Power Efficiency in HEP

(comparing ARM with x86)

Outline

- Motivations and previous work*
- Methodology
  - Available hardware
  - Exporter tools & IPMI validation
- Performance tests with various HEP workloads
  - ATLAS full G4 simulations
  - HEP-Score containers
- Final focus:
  - Open issues
  - Conclusions and future plans

* Previous talk @ **GidPP47**: https://indico.cern.ch/event/1128343/contributions/4787174/
Previous talk @ **HEP WorkShop**: https://indico.cern.ch/event/1170924/contributions/4995836/
Previous talk @ **ACAT2022**: https://indico.cern.ch/event/1106990/contributions/4991256/
D.B. talk @ **WLCG WorkShop**: https://indico.cern.ch/event/1162261/contributions/5134740/

Introduction

- The power consumption of computing is coming under intense scrutiny worldwide, driven both by concerns about the carbon footprint, and by rapidly rising energy costs.
- ARM chips, while widely used in mobile devices due to their power efficiency, are not currently in widespread use as capacity hardware on the Worldwide LHC Computing Grid (WLCG).
- LHC experiments are increasingly able to compile their workloads on the ARM architecture (and … GPUs) to take advantage of various HPC facilities (e.g., ATLAS, CMS).
- To test whether WLCG sites have scenarios where power efficiency can be improved by deploying ARM-based hardware, the energy consumption and execution speed of identical CPU- and RAM-intensive workloads on two almost identical machines were tested: one with an Ampere **arm64** CPU, and the other with a standard AMD **x86_64** CPU.
- The HEP workloads includes full ATLAS simulations and the most recent HEP-Score containerized jobs developed for Run3.

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

Special Thanks: Domenico Giordano (CERN), Gonzalo Menendez Borge (CERN), Johannes Elmsheuser (CERN)

Available Hardware

We have recently purchased two almost identical machines of comparable price, one with an AMD **x86_64** CPU (with HT enabled), the other with an Ampere **arm64** CPU:

x86_64: Single AMD EPYC 7003 series (SuperMicro)

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 Processor (SuperMicro)

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

And we included in the comparison a standard workernode of our Grid cluster, with 2 AMD **x86_64** CPUs (with HT enabled), which is also comparable in price* to the machines above:

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.

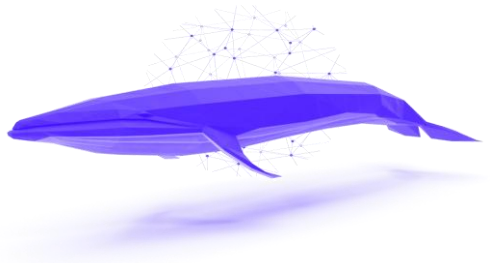
Power Readings

CPU, RAM usage metrics and IPMI power readings are extracted and logged by two custom scripts:

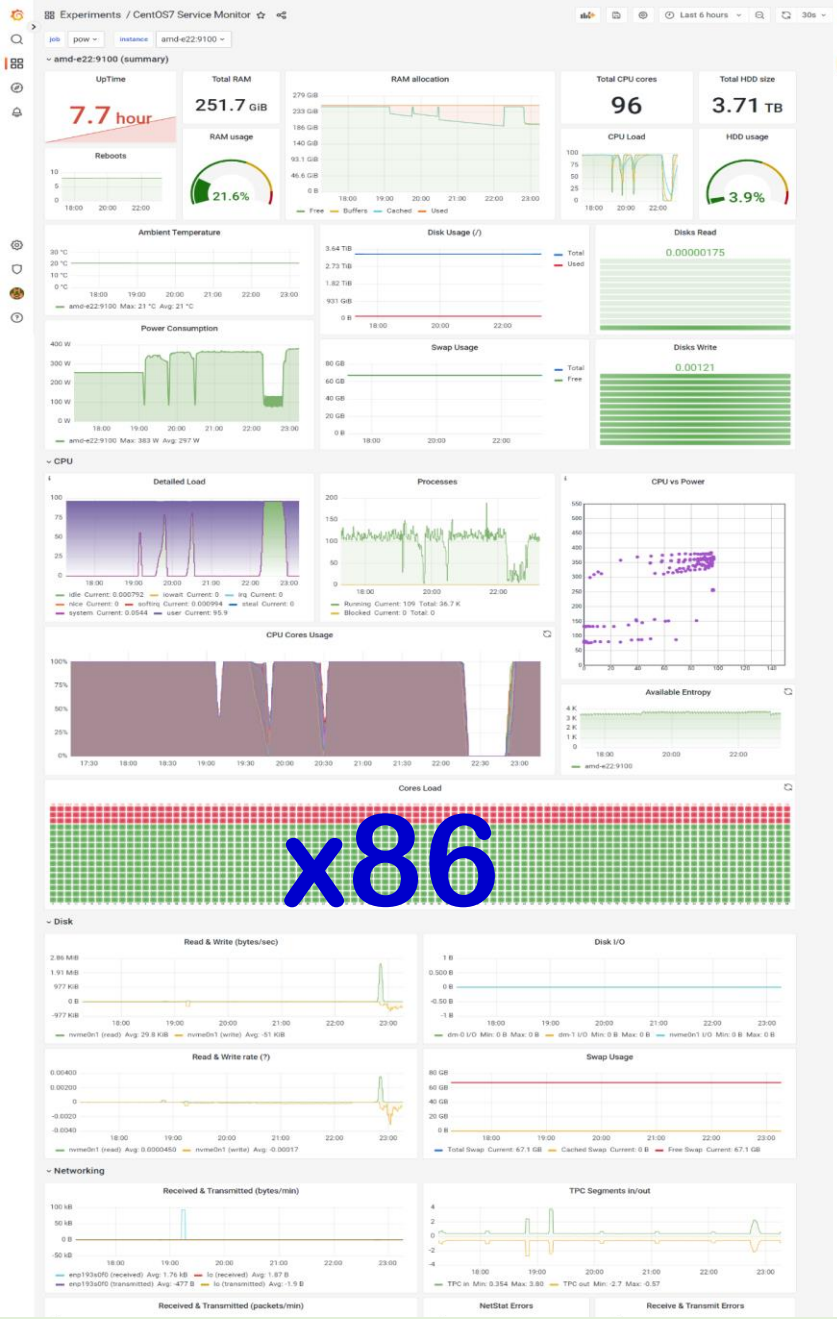
- The 1st script is a cron job (by **root**) that every 10 seconds exports IPMI power readings with a timestamp to `/tmp/ipmidump.txt` (… because *IPMItool* requires **root** privileges).

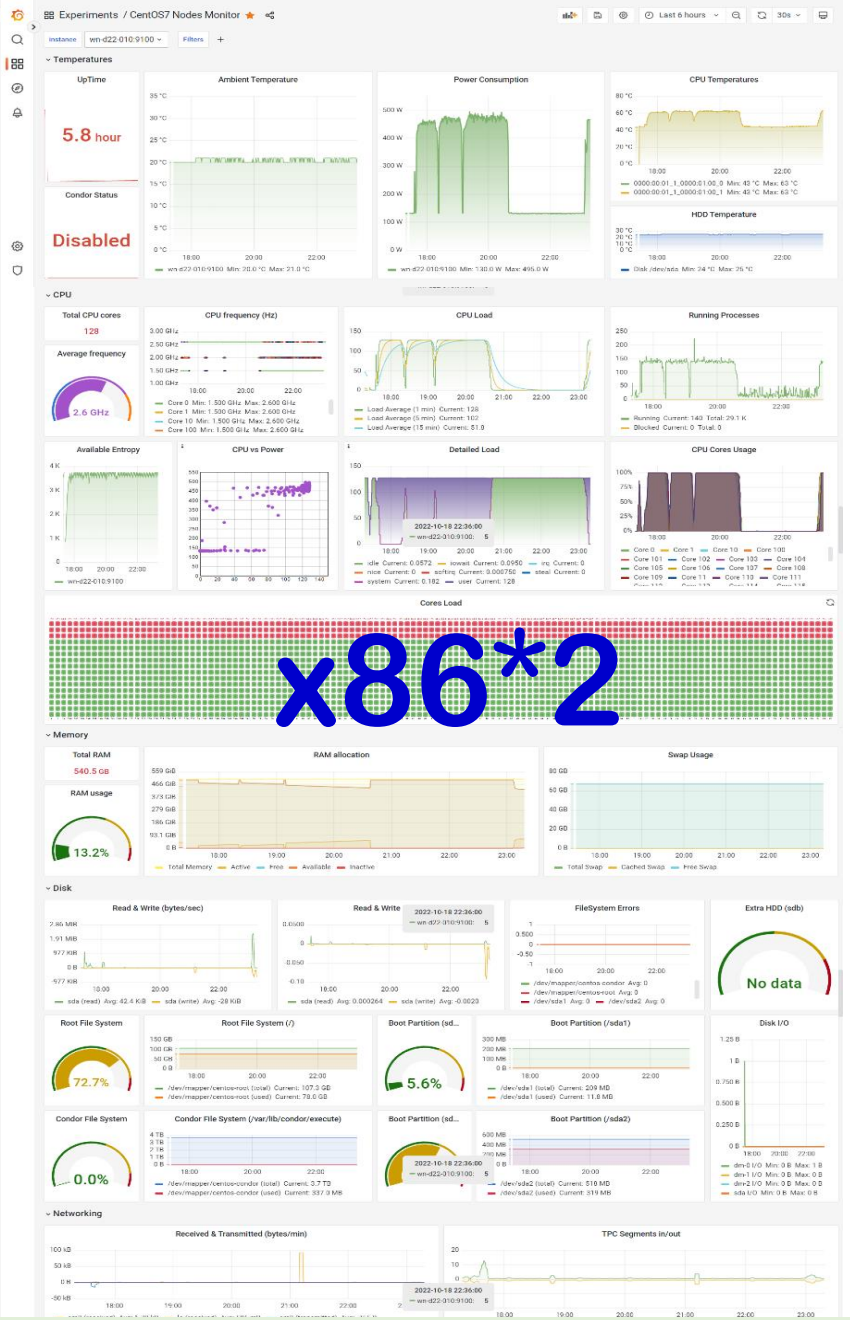
- The 2nd script is executed by the **user** and it takes in the job to be benchmarked as argument. When executed, it starts grabbing the IPMI readings from the dump file, attaches few more info (CPU, RAM) and appends them to a CSV file. After 1 min. sleep (so to measure idle power), it runs the given job, and then waits another min. after the job is finished before quitting.

After the job is done, the CSV file is exported locally and processed in ROOT (time profiles plots and power integration). Cumulative results are then visualized in Excel.



In addition, all servers are running a *node_exporter* client, which feeds (almost) real time metrics to our *Prometheus* server, which in turn feeds an extensive set of *Grafana* dashboards for easy visualization and monitoring purposes.





Site Monitoring @ **vCHEP2021**:
https://indico.cern.ch/event/948465/contributions/4323666/attachments/2248127/3813306/MonitoringAndAutomation_vCHEP2021.pdf

IPMI validation

We have attempted some sort of validation of the IPMI readings using (cheap) metered plugs:

- Instantaneous power was 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 idle and for a complete job - which unfortunately did not have the same duration on both machines.

Idle test (30 min.)

x86: 0.04766 kWh ~ 0.046 kWh (=0.021+0.025 kWh) → Δ = 0.00166 kWh (= -3.5%% of IPMI reading)

arm: 0.05668 kWh ~ 0.054 kWh (=0.024+0.030 kWh) → Δ = 0.00268 kWh (= -4.7% of IPMI reading)

Job test (x86 ~45 min. ; arm ~30 min.)

x86: 0.25729 kWh ~ 0.263 kWh (=0.128+0.135 kWh) → Δ = 0.00571 kWh (= +2.2% of IPMI reading)

arm: 0.13418 kWh ~ 0.134 kWh (=0.064+0.070 kWh) → Δ = 0.00018 kWh (= +0.1% of IPMI reading)

Measurements of energy consumption over a longer duration of at least 30 minutes yielded results which were within **±5%** of the values recorded via IPMI, giving us confidence in the validity of our IPMI results.

Benchmarks

Idle benchmark:

❖ Idle measurement (*sleep*)

HEP benchmarks (typical Grid workload)**:**

❖ Full G4MT ATLAS Simulation (sim-digi)

❖ HEP-Score containers (CMS, ATLAS)

We have created a project in our local GitLab repo to collect test-jobs and benchmarks that we have used in various occasion to run tests in our Tier2 cluster.

TestJobs
Project ID: 12

Star 0 Fork 0

105 Commits 1 Branch 0 Tags 16.6 MB Files 16.6 MB Storage

Collection of tools and script to test job execution on the ARC-Condor cluster or on standalone workernodes for performance measurement. Includes power comparison tools (arm vs. x86).

master testjobs / + History Find file Web IDE Clone

bash sieve
Emanuele Simili authored 2 days ago 4f0e189b

README Auto DevOps enabled Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster

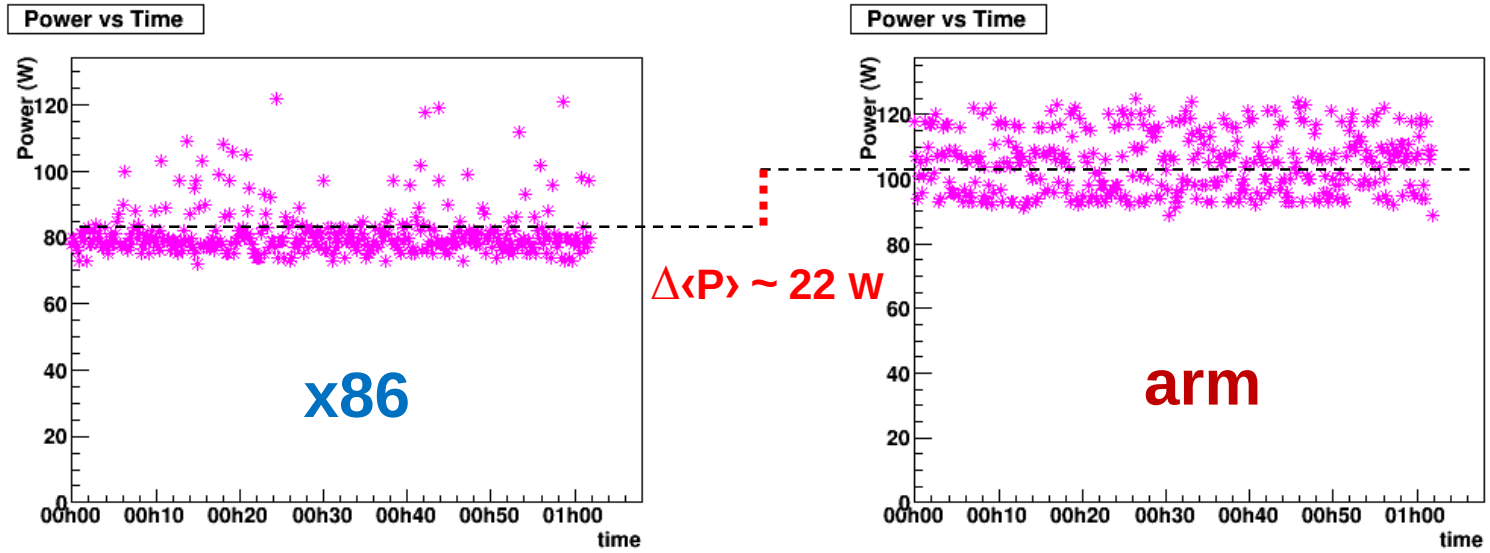
Name	Last commit	Last update
ATLAS	Script Quickfix	1 week ago
HEPscore	HEPscore containers pre ACAT2022	5 days ago
HS06	clean ATLAS ...	7 months ago
ROOT	updates	6 days ago
arcsub	Refreshed repo: initial commit	7 months ago
bash	bash sieve	2 days ago
comp	grrr	5 days ago
exporter	updates	6 days ago
python	...	2 weeks ago
README.md	Updated timestamp of README.md as test.	1 month ago

Idle Power

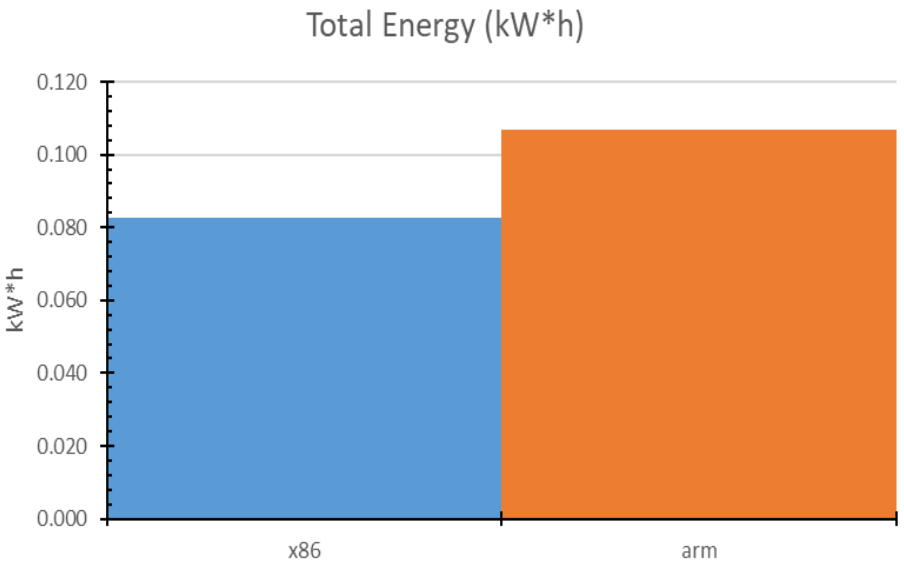
For this measurement we just let the machines idle for 1h, while collecting power metrics. In order to use the IPMI exporter script, we set to execute a *sleep* job:

```
$ sleep 3600
```

Power profiles show that the **arm** has a higher baseline, and larger oscillations between power states, leading to a higher average *



Machine	threads	Time (s)	Time (h)	Tot. Energy	Peak Power (W)	Idle (W)
x86	96	3600	01:00:00	0.083	122.0	81.8
arm	80	3600	01:00:00	0.107	125.0	103.8



lower = better

Key result: the **arm** uses about 30% more energy than the **x86** in idle state, making I/O bound tasks slightly less power efficient on **arm** than on an equivalent **x86** server.

* Note: the two CPUs have a different number of physical cores !

ATLAS Workload

The chosen version of the software: **Athena 23.0.3**
(builds available for both **x86_64** and **aarch64** on CVMFS)

x86
```
Using Athena/23.0.3 [cmake] with platform x86_64-centos7-gcc11-opt at /cvmfs/atlas.cern.ch/repo/sw/software/23.0
```

arm
```
Using Athena/23.0.3 [cmake] with platform aarch64-centos7-gcc11-opt at /cvmfs/atlas.cern.ch/repo/sw/software/23.0
```

Setting up the ATLAS framework on CVMFS and running the job:

```
$ export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
$ alias setupATLAS='source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh'
$ setupATLAS

$ asetup Athena,23.0.3
$ ./TTbarSim2022.sh
```

Input file:

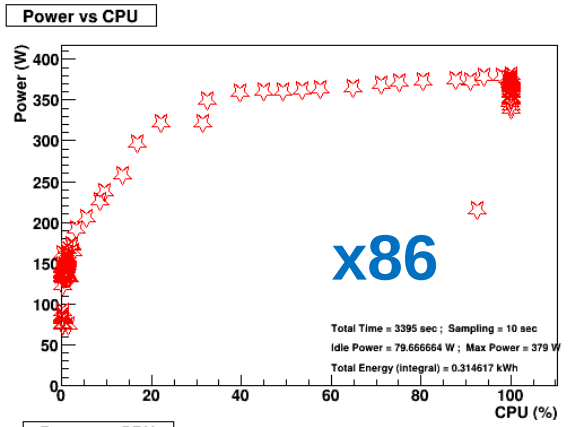
Geant4 MT full ATLAS detector simulation of a given number of TTbar events (from existing TTbar gen-events file)

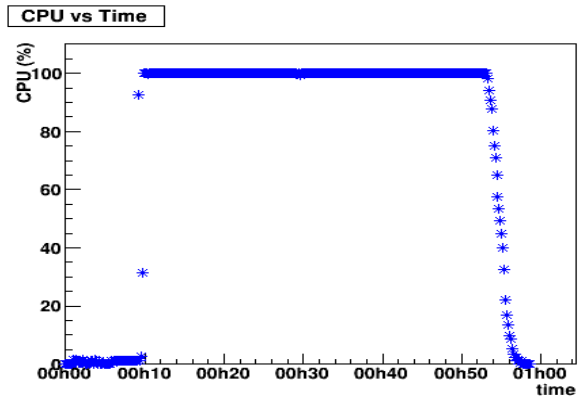
We generated samples of 1k and 10k events …

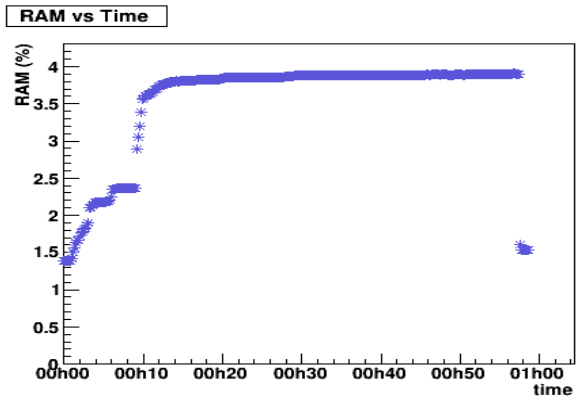
TTbarSim2022.sh

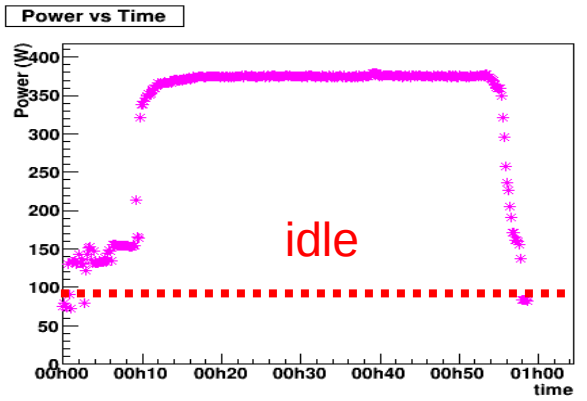
```
#!/bin/sh

export MAXEVENTS=10000
export ATHENA_CORE_NUMBER=$(nproc)
inputdatadir=/cvmfs/atlas.cern.ch/repo/benchmarks/hep-workloads/input-data
inputdata=$inputdatadir/EVNT.13043099._000859.pool.root.1

Sim_tf.py \
      --inputEVNTFile="${inputdata}" \
      --outputHITSFile="TTbar2022.HITS.pool.root" \
      --maxEvents=${MAXEVENTS} \
      --physicsList=FTFP_BERT_ATL \
      --imf=False \
      --randomSeed=6163 \
      --AMIConfig=s3873 \
      --multithreaded=True \
      --jobNumber=1 \
```

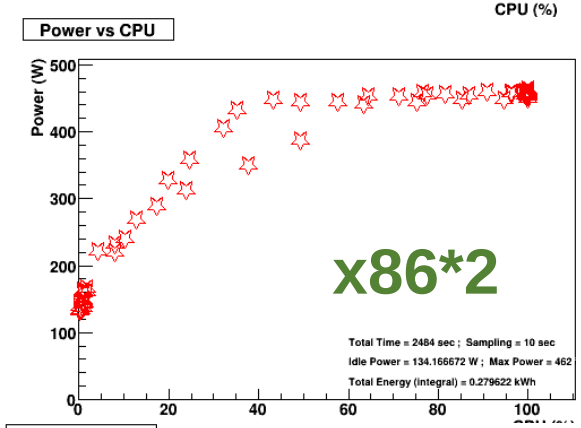
Job Profiles (ATLAS 1k)

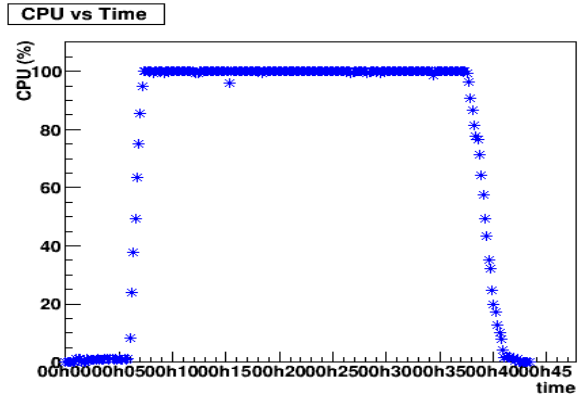


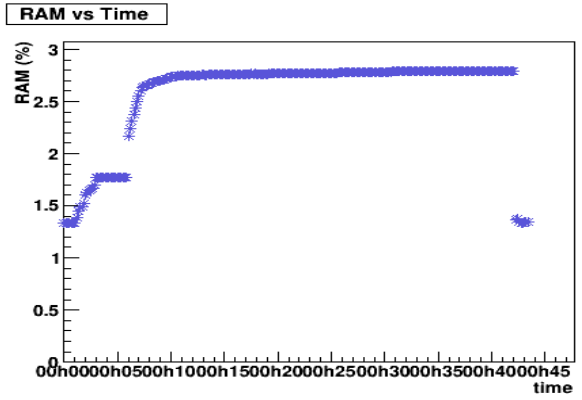


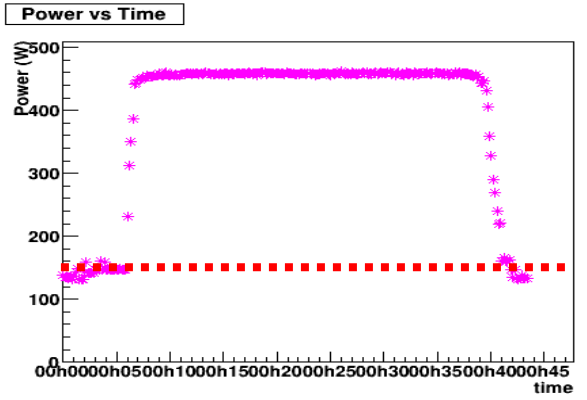


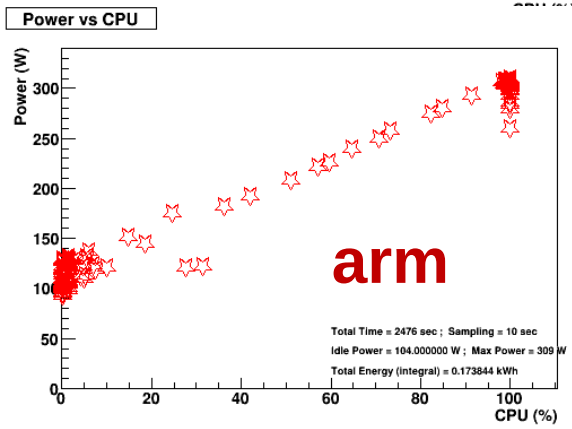


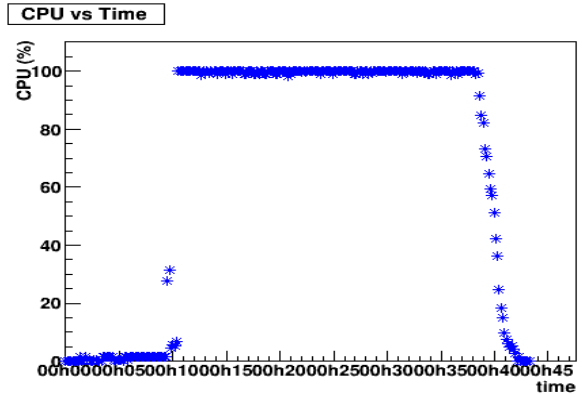


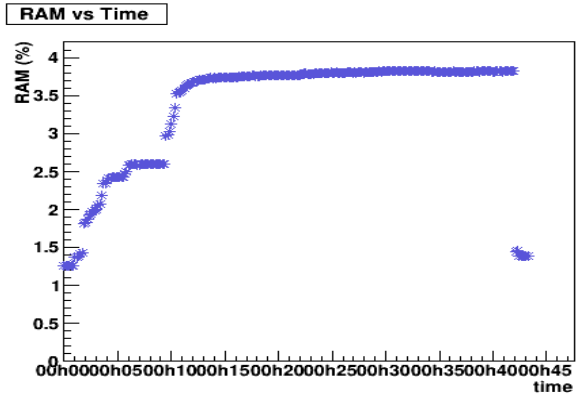


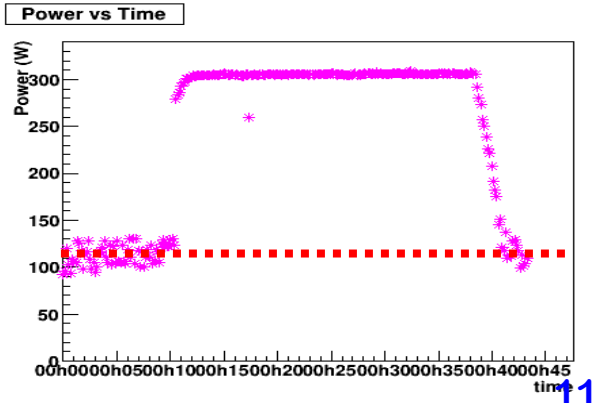










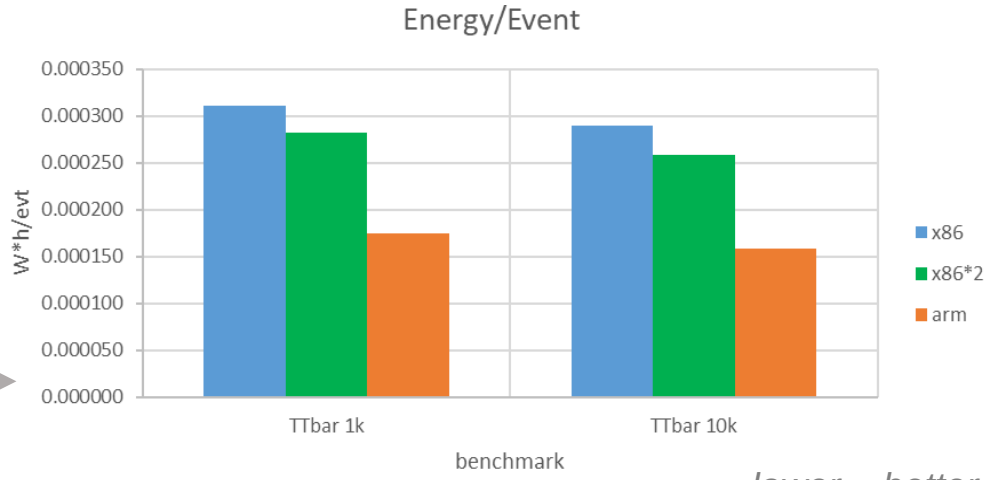


Results (ATLAS 10k)

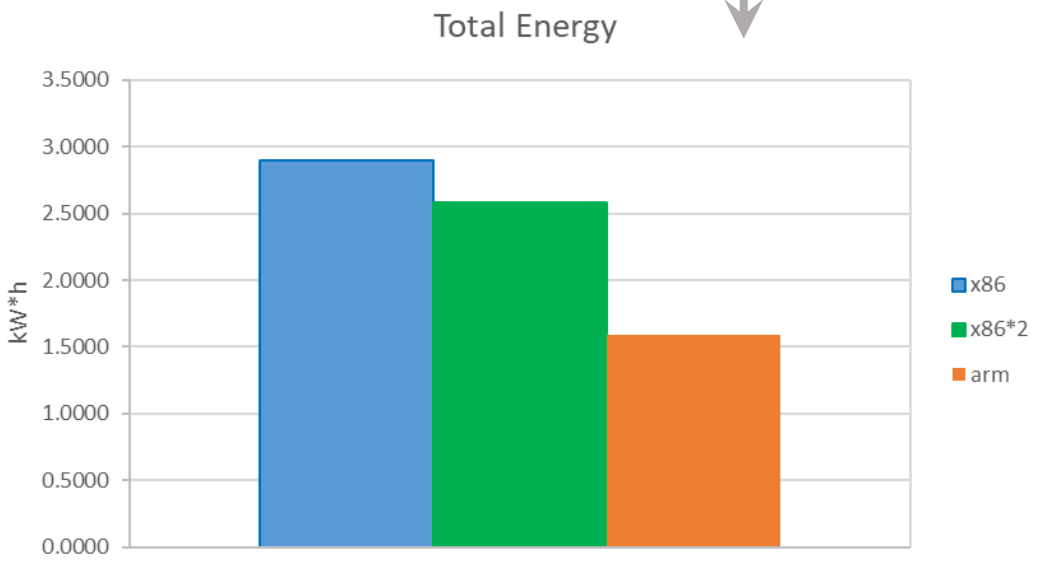
We compared execution time and integrated energy consumption over the job duration for the 3 types of machines available (**x86**, **x86*2** and **arm**). Each job was executed three times, we took the average execution time and energy consumption, and their standard deviation as an estimate of the uncertainty.

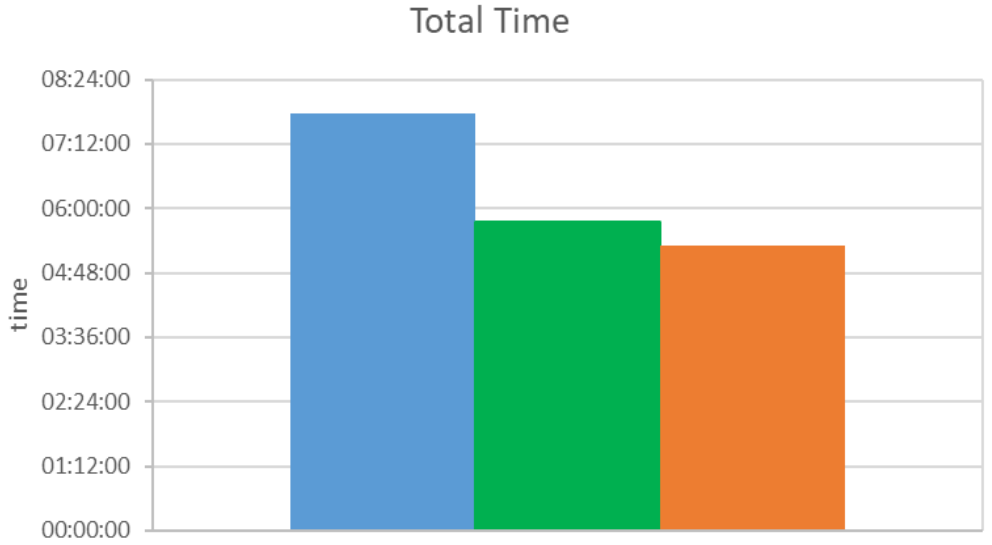
Arch	Threads	ATLAS sim.	N. events	Time (h)	dT (%)	Energy(kW*h)	dE (%)	idle(W)	max(W)	W*h/event
x86	96	TTbar	10'000	07:46:14	0.3%	2.8966	0.6%	85	382	0.000290
x86*2	128	TTbar	10'000	05:44:25	0.4%	2.5843	0.6%	133	463	0.000258
arm	80	TTbar	10'000	05:19:07	2.2%	1.5853	2.5%	110	309	0.000159

The better efficiency of the **arm** machine compared to both the **x86** can be seen in the **Total Energy** plot or in the **Energy/Event** plot.



lower = better





Key result:
the **arm** is slightly faster, and much more energy efficient than **x86** !

lower = better

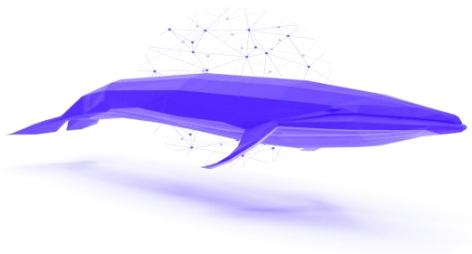
HEP-Score

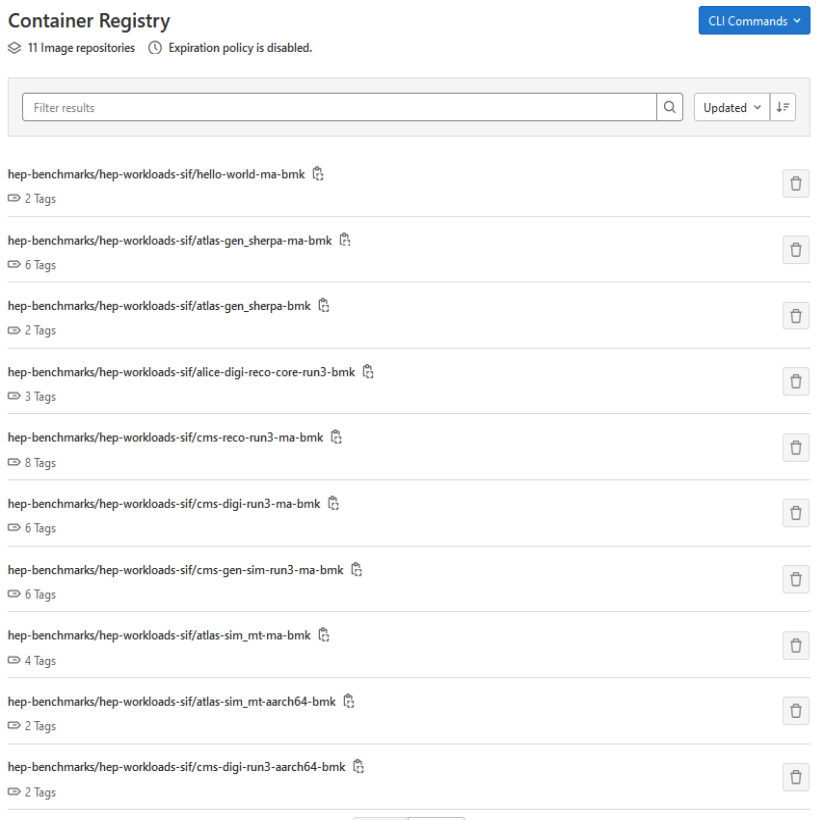
I started interacting with the HEP-Score Task Force earlier this year, because of mutual interest and partial overlap with my research on power efficiency. In particular:

- I have been using the available HEP-Score containers as a standard HEP workload to rate different architectures on power efficiency (which is becoming increasingly important for procurement),
- I helped testing HEP-Score containers on our locally available architectures (**x86** & **arm**),
- We discussed the option to include power readings in the standard output of the HEP-Score suite (peak / idle power & integrated energy consumption for a given workload).

While the full HEP benchmark suite was not yet available for **arm**, an increasing number of experimental workloads were available as standalone containers

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





HEP-Score Containers

HEP-Score containers can run on **Singularity** (or **Docker**, which we do not use):

(x86) *Singularity-CE 3.9.9-1.el7* (previous version *3.8.7-1.el7* disappeared from EPEL and replaced with *AppTainer 1.1.0-1.el7*)

(arm) *singularity version 3.8.4-1.el7* (still available in EPEL)

Example execution of a containerised HEP-Score job:

```
$ mkdir –p /tmp/test/results
$ chmod a+rw /tmp/test/
$ singularity run -B /tmp/test:/results oras://registry.cern.ch/hep-workloads/cms-gen-sim-run3-bmk:latest
```

We used 5 HEP-Score containers from the container_registry:

gitlab-registry.cern.ch/hep-benchmarks/hep-workloads-sif/atlas-sim_mt-ma-bmk:v2.0
gitlab-registry.cern.ch/hep-benchmarks/hep-workloads-sif/atlas-gen_sherpa-ma-bmk:ci-v1.0

} ATLAS (2x)

gitlab-registry.cern.ch/hep-benchmarks/hep-workloads-sif/cms-reco-run3-ma-bmk:v1.1
gitlab-registry.cern.ch/hep-benchmarks/hep-workloads-sif/cms-digi-run3-ma-bmk:v1.0
gitlab-registry.cern.ch/hep-benchmarks/hep-workloads-sif/cms-gen-sim-run3-ma-bmk:v1.0

} CMS (3x)

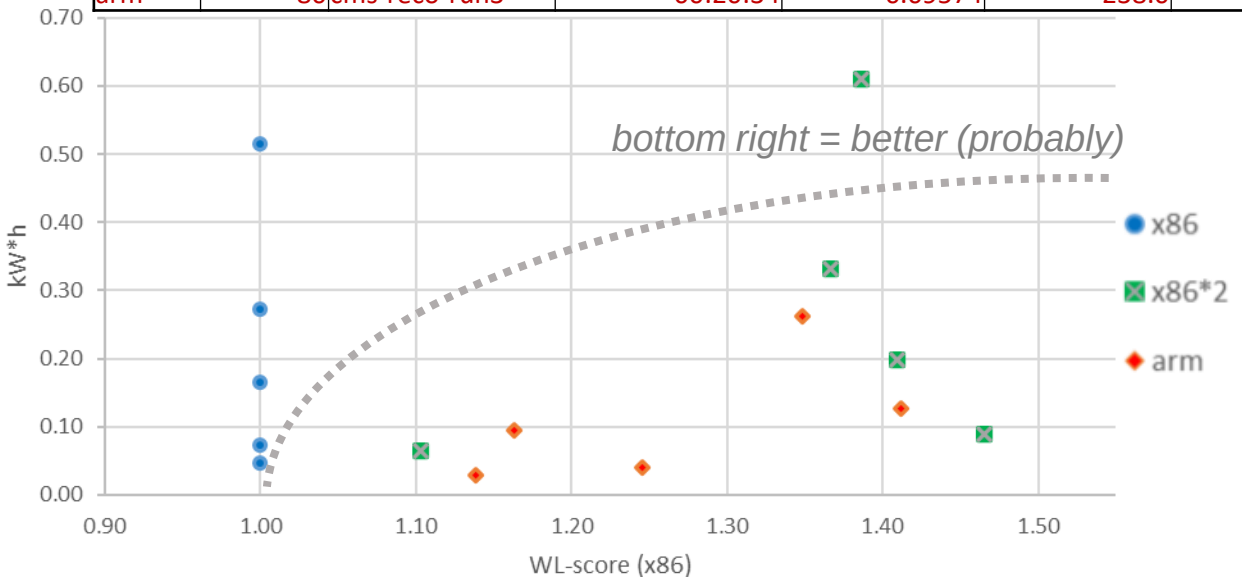
Note: the HEP workloads are designed to scale with the number of available threads, therefore power consumption cannot be directly compared among machines with a different number of cores (/threads), as the machine with more threads would have done more work …

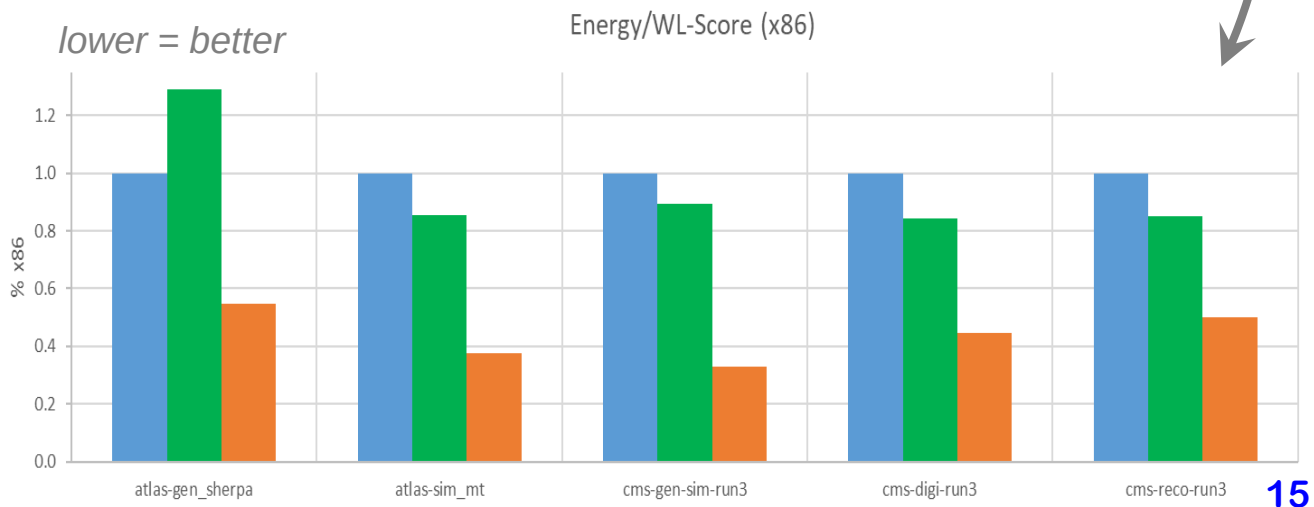
HEP-Score Results

Here the cumulative results of the 5 HEP-Score containers on the 3 machines (**x86**, **x86*2** and **arm**):

Arch	Threads	Benchmark	Time (hh:mm:ss)	Energy (kW*h)	RAM max (Gb)	idle (W)	max (W)	WL-Score	WL-Score[#] (wrt x86)	E/WLs (kW*h)	E/WLs[#] (wrt *x86*)
x86	96	atlas-gen_sherpa	00:08:21	0.04576	154.8	94	390	113.3959	100.0%	0.0004	100.0%
x86	96	atlas-sim_mt	01:23:49	0.51478	54.4	82	383	0.4068	100.0%	1.2655	100.0%
x86	96	cms-gen-sim-run3	00:44:13	0.27235	31.3	98	385	3.7347	100.0%	0.0729	100.0%
x86	96	cms-digi-run3	00:12:31	0.07224	102.1	88	392	15.0538	100.0%	0.0048	100.0%
x86	96	cms-reco-run3	00:26:53	0.16447	119.6	86	389	6.4072	100.0%	0.0257	100.0%
x86*2	128	atlas-gen_sherpa	00:10:11	0.06509	207.9	133	488	125.0605	110.3%	0.0005	129.0%
x86*2	128	atlas-sim_mt	01:20:58	0.6109	74.0	133	473	0.5639	138.6%	1.0834	85.6%
x86*2	128	cms-gen-sim-run3	00:43:14	0.3319	43.2	134	477	5.1025	136.6%	0.0650	89.2%
x86*2	128	cms-digi-run3	00:12:41	0.08937	138.2	133	494	22.0662	146.6%	0.0041	84.4%
x86*2	128	cms-reco-run3	00:26:03	0.19734	160.9	134	483	9.0284	140.9%	0.0219	85.2%
arm	80	atlas-gen_sherpa	00:06:21	0.02856	126.4	108	348	129.0706	113.8%	0.0002	54.8%
arm	80	atlas-sim_mt	00:56:30	0.26156	64.1	104	310	0.5486	134.9%	0.4768	37.7%
arm	80	cms-gen-sim-run3	00:26:44	0.12691	148.8	116	306	5.2721	141.2%	0.0241	33.0%
arm	80	cms-digi-run3	00:09:01	0.04008	183.9	104	318	18.7561	124.6%	0.0021	44.5%
arm	80	cms-reco-run3	00:20:34	0.09574	238.0	102	310	7.4504	116.3%	0.0129	50.1%

[#] *normalised to **x86***

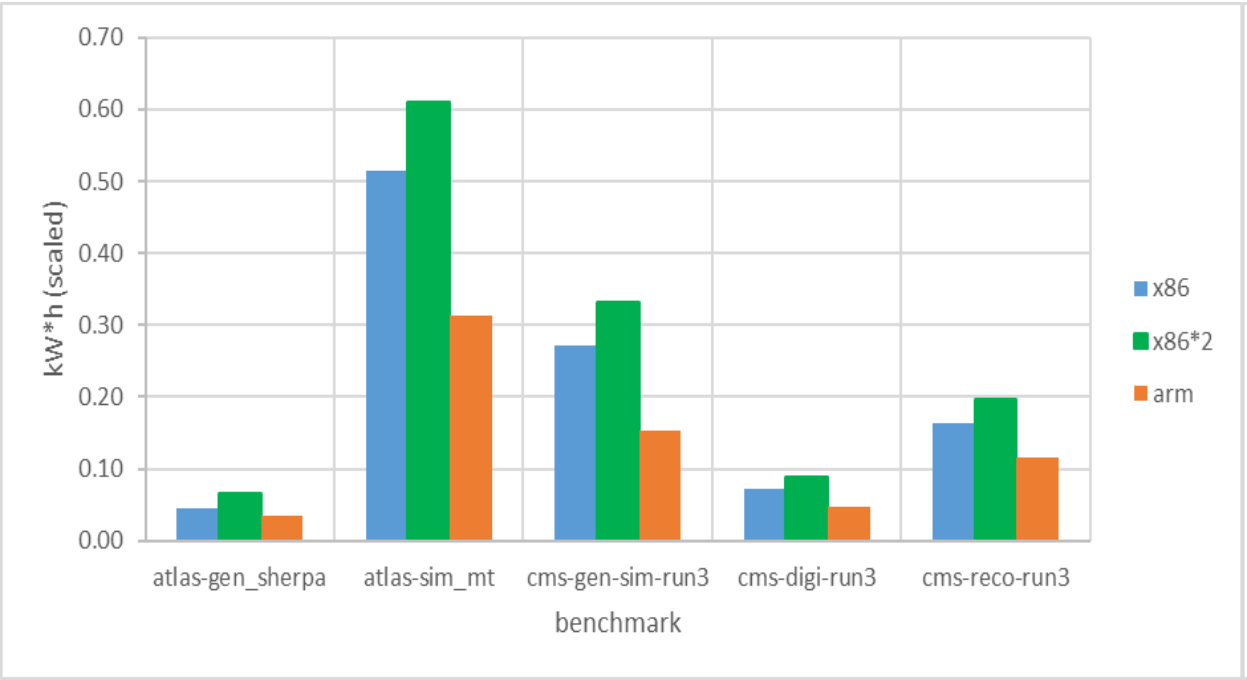


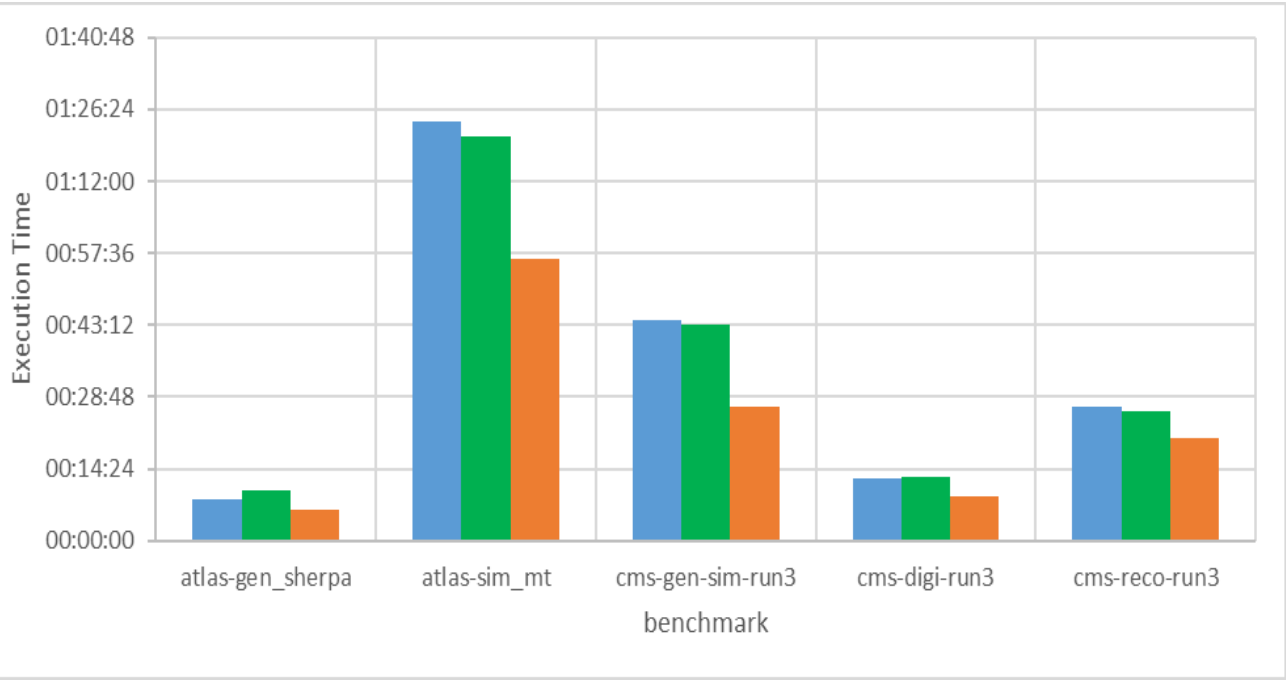


HEP-Score Summary

In conclusion, we compared an **arm** and **x86** processor of very similar specs and almost identical cost. We found that on average, the **arm** processor ran ~20% quicker and used ~35% less power per HEP task than the equivalent **x86** *

* *averages renormalized to the same amount of work (**x86** = 1)*





Note:

while the HEP-Score geometric average is not yet available (as we did run standalone containers outside the HEP Benchmarking suite), we struggled to find an intuitive way to compare the machines …

Open Issues

- Despite the active development within all LHC experiments, support for architectures other than **x86** is patchy, at best
- …
- Same is true for hardware suppliers, at least in our experience: our *DELL* supplier does not sell any **arm** server (yet). We ordered it from a small local supplier and it took over 2 months to arrive …
- The **arm** machine seem to score better in speed and energy efficiency, but we never really looked under the hood: e.g., physics output was not validated.
- We trust enough our IPMI power readings (± 5%), but we may want a more precise comparison (e.g., by using a metered PDU that allows remote power readings from each plug).
- There was no error propagation. We just took the 5% error on IPMI reading as the major source …
- All results in this PPT were presented at ACAT last month, and they were a bit rushed due to the delay in the **arm** server delivery. The next iteration of this study is already ongoing.

Summary & Outlook

- This study addressed almost all the limitation that affected of our previous one on power efficiency (ref. *GridPP47*), as the benchmark were now performed on two almost identical servers installed locally in a closely controlled environment.

- In almost all benchmarking tests, we see that for the same price **arm architecture gives better performance in term of speed and power efficiency**.
  We also see that our **arm** server has a higher energy consumption in its idle state than its **x86** counterpart, which makes I/O bound operations slightly less power efficient.

- We wish to continue this study by performing a better estimate of all sources of errors, and by doing some sort of validation of the jobs output - to guarantee that the better performance does not come at the cost of a lower accuracy.

- We also wish to extend our set of benchmark to GPUs. We already have some hardware available …

Thanks.

Exporter Script

TimeStamp: `date +"%F , %T"`
CPU: `top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1}'`
RAM: `free -t | awk 'FNR == 2 {printf("%.2f"), $3/$2*100}'`
GPU: `nvidia-smi --query-gpu=power.draw --format=csv,noheader,nounits | awk '{s+=$1} END {print s}'`
Power (IPMI): `ipmitool dcmi power reading | grep "Instantaneous power reading:"`

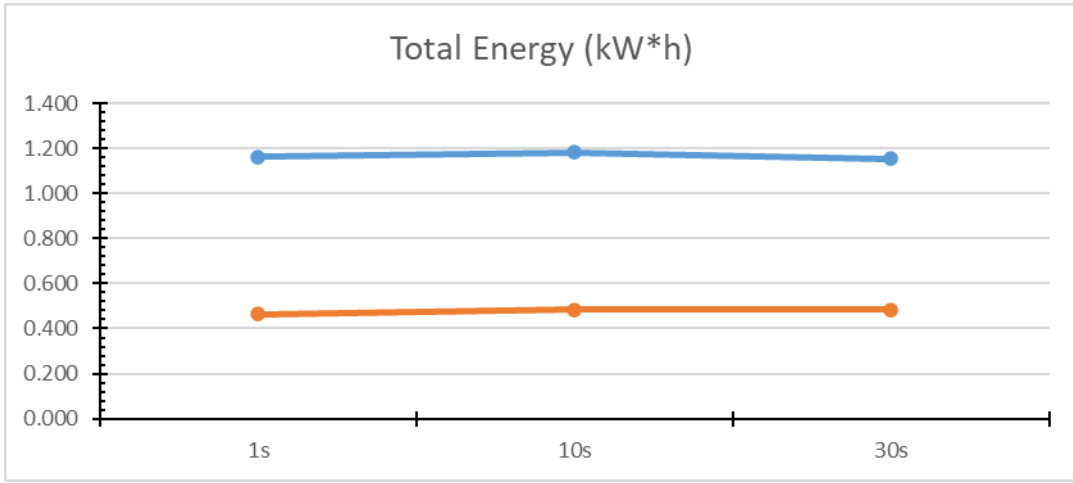
Sampling Frequency

Check that the sampling frequency we chose does not affect the integrated Energy consumption (the effect is less than 2%).

Prime Number sieve (1 to 100M) using compiled C code with OMP (gcc version 4.8.5 20150623)
IPMI sampling interval = 1sec , 10sec,
30sec

Job	Sampling	threads	Time (s)	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)
x86	1s	96	10982	03:03:02	1.162	402.0	90.9
	10s	96	11002	03:03:22	1.183	402.0	84.0
	30s	96	11017	03:03:37	1.154	391.0	85.5
arm	1s	80	8259	02:17:39	0.465	215.0	103.9
	10s	80	8229	02:17:09	0.483	215.0	104.5
	30s	80	8229	02:17:09	0.485	220.0	100.5

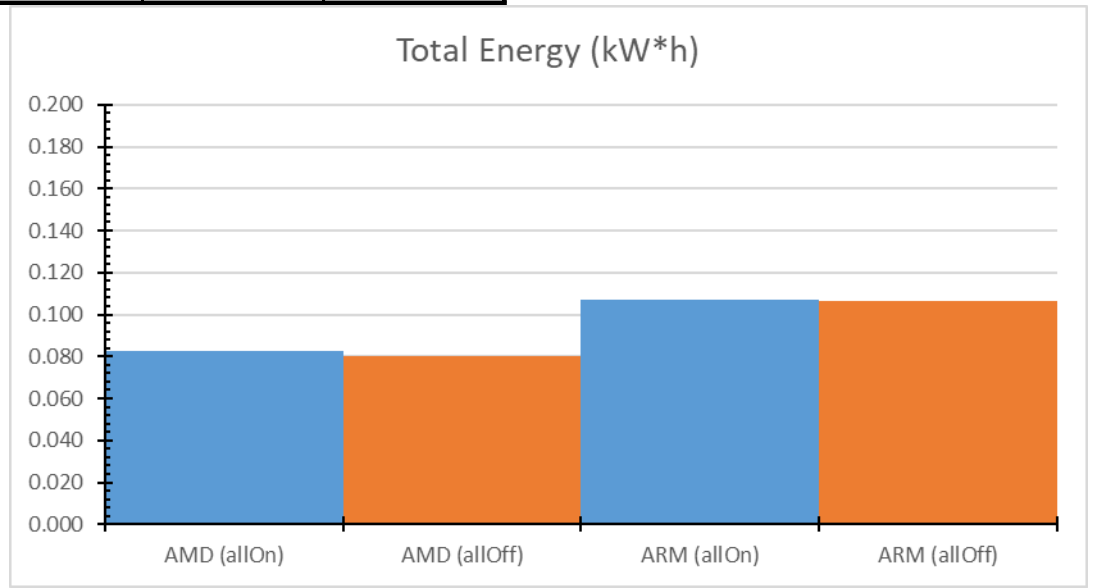
dE (x86)	0.0123	1.1%
dE (arm)	0.0087	1.8%



Exporters Effect

Check that the active exporters (*node_exporter*, *PromTail*, *Pakiti*) do not add extra power consumption.
(the effect seems to be about 2% … but it would be better to gather more samples)

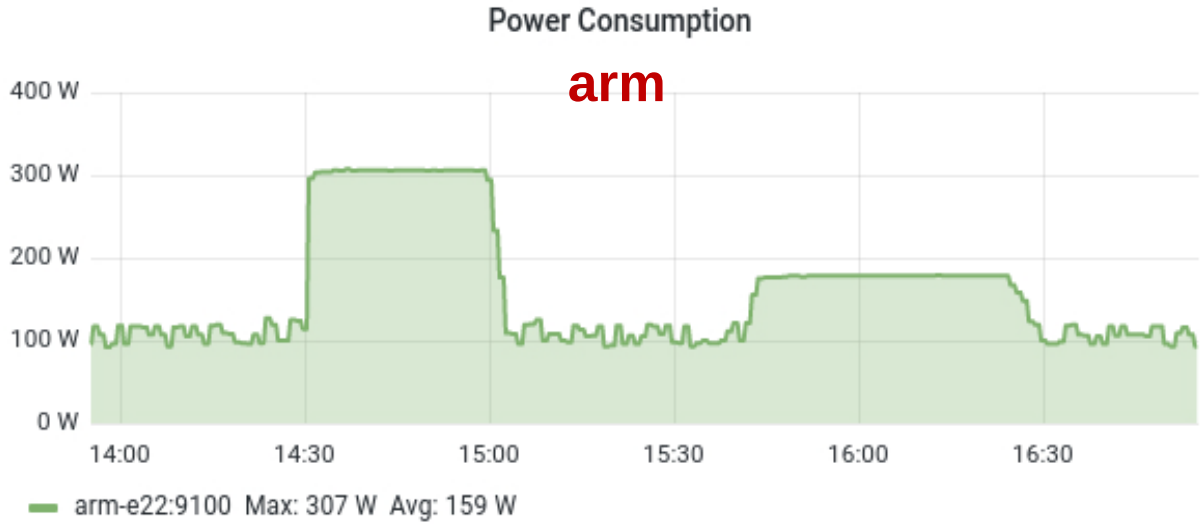
Idle Job (1h) with/without exporters (node_exporter, PromTail, Pakiti, cron)							
IPMI sampling interval = 10sec							
Job	**Exporters**	**threads**	**Time (s)**	**Time (h)**	**Tot. Energ**	**Peak Pow**	**Idle (W)**
AMD	AMD (allOn)	96	3600	01:00:00	0.083	122.0	81.8
	AMD (allOff)	96	3600	01:00:00	0.080	137.0	76.3
ARM	ARM (allOn)	80	3600	01:00:00	0.107	125.0	103.8
	ARM (allOff)	80	3600	01:00:00	0.107	120.0	105.2

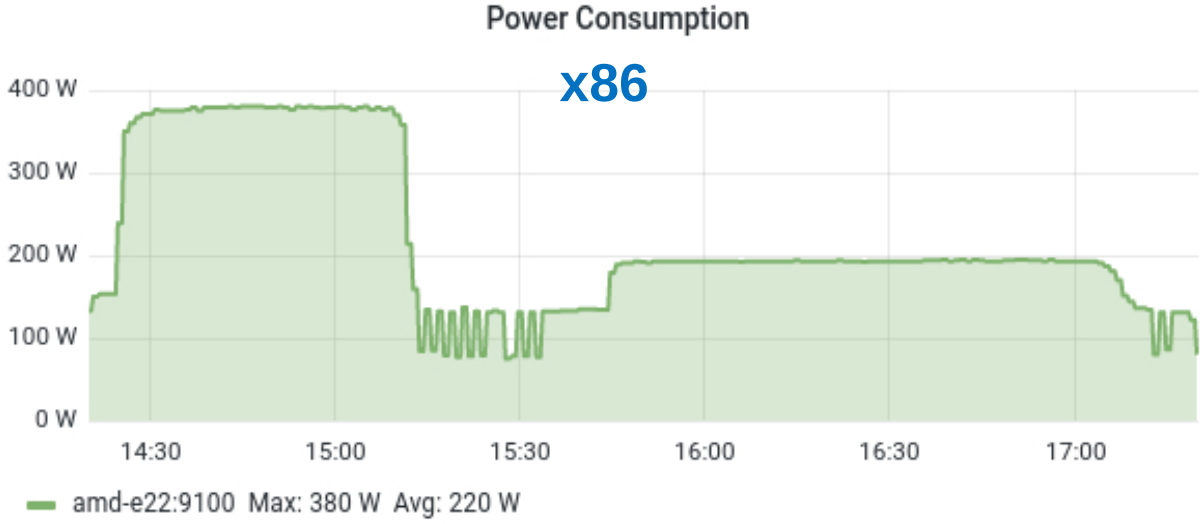


CPU throttling

Following the suggestions in Rod's PPT, we tried to run the ATLAS load at different CPU frequencies … (the goal is to reduce the electricity usage over peak hours, while also not killing jobs)

$ sudo cpupower frequency-set -g performance # conservative # powersave
$ cpupower frequency-info --> current CPU frequency: 2.60 GHz (asserted by call to kernel)
$ nohup ./getStartA.sh &





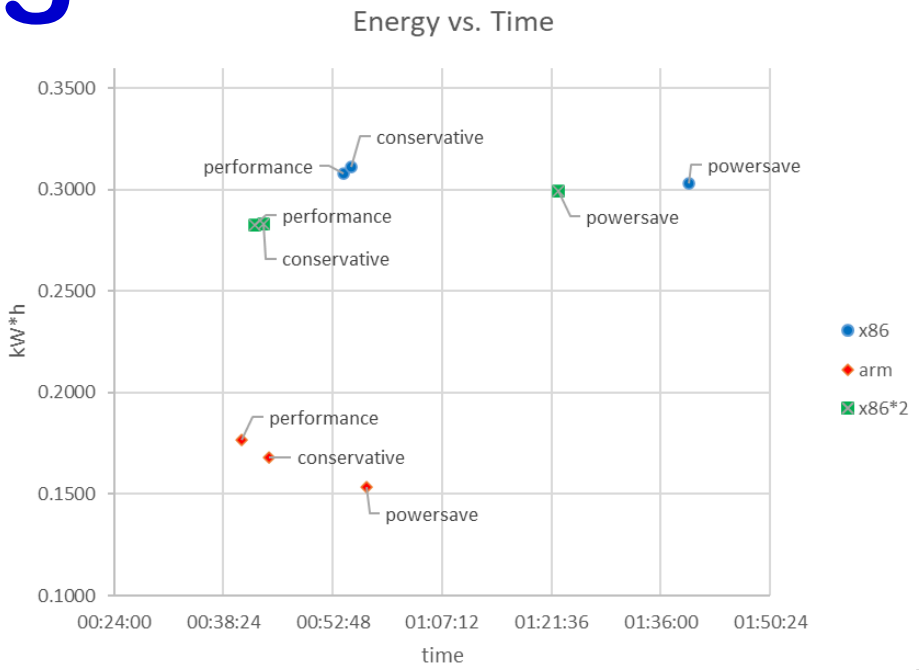
Running at lower clock speed saves some power (not much) at the expenses of a longer execution time (almost twice on the x86, less on the ARM). There isn't much difference between 'performance' and 'conservative' because the frequency scales up to MAX as soon as the job starts.

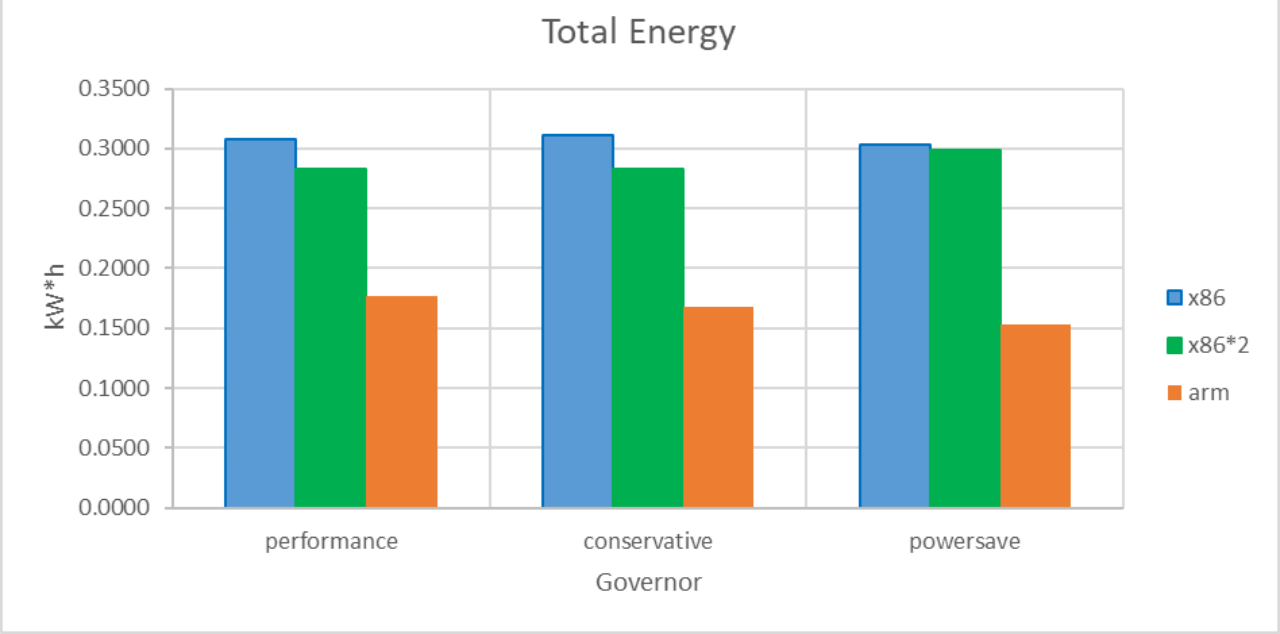
new

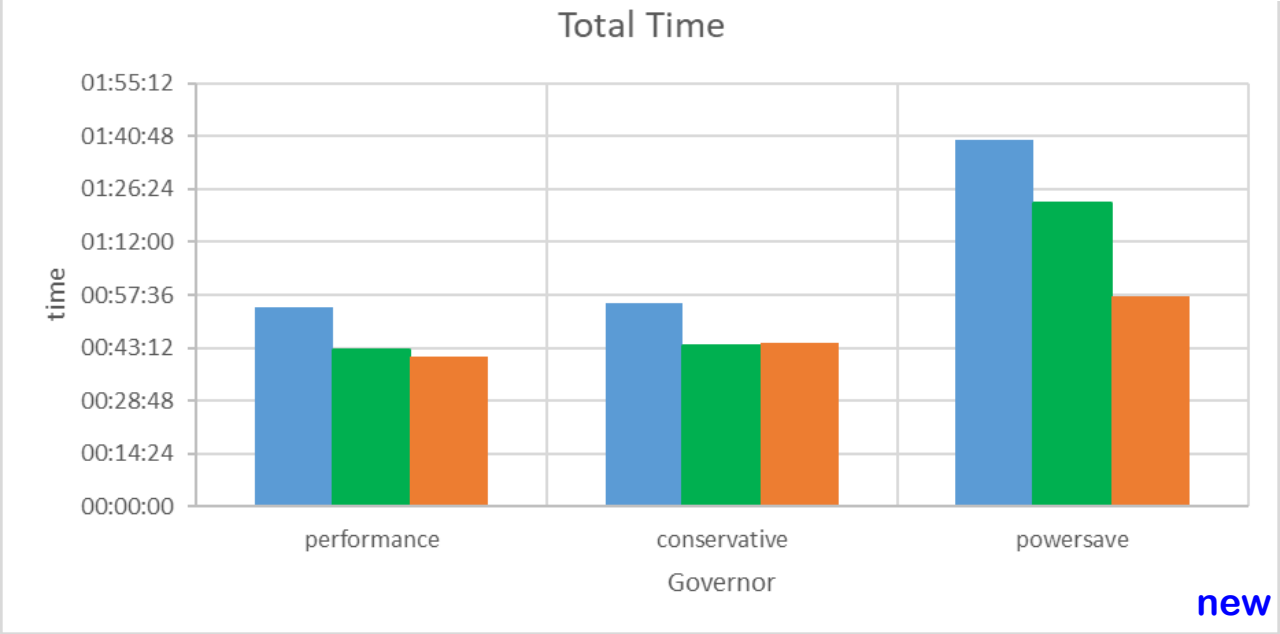
CPU throttling

Here the execution time and integrated energy consumption from the 3 machine considered, with the 3 different Governor settings:

Arch	Threads	Governor	Freequency (GHz)	N. events	Time (h)	Energy(kW*h)	idle(W)	max(W)	W*h/event
x86	96	performance	2.30	1'000	00:54:13	0.3077	85	381	0.000310
x86	96	conservative	1.50 - 2.30	1'000	00:55:15	0.3114	92	381	0.000174
x86	96	powersave	1.50	1'000	01:39:46	0.3029	90	195	0.000170
x86*2	128	performance	2.60	1'000	00:42:32	0.2827	134	464	0.000181
x86*2	128	conservative	1.50 - 2.60	1'000	00:43:44	0.2828	138	462	0.000175
x86*2	128	powersave	1.50	1'000	01:22:34	0.2991	130	247	0.000280
arm	80	performance	3.00	1'000	00:40:47	0.1766	102	311	0.000282
arm	80	conservative	2.00 - 3.00	1'000	00:44:29	0.1681	107	307	0.000283
arm	80	powersave	2.00	1'000	00:57:20	0.1536	98	179	0.000281







new