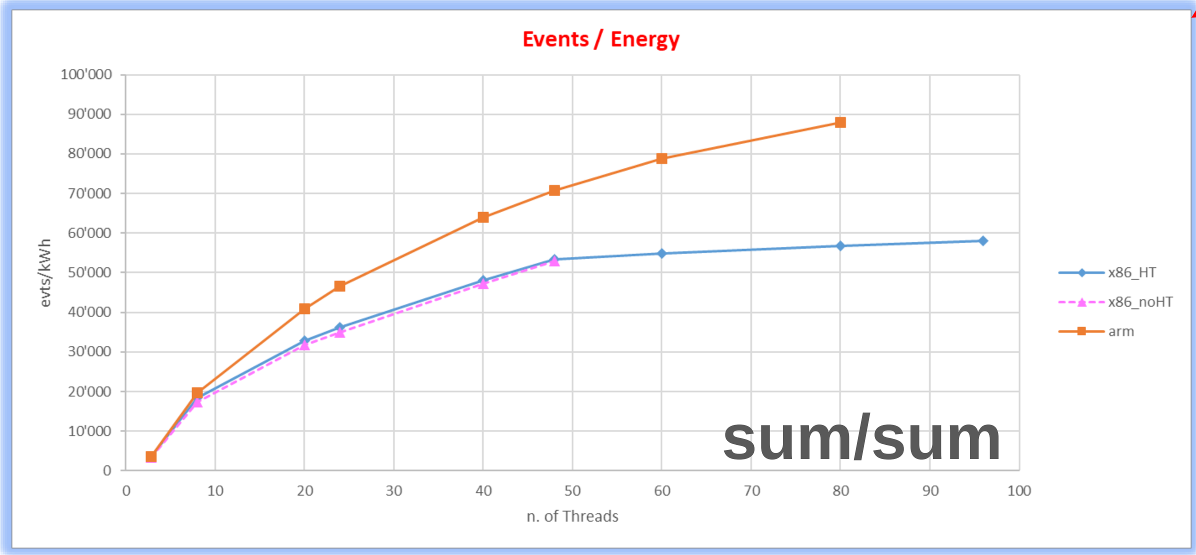


Thread Scan (scores)

As in the HEP-Score suite, we can combine the various scores using a Geometric Mean:

$$\{\textbf{Score}\} = \sqrt[n]{\prod_i (\text{WL-Score}_i)}$$





We can then plot the:

{Score} / Watt

Which is equivalent to:

Events / Total Energy

(with some conversion constant *k*).

Conclusions:

When Power (or Energy) is taken into account, the **ARM** appears to be the best choice for HEP workloads.

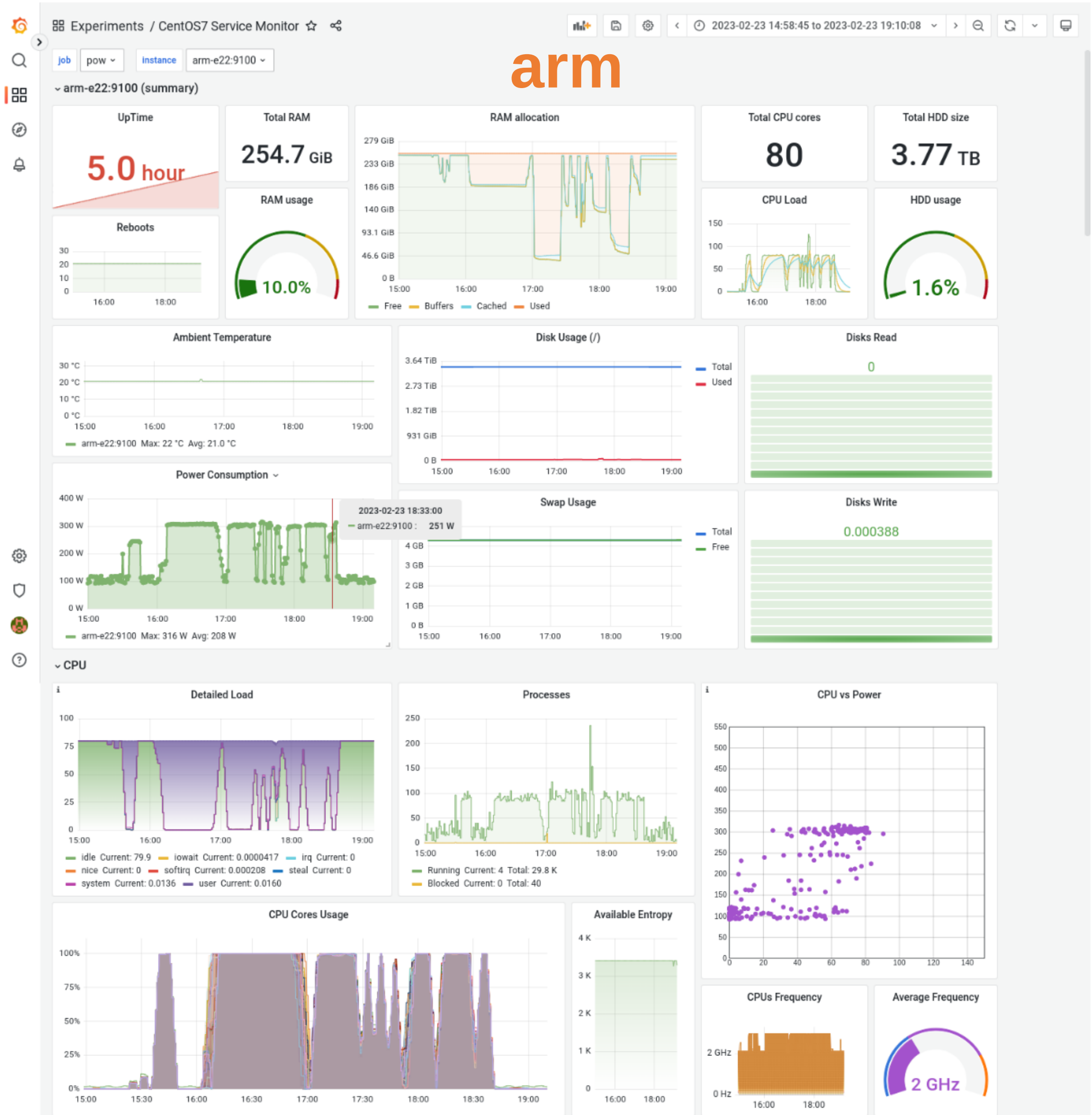
The higher performance per cores of the **x86** are lost once hyper-threading starts, and the higher number of physical cores on the **ARM** makes it way more performant !

see Excel file for details about the plots shown the last 3 slides: threadScan_gpp49.xlsx

HEP-Score 2023

HEP-Score 2023 run

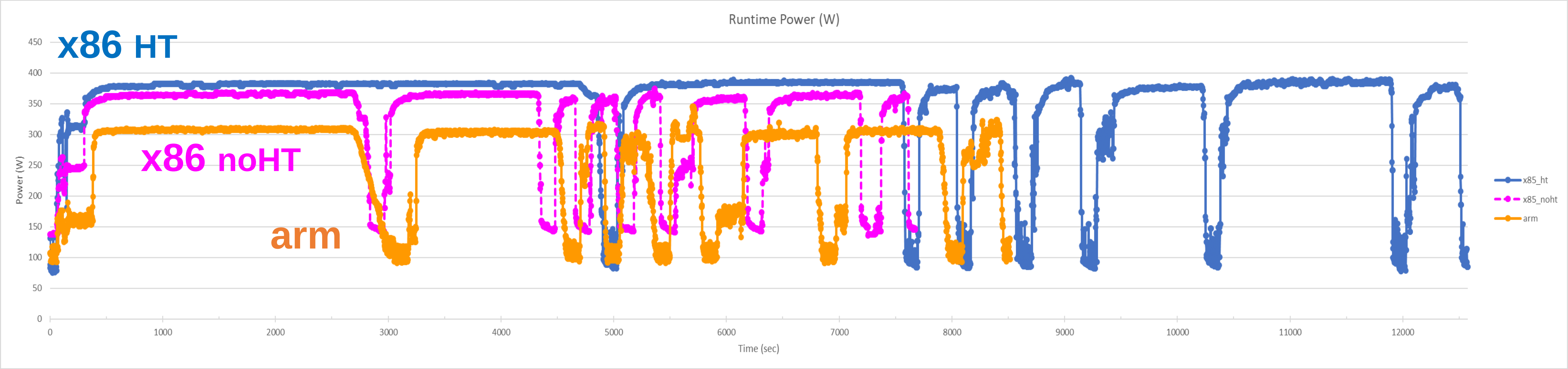
Grafana runtime profile of a full run of the 8 most recent HEP-Score containers available for **arm** & **x86**:



Power Profiles (runtime)

Runtime power profile extracted from the **arm** and the **x86** (with and without Hyper-Threading).

<u>Note:</u> the **x86 HT** run takes the longest because, having more threads, is running more copies **(*)**



Full list of workloads
(in order of execution).

The number of copies (**cp** column) refers to the **x86 HT** with 96 total threads available.

WorkLoad	Container	cp	thr/cp	evt/thr	tot evts
ATLAS Sim	atlas-sim_mt-ma-bmk	24	4	20	1920
CMS Gen-Sim	cms-gen-sim-run3-ma-bmk	24	4	100	9600
Belle2 Gen-Sim-Reco	belle2-gen-sim-reco-ma-bmk	96	1	50	4800
ALICE Deigi-Reco	alice-digi-reco-core-run3-ma-bmk	24	4	3	288
ATLAS Gen Sherpa	atlas-gen_sherpa-ma-bmk	96	1	500	48000
ATLAS Reco	atlas-reco_mt-ma-bmk	24	4	100	9600
CMS Reco	cms-reco-run3-ma-bmk	24	4	100	9600
LHCb Sim	lhcb-sim-run3-ma-bmk	96	1	5	480

HEP-Score results

Again, as I am not running within the HEP-Score suite, I should normalize each WL-Scores myself.
Luckily there are <u>reference values</u> available:

Arch	Max Thr	WorkLoad	cp	thr/cp	evt/thr	tot evts	Time (H:m:s)	Time (s)	Energy(kW*h)	<Pow> (W)	WL score	ref_scores	score_n	score_n/W	score_n/kWh	score_n/(kWh/cp)
x86_HT	96	ATLAS Sim	24	4	20	1920	01:22:48	4968	0.5148	373	0.4077	0.286	1.43	0.004	2.77	66.52
x86_HT	96	CMS Gen-Sim	24	4	100	9600	00:43:12	2592	0.2715	377	3.7347	2.665	1.40	0.004	5.16	123.88
x86_HT	96	Belle2 Gen-Sim-Reco	96	1	50	4800	00:05:46	346	0.0347	361	20.2715	15.400	1.32	0.004	37.98	3'645.92
x86_HT	96	ALICE Deigi-Reco	24	4	3	288	00:07:00	420	0.0407	349	0.7713	0.762	1.01	0.003	24.85	596.29
x86_HT	96	ATLAS Gen Sherpa	96	1	500	48000	00:07:32	452	0.0443	353	112.8831	38.580	2.93	0.008	66.09	6'344.95
x86_HT	96	ATLAS Reco	24	4	100	9600	00:16:33	993	0.0980	355	12.1419	9.062	1.34	0.004	13.67	328.13
x86_HT	96	CMS Reco	24	4	100	9600	00:26:11	1571	0.1642	376	6.4016	4.814	1.33	0.004	8.10	194.34
x86_HT	96	LHCb Sim	96	1	5	480	00:08:51	531	0.0480	325	2'803.8279	1'950.000	1.44	0.004	29.97	2'876.92
x86_HT	96					0										
x86_noHT	48	ATLAS Sim	12	4	20	960	00:47:30	2850	0.2759	349	0.3684	0.286	1.29	0.004	4.67	56.08
x86_noHT	48	CMS Gen-Sim	12	4	100	4800	00:23:30	1410	0.1397	357	3.4565	2.665	1.30	0.004	9.29	111.43
x86_noHT	48	Belle2 Gen-Sim-Reco	48	1	50	2400	00:03:06	186	0.0174	337	19.2020	15.400	1.25	0.004	71.66	3'439.67
x86_noHT	48	ALICE Deigi-Reco	12	4	3	144	00:04:40	280	0.0249	321	0.5973	0.762	0.78	0.002	31.44	377.31
x86_noHT	48	ATLAS Gen Sherpa	48	1	500	24000	00:04:06	246	0.0236	345	100.9070	38.580	2.62	0.008	110.87	5'321.97
x86_noHT	48	ATLAS Reco	12	4	100	4800	00:11:06	666	0.0593	320	10.1683	9.062	1.12	0.004	18.94	227.26
x86_noHT	48	CMS Reco	12	4	100	4800	00:15:03	903	0.0875	349	5.7325	4.814	1.19	0.003	13.62	163.40
x86_noHT	48	LHCb Sim	48	1	5	240	00:04:51	291	0.0265	328	2'343.9215	1'950.000	1.20	0.004	45.32	2'175.59
x86_noHT	48															
arm	80	ATLAS Sim	20	4	20	1600	00:56:19	3379	0.2625	280	0.5445	0.286	1.91	0.007	7.26	145.19
arm	80	CMS Gen-Sim	20	4	100	8000	00:25:34	1534	0.1240	291	5.3713	2.665	2.02	0.007	16.25	325.03
arm	80	Belle2 Gen-Sim-Reco	80	1	50	4000	00:04:46	286	0.0222	280	21.2615	15.400	1.38	0.005	62.16	4'972.96
arm	80	ALICE Deigi-Reco	20	4	3	240	00:05:40	340	0.0232	245	0.8546	0.762	1.12	0.005	48.42	968.50
arm	80	ATLAS Gen Sherpa	80	1	500	40000	00:05:41	341	0.0272	287	128.0047	38.580	3.32	0.012	122.03	9'762.13
arm	80	ATLAS Reco	20	4	100	8000	00:15:52	952	0.0690	261	11.8797	9.062	1.31	0.005	18.99	379.76
arm	80	CMS Reco	20	4	100	8000	00:19:44	1184	0.0944	287	7.4774	4.814	1.55	0.005	16.46	329.26
arm	80	LHCb Sim	80	1	5	400	00:07:11	431	0.0327	273	3'263.3906	1'950.000	1.67	0.006	51.18	4'094.27
arm	80															

(*)

HEP-Score results (2)

Normalised scores already look higher for the **arm** than the **x86** (i.e., it is more efficient at these HEP tasks).

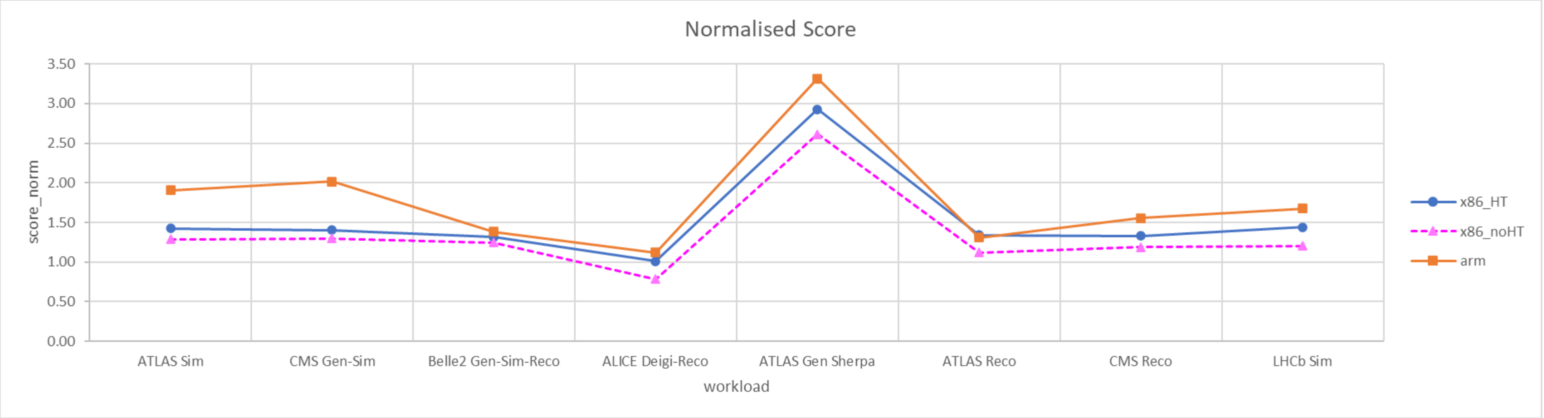
The advantage becomes even clearer when we consider the power usage: the **arm** uses less average power to deliver better performance than the **x86**.

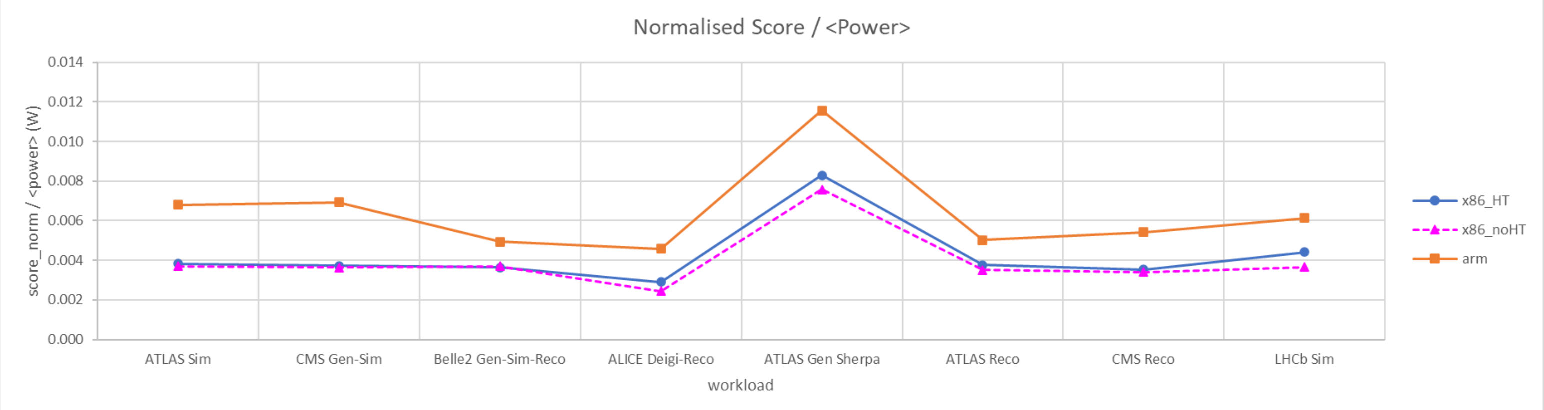
If we consider the total energy, we must take into account the different amount of work done, which depends on threads **(*)**:

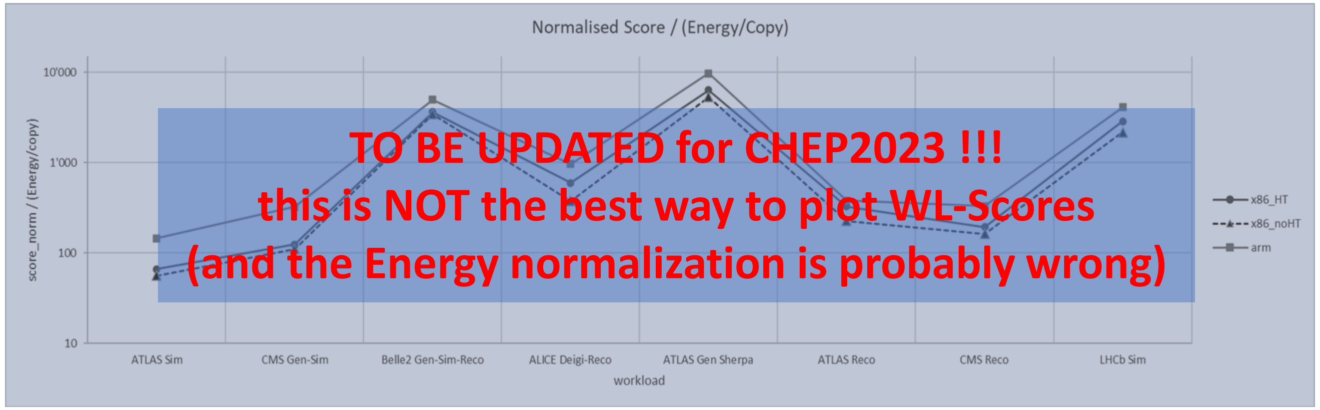
Normalized score over the scaled energy (on log scale, because …)

Geometric mean of the score over the total energy (scaled by the n. of copies):

x86_HT 31.13
x86_noHT 25.27
arm 57.01







energy_plugin

HEP-Score Energy Plug-In

I have committed a first implementation of the energy plug-in to the development branch of the HEP-Score suite and it is being integrated in the workflow

This **energy plug-in** is a Python module that runs alongside with the HEP-Score workloads, while extracting CPU and RAM usage, core Frequency, and IPMI (and GPU) power readings.
When the workload is finished, the plug-in will calculate a number of execution statistics and save these, together with the time-stamped runtime data, as a dictionary in a *json* file.

energy_data.json


```
{
  "measurements": {
    "2023-02-07T15:49:45.879999Z": {
      "cpu": 0.2,
      "frq": 1.5,
      "ram": 3.7,
      "powa": 82.0,
      "gpu": 0
    },
    "2023-02-07T15:49:47.115694Z": {
      "cpu": 0.2,
      "frq": 1.5,
      "ram": 3.7,
      "powa": 82.0,
      "gpu": 0
    },
    ...
```

```
  "statistics": {
    "totaltime": 10,
    "sampling": 1.0,
    "energy": 0.000251944444444444445,
    "gpu_energy": 0.0,
    "cpu_min": 0.1,
    "cpu_max": 0.2,
    "frq_min": 1.5,
    "frq_max": 1.5,
    "ram_min": 3.7,
    "ram_max": 3.7,
    "pow_min": 81.0,
    "pow_max": 85.0,
    "pow_avg": 82.45454545454545,
    "gpu_avg": 0.0
  }
```


For now the plugin had been tested standalone (by me).
It seems to work, but some more testing is needed.

It will be officially integrated into the HEP-Score release in v3.0 (?)

Energy Plug-In

The **energy_plugin** class implements an internal timer, which is used to regularly grab runtime metrics during execution, and an analyser which calculates execution stats from runtime metrics.

HEP dependencies (~50%)

Option for non-root user (!): grab metrics from dump file

Starts the timer (just before starting the job) …

Called at regular intervals to grab metrics, such as time-stamp, IPMI power reading, CPU usage, etc.

Calculates statistics using the 'trapezoidal sum', which takes care of irregular sampling intervals or missing time-stamps.

```
# Skeleton class by Gonzalo, content by Emanuele (v0.5)

import json, ...
from hepbenchmarksuite.plugins.stateful_plugin import StatefulPlugin
#from hepbenchmarksuite.plugins.extractor import Extractor

class EnergyPlugin(StatefulPlugin):
    ...
    self.dumpfile = "/tmp/ipminfo.txt“

    def start(self) -> None:

    # IPMI loop function (runtime): dumps system metrics to a dictionary
    def grab_metrics(self,start_time):
        time_stamp = dt.datetime.utcnow()
        time_key = str(time_stamp.isoformat(timespec='milliseconds')) + "Z"
        ...
        cmd_ipmi = r""" ipmitool dcmi power reading | grep "Instantaneous power reading:" """
        powa = self.get_numbers(self.run_command(cmd_ipmi),0)
        ...

    # IPMI analiser functions (postrun): calculates statistics and averages
    def calculate_statistics(self, measurements: dict) -> Dict[str, float]:
        ...
        for k in sorted(measurements.keys()):
            ...
            deltaSec = (time_stamp - time_prev).total_seconds()
            powAve = (powa + powa0) / 2
            ...
        return self.summary_dict
```

https://gitlab.cern.ch/hep-benchmarks/hep-benchmark-suite/-/blob/fc84702f7e90a1ce850abdd461fe6aaf9e374a1c/hepbenchmarksuite/plugins/energy_plugin.py

Summary / Outlook

- Finalising results in view of **CHEP2023** (I will "present" a poster there next May)
  https://www.jlab.org/conference/CHEP2023

- Submitted **ACAT2022** paper (I gave an actual talk there last October)
  https://indico.cern.ch/event/1106990/contributions/4991256/

- Testing the **HEP-Score** suite 2023 and result submission:
  - successful test on **arm** & **x86**, but having issue on a dual socket DELL workernode (**2*x86**)
    (apparently, running 128+ threads as non-root user hits the system's **ulimit**),
  - result submission is a little tedious (2 * Grid pw per every submission), but it works.

- Energy **plug-in** for the HEP-Score suite in development (and ready for testing)

- Upcoming **ARM cluster** at Glasgow:
  … we will soon get ~1k arm64 cores @ Glasgow from Ampere (US), meaning that we will have to solve technical issues related to exposing ARM resources on the Grid. **(Any experience within GridPP?)**

End