On Power Efficiency: x86 vs. ARM

Outline

- Comparing **x86_64** vs **arm64** architectures
- Exporter tools & visualization
- Obvious Limitations of our approach
- Benchmarks (BASH, C, ATLAS)
- Results
- Conclusions

ScotGrid Glasgow:

Emanuele Simili, Gordon Stewart, Samuel Skipsey, David Britton

Power Comparison

Leveraging the hardware available at Glasgow and Leicester, we have compared the power efficiency and execution speed of different architectures under similar loads. The study was limited in scope, it only involved 2 remote **arm64** machines and 2 local **x86_64** machines of different generations.

Various benchmarks have been executed and several job profiles were collected (memory, CPU, power). The benchmarks included a BASH script, a few compiled C programs and two different types of ATLAS simulations (where possible).

No substantial differences in power consumption were found among the architectures under exam, despite these being expected from different generations of processors.

Results are far from conclusive. Moreover, the whole study suffered from some obvious limitations **(*)**, which must be addressed in the future if such project should continue.

Special thanks to: Davide Costanzo (Sheffield), Oana Boeriu (CERN), Johannes Elmsheuser (CERN), Jon Wakelin (Leicester) and the *Excalibur Project*

Available Hardware

DELL Epyc (Glasgow)
DELL PowerEdge C6525
2 * 32cores Epyc *x86_64* CPUs & 512GB RAM *(128 threads)* ★

HP Xeon (Glasgow)
HP ProLiant DL60 Gen9
2 * 10cores Xeon *x86_64* CPUs and 156GB RAM *(40 threads)*

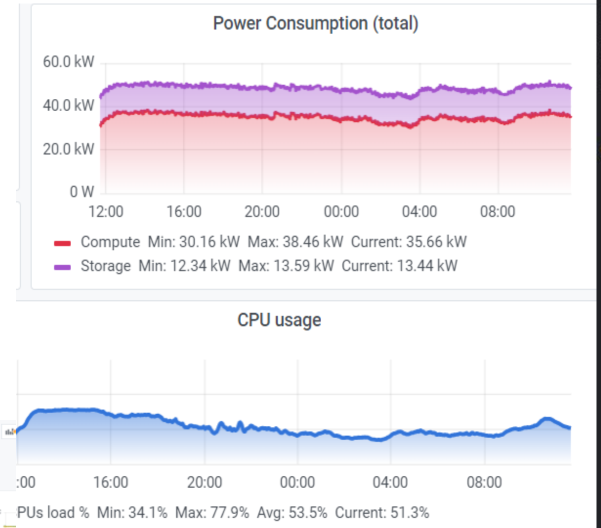
ARM+GPU (Leicester)
Ampere Q80
Neoverse-N1 *arm64* CPU & 512GB RAM *(80 threads)* ★
+ 2 * Nvidia A100 (40GB) GPUs

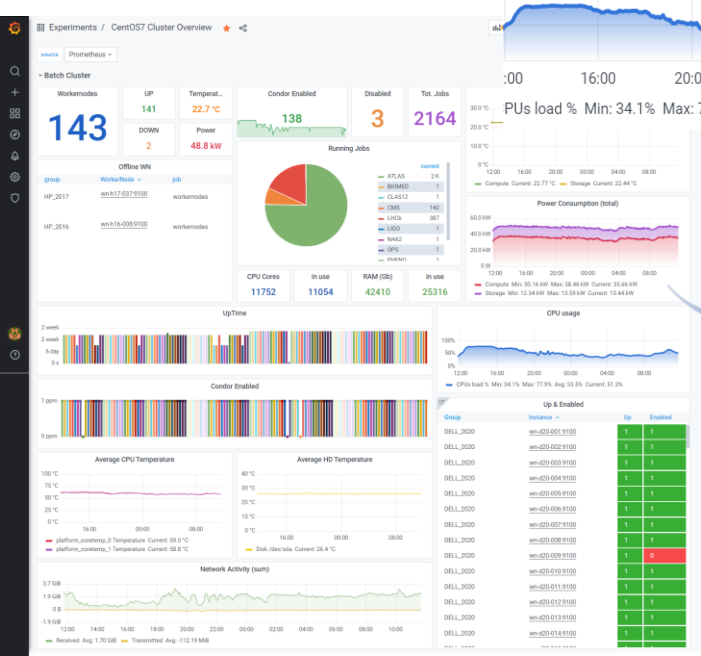
ARM (Leicester)
HPE CN99XX
2 * 112cores *arm64* Cavium ThunderX2 CPUs & 256GB RAM *(224 threads)*

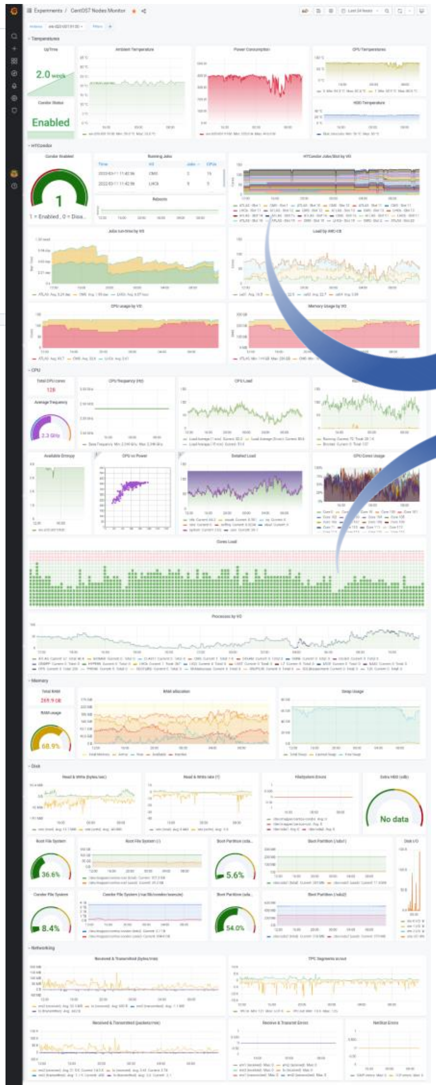
Visualization (local)

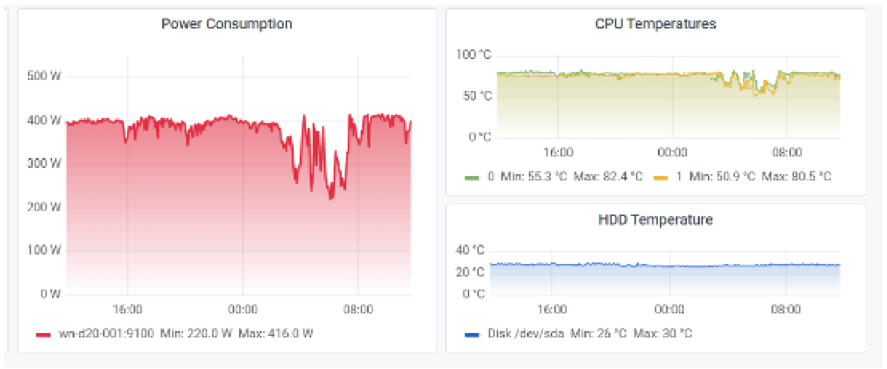
Local power readings and resource usage (at Glasgow) is colourfully visualised in our Grafana dashboards:

Cluster Overview

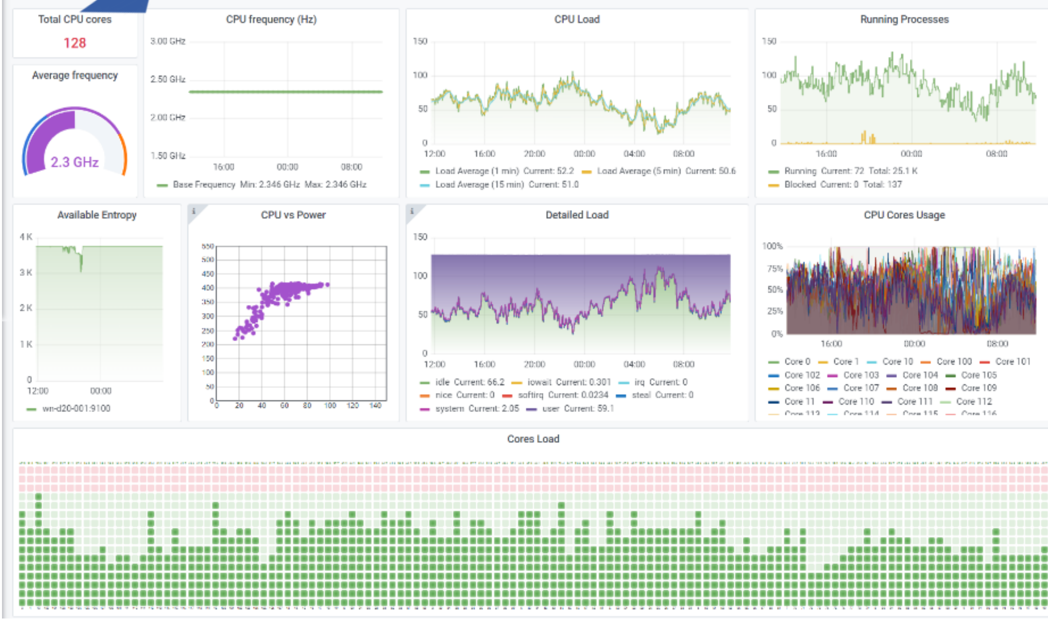








WorkerNode View



Power Readings (remote)

Remote readings (at Leicester) are achieved by two custom scripts, that collect and export metrics such as CPU, RAM and Power Usage:

1) Every 30 seconds, a cron job (**root**) exports IPMI power reading with timestamp to `/tmp/ipmidump.txt` (… because *IPMItool* requires **root** privileges).
   The Cavium ARM machine has a custom Kernel module (*tx2mon* *****) in place of IPMI.

2) Before starting the job, I (**user**) run a background script that grabs these IPMI readings, attaches more info (CPU, RAM) and appends them to a CSV file.
   On the ARM+GPU machine I use also *nvidia-smi* to grab the GPUs' power usage.

When the job is done, the CSV file is exported to Excel for analysis and visualization.
So … most of the analysis was painfully done in *MS Excel* ☹

The exact same apparatus has been installed on both remote and local machines, in order to get the same type of data for an easier comparison.

(*) https://github.com/Marvell-SPBU/tx2mon

Limitations

(*) There were obvious limitations to this approach, which relied on resources at two different geographical locations: local **x86** machines (Glasgow) & remote **arm** machines at Leicester.

- In Glasgow I have full sys-admin control, at Leicester I am just a normal user with limited privileges. Therefore, I have limited control over the activity of these remote machines, and indeed we could see different levels of usage depending on the day/time.

- Machines have very different hardware, in particular the ARM+GPU node at Leicester that can run ATLAS has 2 powerful **Nvidia Ampere GPUs** just sitting idle.

- Power measurements are mainly read by *IPMI tools* interacting with different hardware, and in one case with a custom tool (*tx2mon* on Cavium). So, readings might not be directly comparable, and there is no validation of such data (e.g., an external power-meter).

- The ARM+GPU machine is accessed through *Slurm* (which adds an extra layer).
  I can send my job requests with the flag *--exclusive*, which gives me some sort of priority.

- Cooling is totally neglected: more watts go in, more heat comes out. This might be an important factor for the overall power required by the data center (including coolers).

Benchmark Jobs

Since ATLAS could run only on one of the two ARM machines available, various other benchmarks were attempted. In all cases we made use of multithreading:

- Prime number sieve (BASH script): prime numbers up to 1 M, tot. 78,498
- Prime number sieve (C with OMP): prime numbers up to 100 M, tot. 5,761,455
- Large Matrix Multiplication (C with OMP): 20k x 20k random matrix (*int* & *float* *)
- Full G4MT ATLAS Simulation (TTbar & Charginos): 1k and 10k events

		x86		arm64	
		DELL PowerEdge C6525	HP ProLiant DL60 Gen9	HPE CN99XX	Ampere Q80
bash	Prime Number Sieve (bash)				
compiled	Prime Number Sieve (C + OMP)				
	Large Matrix Multiplication (int)				
	Large Matrix Multiplication (float)				
AthSimulation	ATLAS TTbar (1k events)		▨	■	
	ATLAS TTbar (10k events)		▨	■	
	ATLAS Decaying Charginos (10k events)		▨	■	
	ATLAS whatever (25k events)	▨	▨	■	▨

(*) actually *double*

BASH Sieve

Eratostene’s prime numbers sieve in a BASH script, to find primes up to 1 M

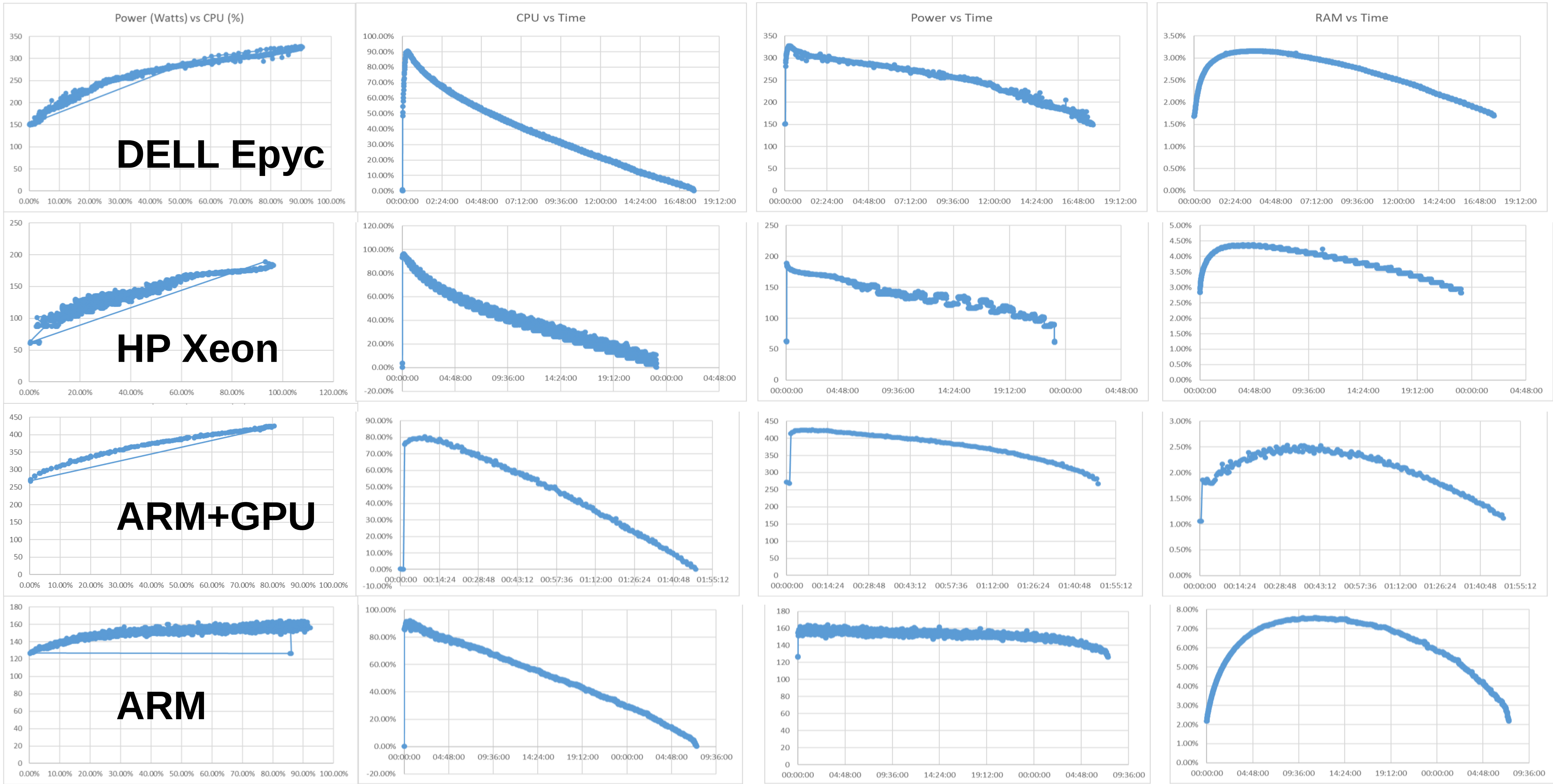
- Script inspired by RosettaCode **(*)** with added multithreading
- It is executed from command line while collecting metrics

```
$ ./get2IPMI.sh > ipminfo.csv

$ ./multiSieve.sh
```

```
#!/bin/bash
...

(( chunk = max / cores ))
...

while (( max > num )); do
    FILE="${FILENAME}${filenum}.txt"
    (( prev = num + 1 ))
    (( num = prev + chunk ))
...
    ./sieve.sh ${prev} ${nmx} > ${FILE} &
done
```

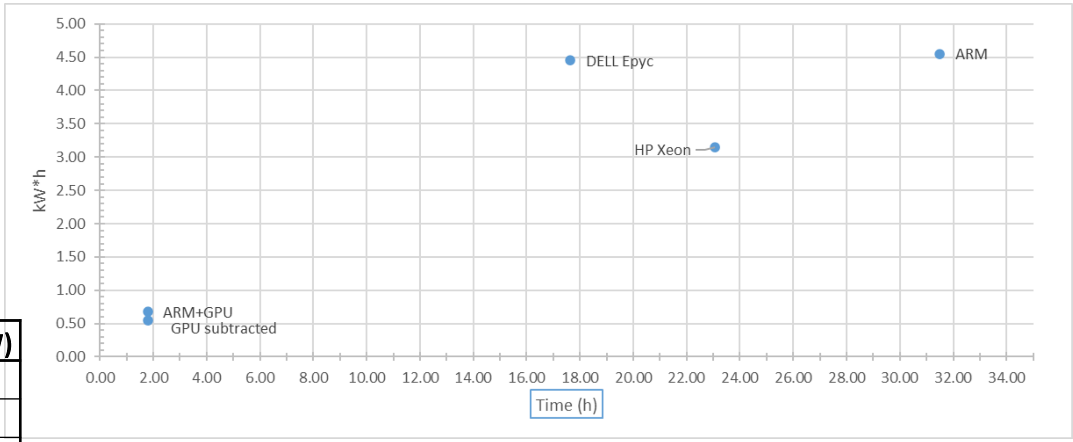
(*) https://rosettacode.org/wiki/Sieve_of_Eratosthenes#UNIX_Shell

Job Profiles (bash sieve)



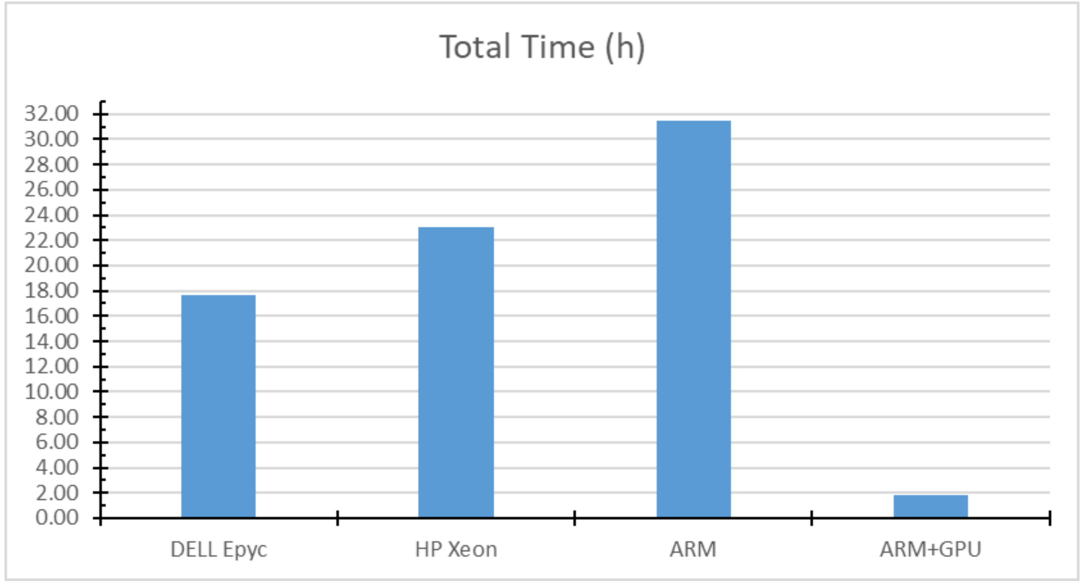
Results (bash sieve)

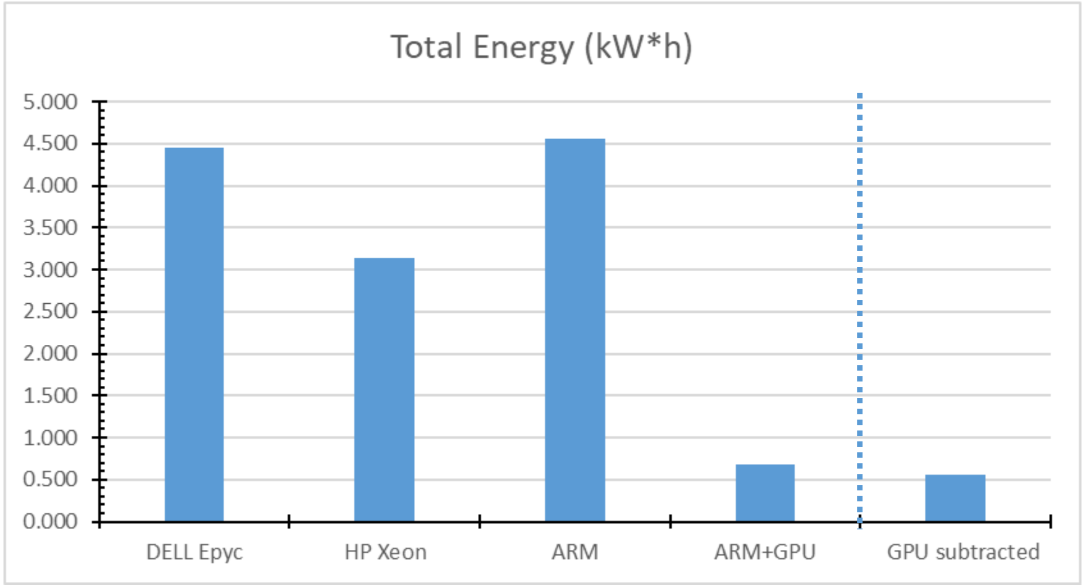
Looks like the **ARM+GPU** machine is the best at finding prime numbers in BASH …



Prime Numbers (bash)

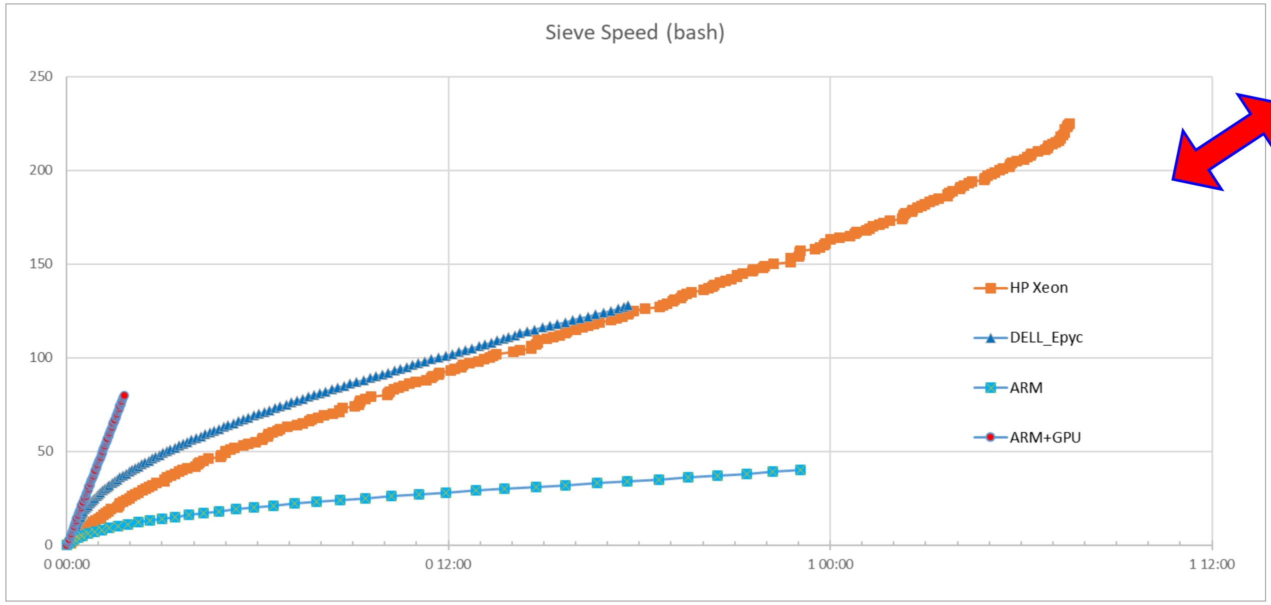
Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)
DELL Epyc	128	17.63	4.450	328	152	
HP Xeon	40	23.08	3.142	189	63	
ARM	224	31.48	4.551	164	127	
ARM+GPU	80	1.80	0.678	425	268	71
GPU subtracted		1.80	0.551	354	197	0

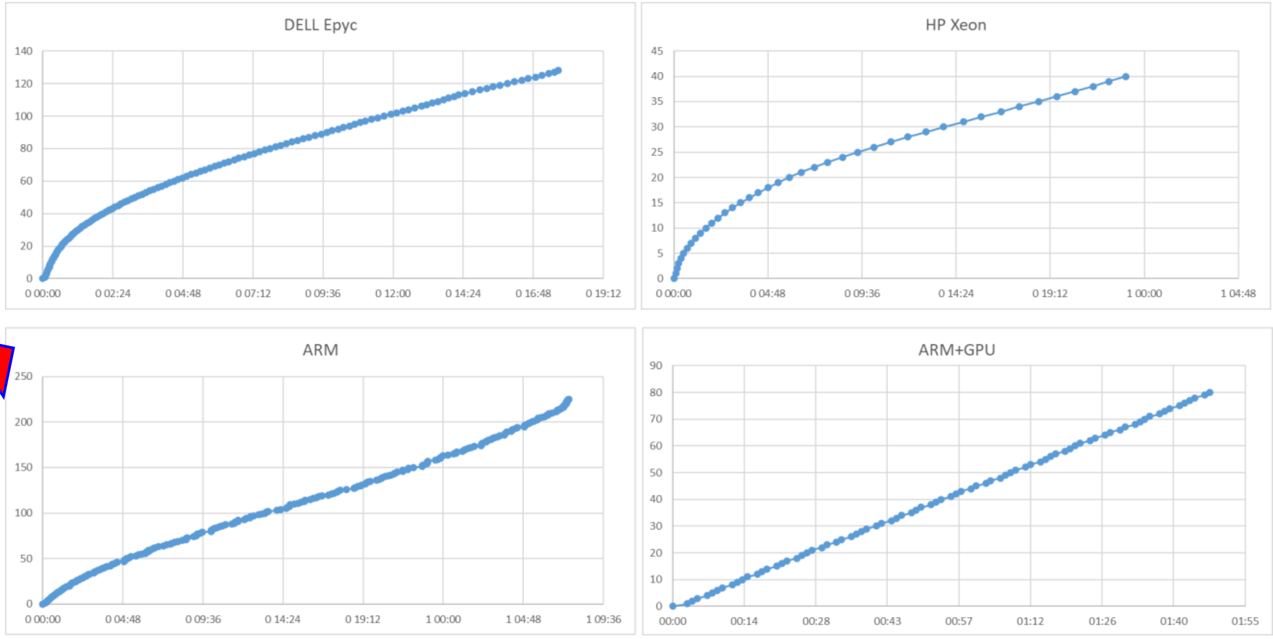


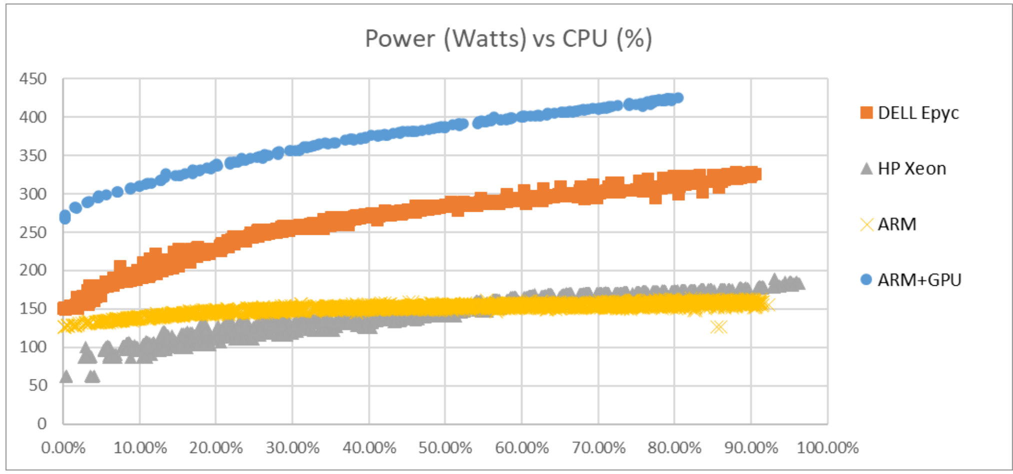


Speed and Power

Other interesting results of this run include the speed at which primes number are found (e.g., **ARM+GPU** is linear!)







Overlapping the power profiles, we can see how the power draw increases with CPU activity (e.g., **ARM** is almost flat)

C Sieve

Eratostene’s prime number sieve in C compiled with OMP to find primes up to 100 M

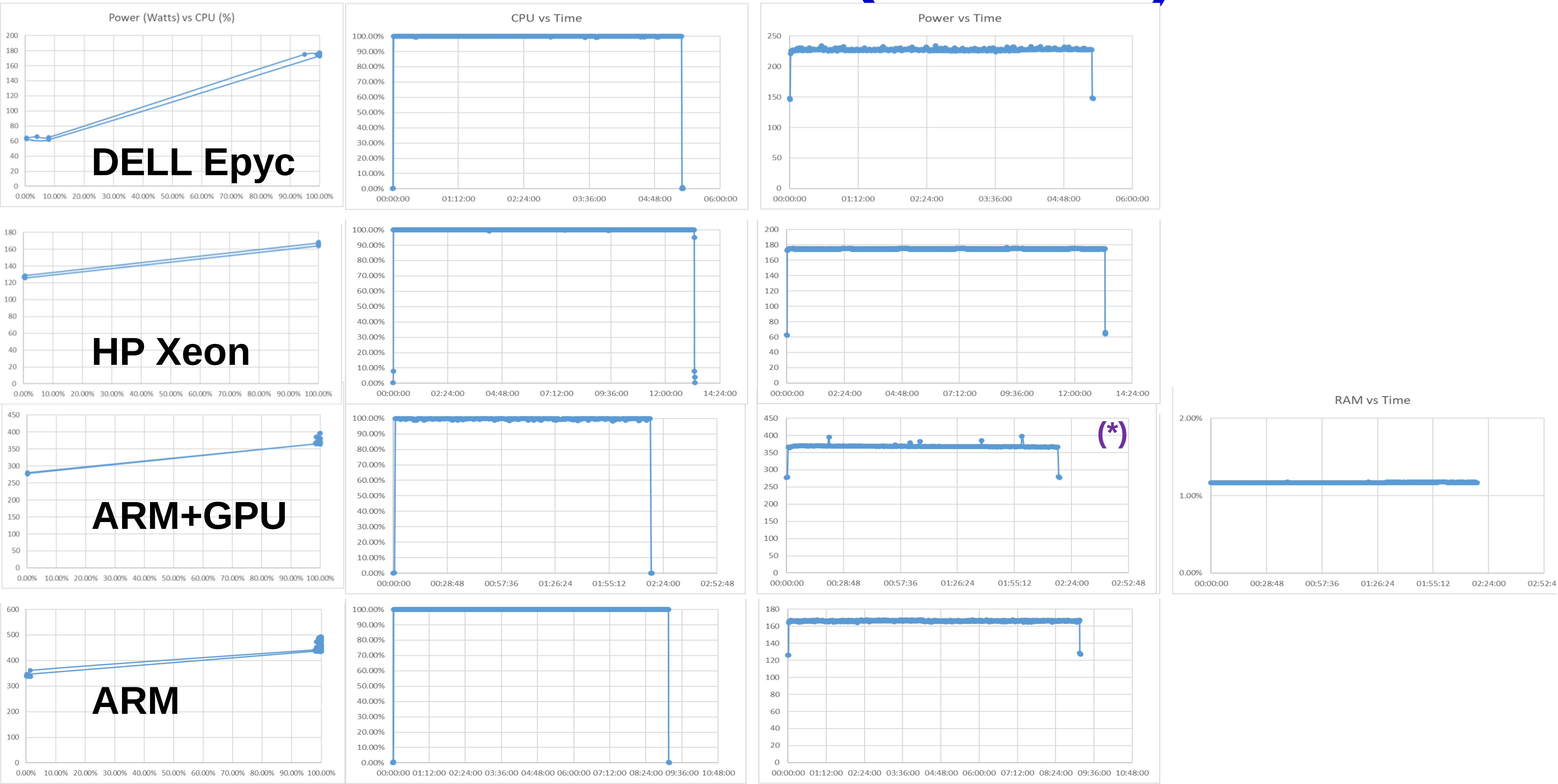
- Code from the web (*)
- Compiled on each machine with **gcc** and the OMP library
- It is executed while collecting metrics

```
$ ./get2IPMI.sh > ipminfo.csv

$  gcc -fopenmp mtprime.c

$ ./a.out
```

```
#include <omp.h>
...

#pragma omp parallel for schedule(dynamic) reduction(+ : primes)
    for (num = 1; num <= limit; num++)
    {
...
```

(*) https://stackoverflow.com/questions/9244481/how-to-get-100-cpu-usage-from-a-c-program

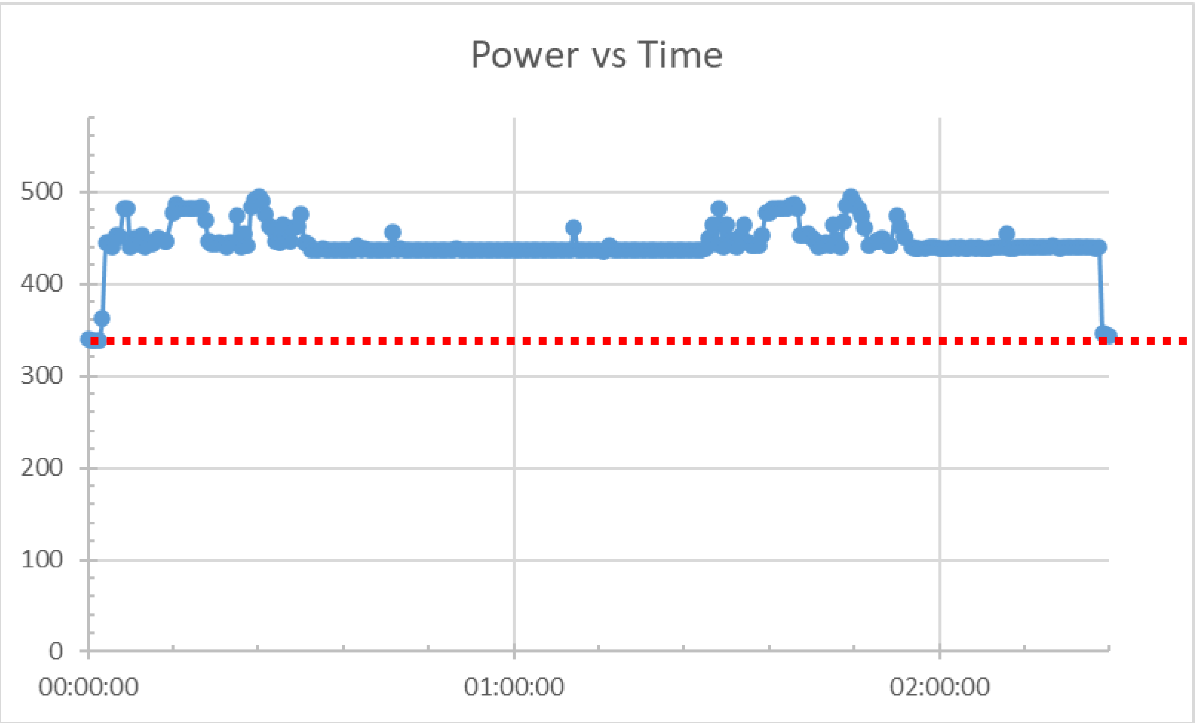
Job Profiles (C sieve)



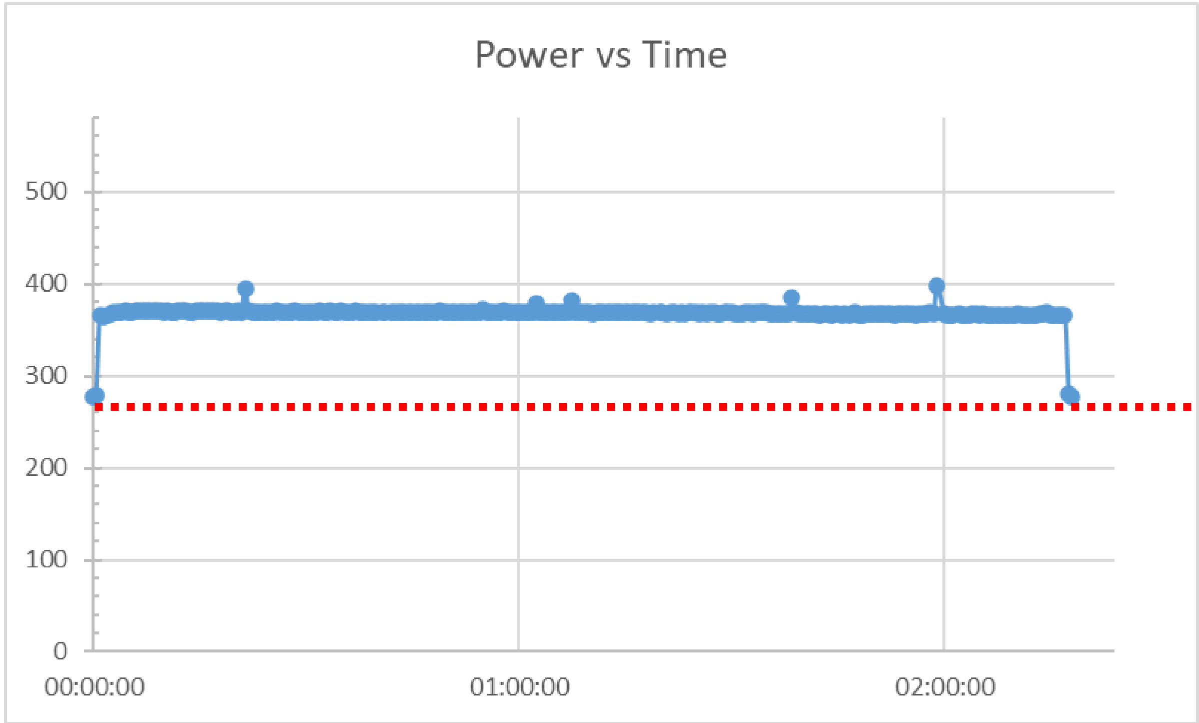
Re-run (C sieve)

(*) We had to re-run the job on the **ARM+GPU** machine, because on the first run we observed weird wobbles in the power usage, probably due to other underlying processes running on the system …

1st run (power wobbles & high idle ~320 W)

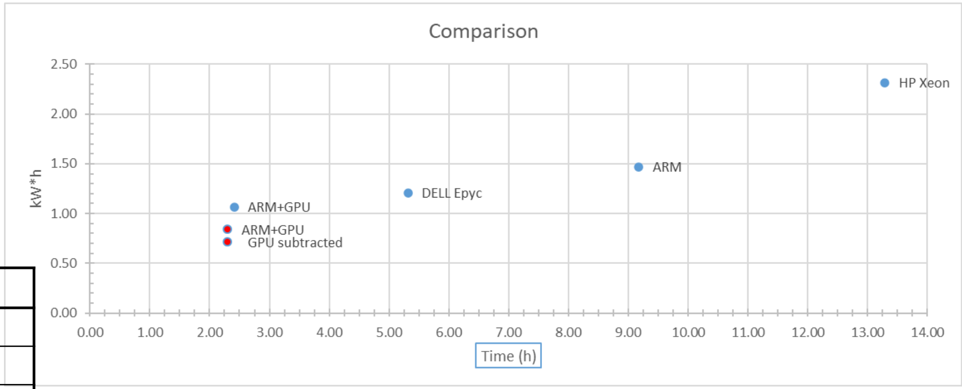


2nd run (less wobbles & lower idle ~270 W)



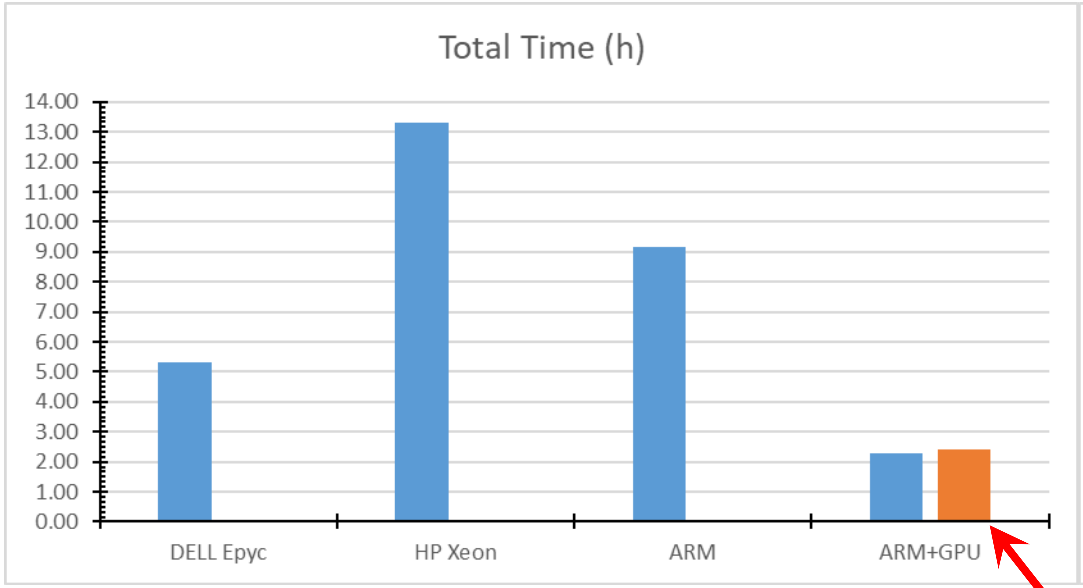
Results (C sieve)

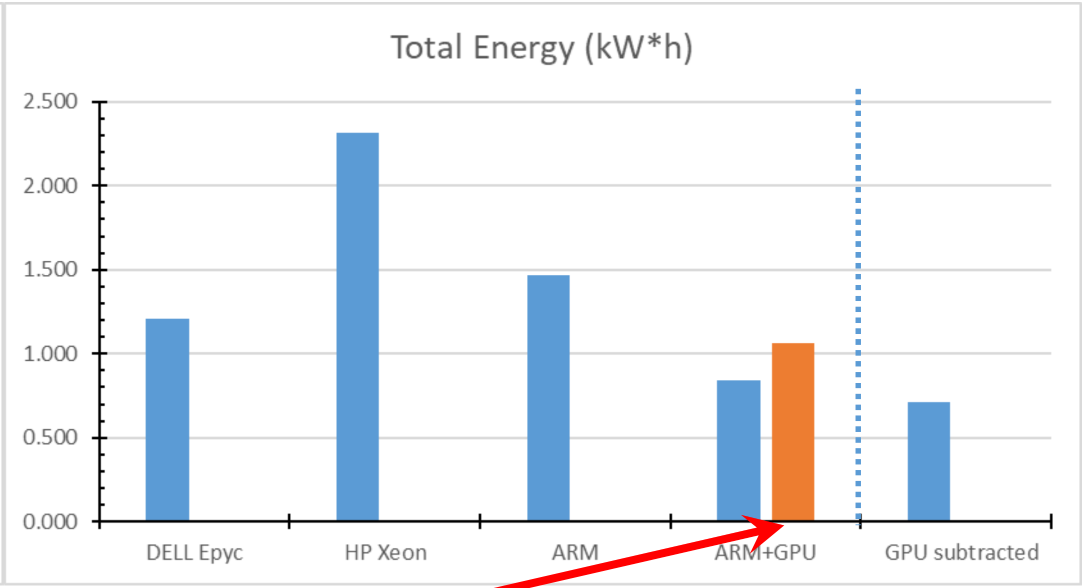
Also with compiled code, it looks like the **ARM+GPU** machine is the best at finding primes …



Prime Numbers C

Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)
DELL Epyc	128	5.32	1.209	234	146	
HP Xeon	40	13.29	2.316	177	62	
ARM	224	9.17	1.468	168	126	
ARM+GPU	80	2.30	0.838	397	277	68
GPU subtracted		2.30	0.710	329	209	0





1sr run (bad)

Large Matrix Multiplication

Large Matrix Multiplication in C with OMP using two 20k x 20k random matrices

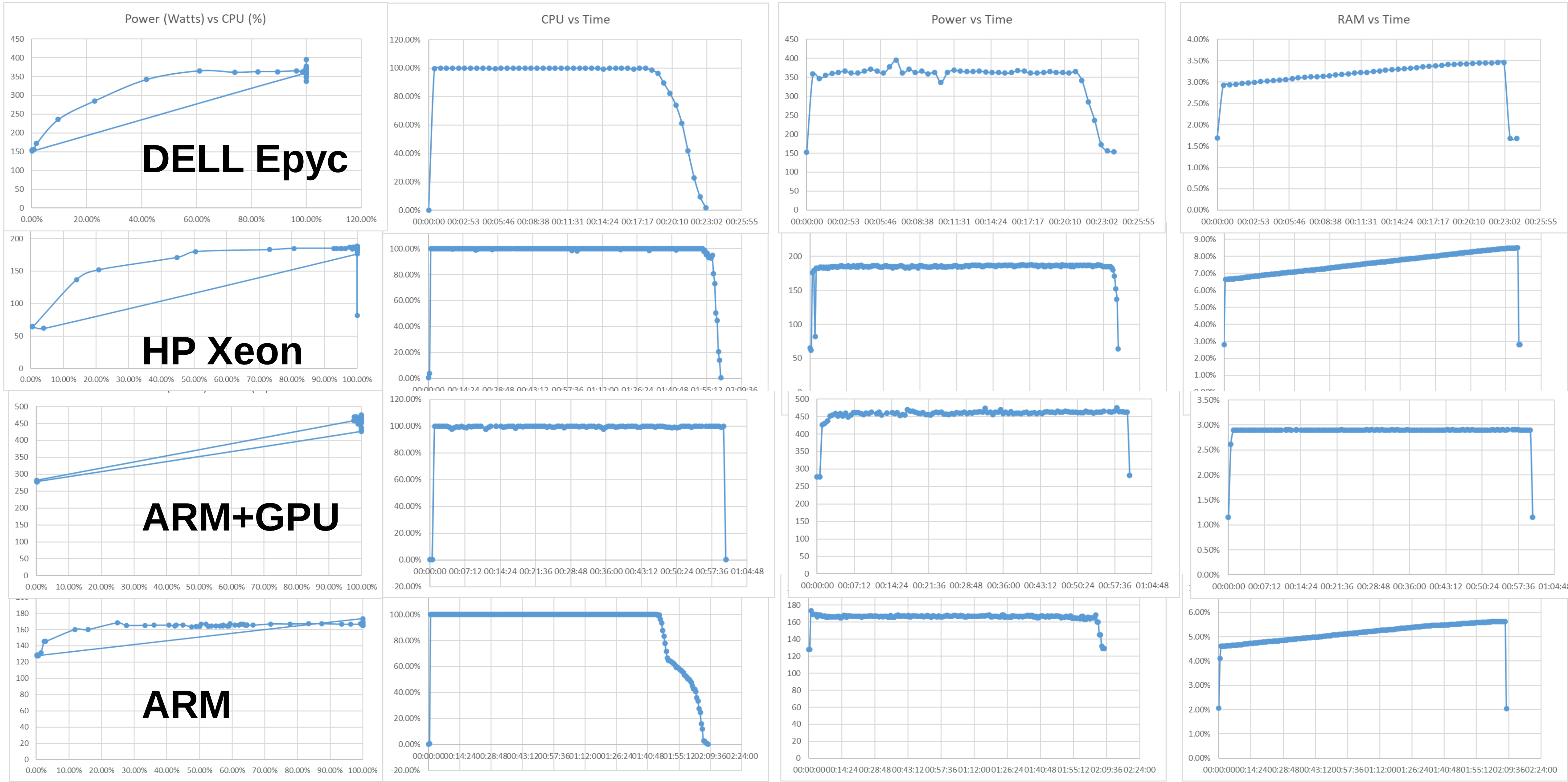
- Code is a modified version of a GitHub example **(*)**
- Compiled with the OMP library and *-mcmodel=large* flag
- Tried 2 types of matrices: integers and floating point (*double*)

```
$ ./get2IPMI.sh > ipminfo.csv

$ gcc -fopenmp -mcmodel=large matmul.c

$ ./a.out
```

```c
#include <omp.h>
...
#define N 20000
...

    #pragma omp parallel for private(i,j,k) shared(A,B,C)
    for (i = 0; i < N; ++i)
    {
        for (j = 0; j < N; ++j)
        {
            for (k = 0; k < N; ++k)
            {
                C[i][j] += A[i][k] * B[k][j];
```

(*) https://gist.github.com/metallurgix/0dfafc03215ce89fc595

Job Profiles (float Matrix)



Results (matrix)

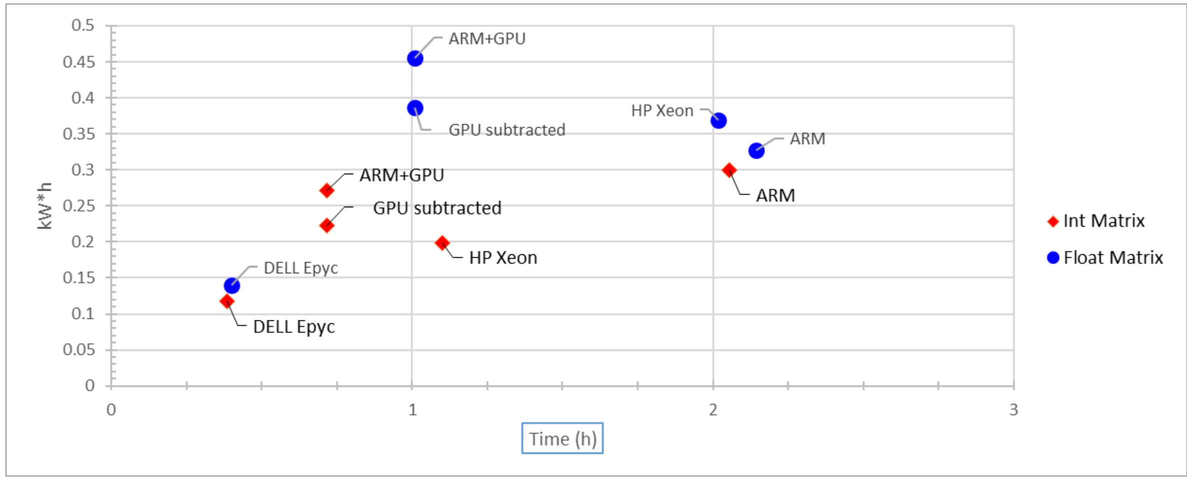
When memory usage is involved, then the **DELL Epyc** outperforms the **ARM+GPU** in speed and energy efficiency

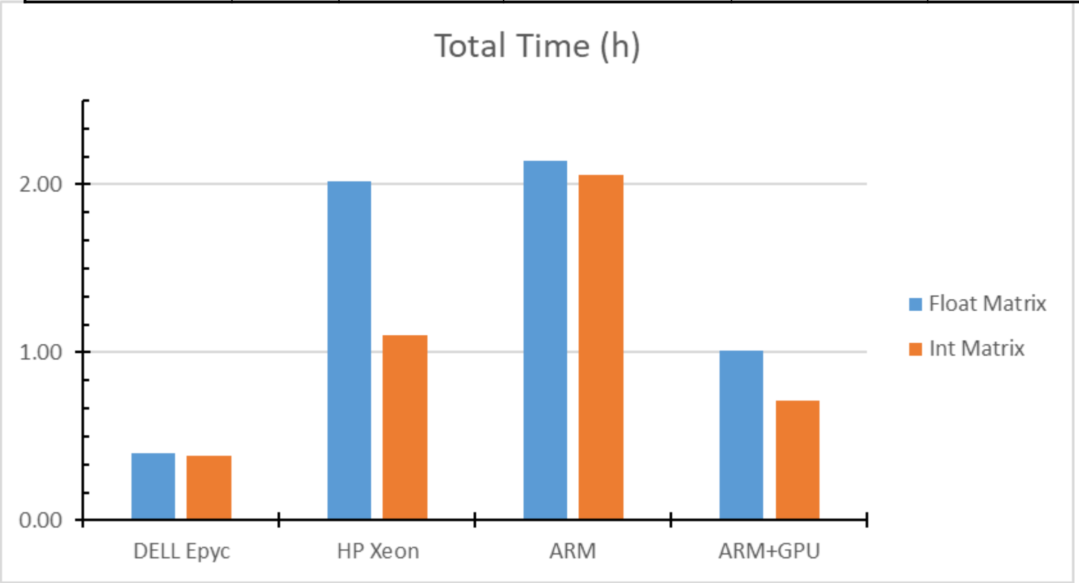
Float Matrix

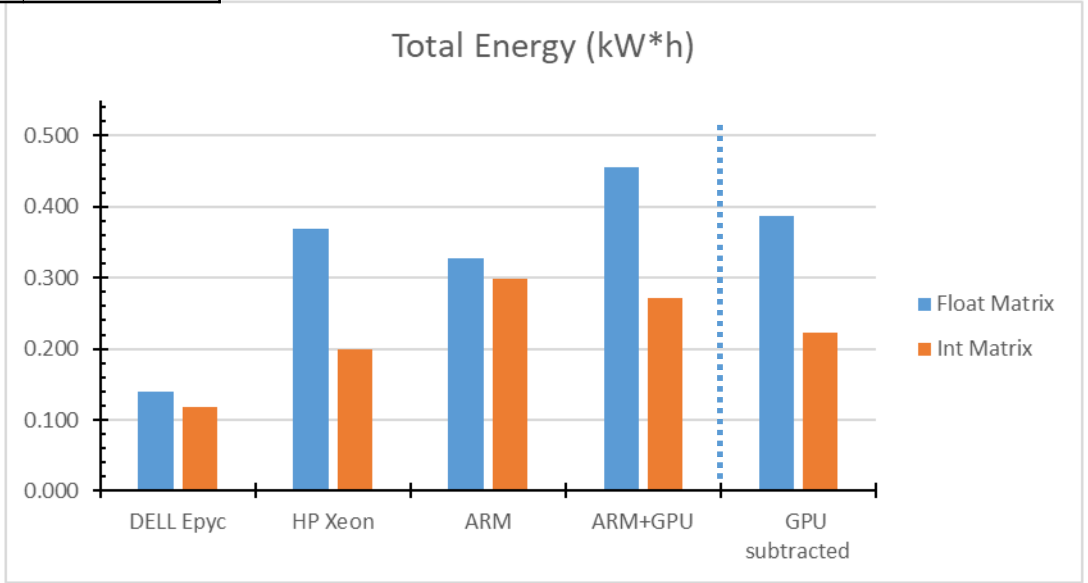
Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)
DELL Epyc	128	0.40	0.140	395	152	
HP Xeon	40	2.02	0.369	188	65	
ARM	224	2.14	0.327	173	128	
ARM+GPU	80	1.01	0.455	475	278	68
GPU subtracted		1.01	0.387	354	197	0

Int Matrix

Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)
DELL Epyc	128	0.38	0.118	368	149	
HP Xeon	40	1.10	0.199	184	62	
ARM	224	2.05	0.299	166	128	
ARM+GPU	80	0.72	0.272	393	272	68
GPU subtracted		0.72	0.223	325	204	0







About ATLAS

Then we tried using full ATLAS simulation as our next benchmark **(#)**. It has been challenging:

- AthSimulation is a colossal and complex piece of software, and we have very limited experience with it **(*)**

- It cannot really run standalone by a non root user …
  - It relies on CVMFS (picking up packages from several repo)
  - It runs more easily in Singularity

- As a non-initiated ATLAS user, it has been hard to find and select a stable version

- It is not compiled specifically on each machine, but rather centrally in *nightly builds*, which eventually disappear after a month

- To achieve multithreading, I have been using a hand-crafted combination of flags. There was no validation of the produced data. $\longrightarrow$

```
ATHENA_CORE_NUMBER=128
Sim_tf.py \ ...
  --simulator 'FullG4MT'
  --multithreaded True
```

(#) This was actually the original idea …

(*) So far, we haven't been much in touch with the ATLAS people in Glasgow.
We will definitely involve them in future developments.

ATLAS Simulations

Two different physics simulations were used, running samples of 1k and 10k Full G4 events of two different simulation types: *TTbar & Decaying Charginos*

The chosen version of the software: **AthSimulation/22.0.53**

DELL Epyc

```
Using AthSimulation/22.0.53 [cmake] with platform x86_64-centos7-gcc11-opt at
 /cvmfs/atlas-nightlies.cern.ch/repo/sw/master_AthSimulation_x86_64-centos7-gcc11-opt/2022-01-29T2101
```

ARM+GPU

```
Using AthSimulation/22.0.53 [cmake] with platform aarch64-centos7-gcc8-opt at
 /cvmfs/atlas-nightlies.cern.ch/repo/sw/master_AthSimulation_aarch64-centos7-gcc8-opt/2022-01-29T2101
```

Setting up the ATLAS framework with Singularity and CVMFS:

```
$ export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
$ alias setupATLAS='source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh’
$ setupATLAS -c centos7

Singularity> asetup AthSimulation,master,r2022-01-29T2101
Singularity> ./TTbar_10k.sh
...
```

AthSimulation Input

TTbar_10k.sh

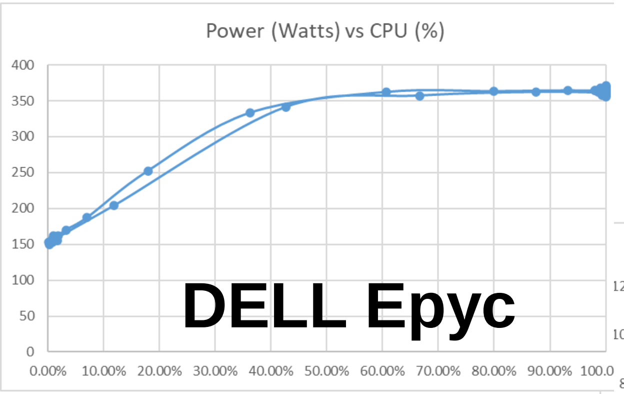
```
#!/bin/sh

export ATHENA_CORE_NUMBER=128
export TRF_ECHO=1
export MAXEVENTS=10000

Sim_tf.py \
--conditionsTag 'default:OFLCOND-MC16-SDR-14' \
--physicsList 'FTFP_BERT_ATL' \
--truthStrategy 'MC15aPlus' \
--simulator 'FullG4MT' \
--postInclude 'default:PyJobTransforms/UseF
--preInclude
'EVNTtoHITS:SimulationJobOptions/preInclude
howersFCalOnly.py' \
--preExec 'EVNTtoHITS:simFlags.TightMuonSte
--DataRunNumber '284500' \
--geometryVersion 'default:ATLAS-R2-2016-01
--inputEVNTFile "/cvmfs/atlas-nightlies.cer
art/SimCoreTests/valid1.410000.PowhegPythia
.EVNT.08166201._000012.pool.root.1" \
--outputHITSFile "TTbar2022.HITS.pool.root"
--imf False \
--maxEvents $MAXEVENTS \
--multithreaded True
```

/cvmfs/atlas-nightlies.cern.ch/repo/sw/master_AthSimulation_x86_64-centos7-gcc11-opt/2022-01-29T2101/AthSimulation/22.0.53/InstallArea/x86_64-centos7-gcc11-opt/bin/test_RUN3_FullG4_ttbar_2evts.sh

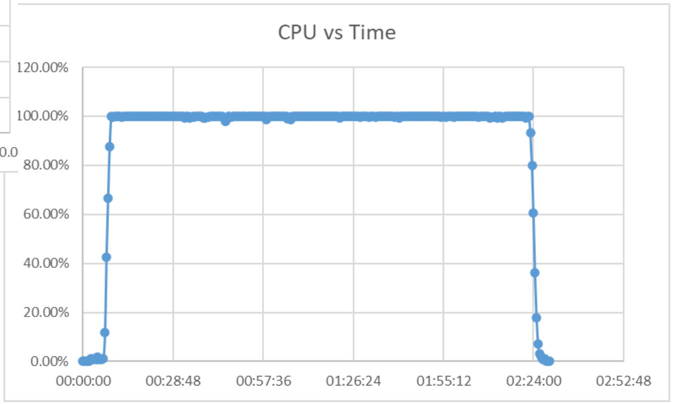
DeCh _10k.sh

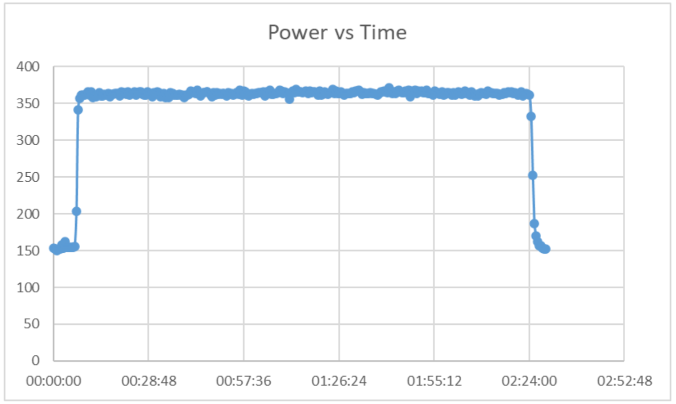
```
#!/bin/sh

export ATHENA_CORE_NUMBER=128
export TRF_ECHO=1
export MAXEVENTS=10000

Sim_tf.py \
--conditionsTag 'default:OFLCOND-MC16-SDR-14' \
--physicsList 'FTFP_BERT_ATL' \
--truthStrategy 'MC15aPlusLLP' \
--simulator 'FullG4MT' \
--postInclude 'default:PyJobTransforms/UseFrontier.py' \
--preInclude
'EVNTtoHITS:SimulationJobOptions/preInclude.BeamPipeKill.py,SimulationJobOptions/preInclude.FrozenS
howersFCalOnly.py' \
--DataRunNumber '284500' \
--geometryVersion 'default:ATLAS-R2-2016-01-00-01' \
--inputEVNTFile "/cvmfs/atlas-nightlies.cern.ch/repo/data/data-
art/SimCoreTests/mc15_13TeV.448307.MGPy8EG_A14N23LO_mAMSB_C1C1_5000_208000_LL4p0_MET60.evgen.EVNT.e
6962.EVNT.15631425._000001.pool.root.1" \
--outputHITSFile "DeCh_HITS.pool.root" \
--maxEvents $MAXEVENTS \
--imf False \
--multithreaded True
```

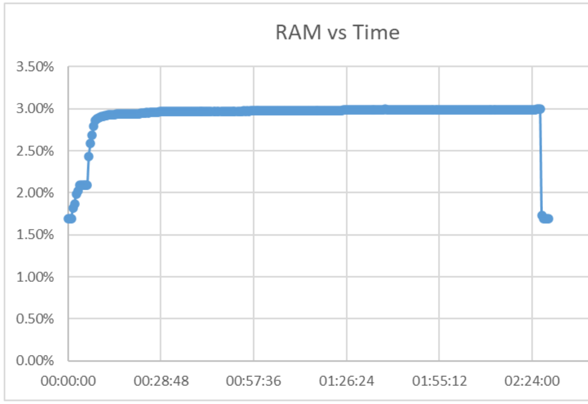
Job Profiles (ATLAS 10k)

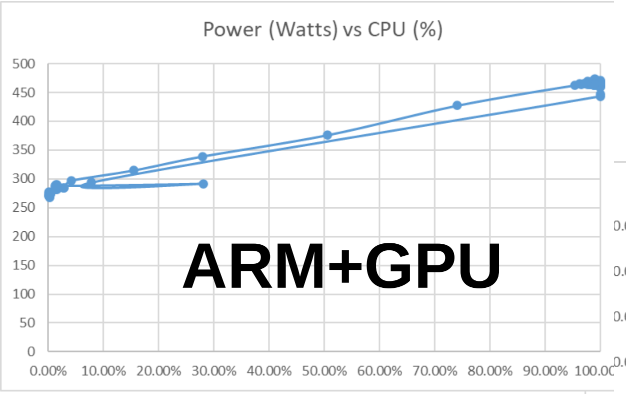


DELL Epyc

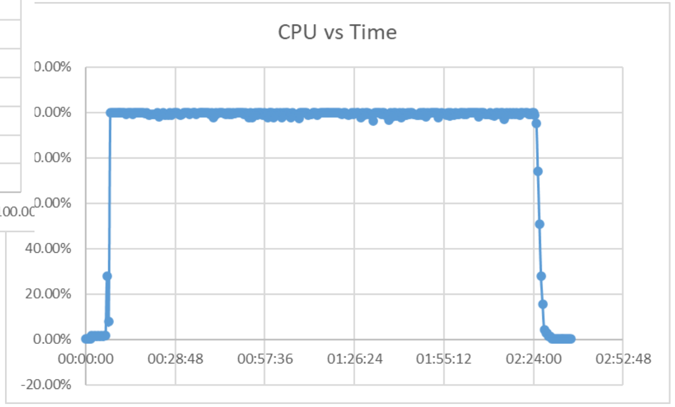


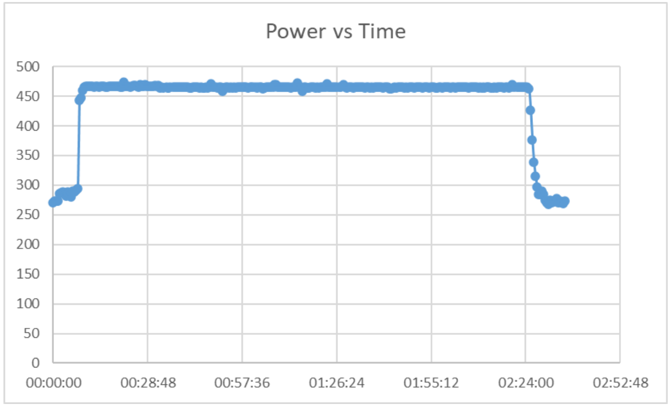


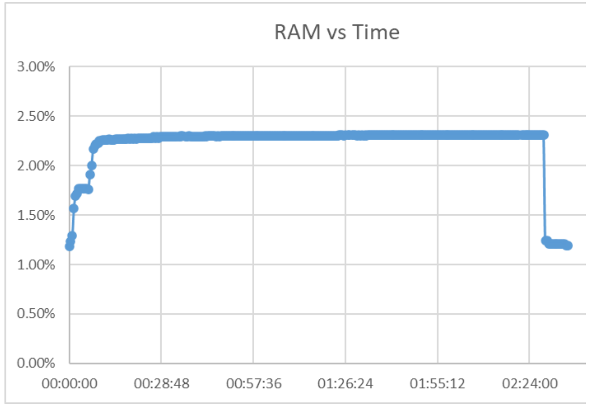




ARM+GPU







Results (ATLAS 10k)

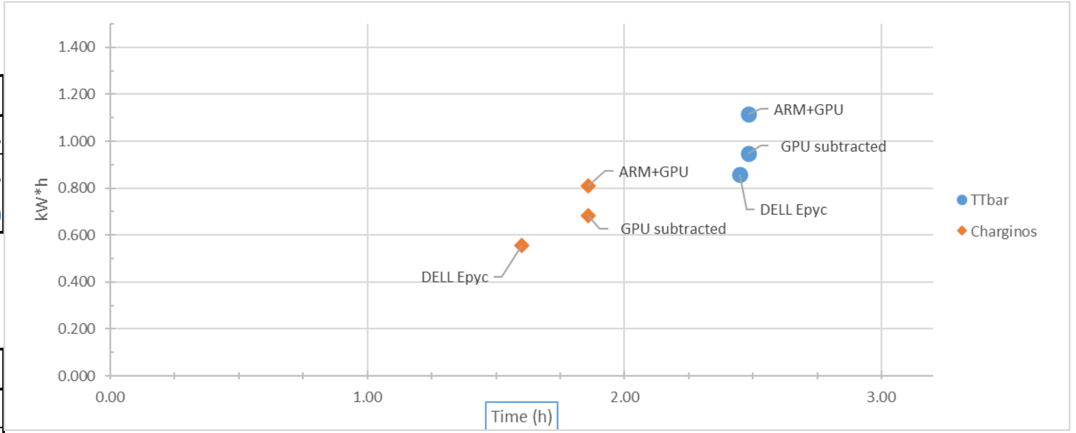
In both cases, the execution time is very similar. The **DELL Epyc** machine looks slightly more energy efficient.

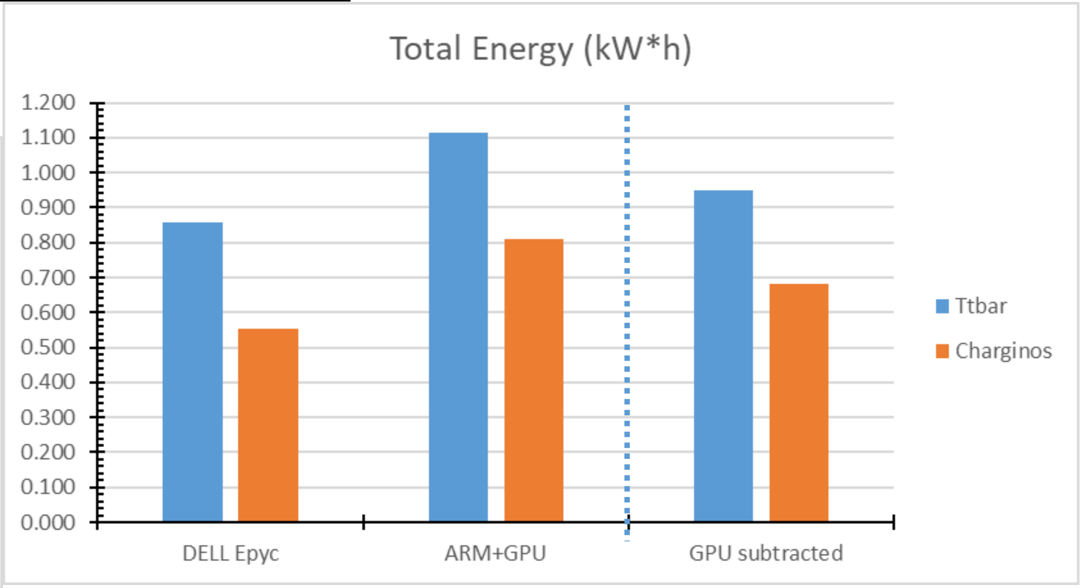
TTbar

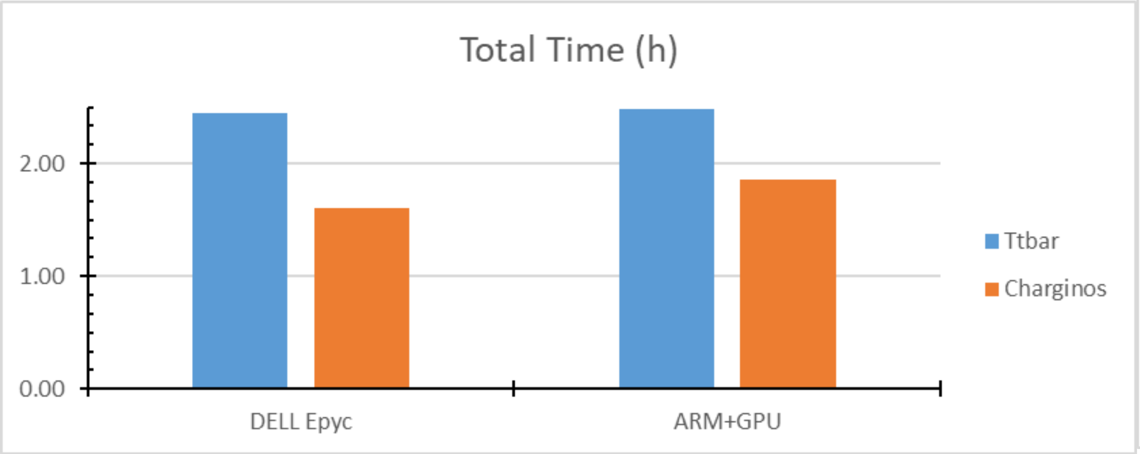
Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)	Above Idle (kW*h)
DELL Epyc	128	2.45	0.858	371	150		0.4908
ARM+GPU	80	2.48	1.114	474	274	67	0.4338
GPU subtracted		2.48	0.949	407	207	0	0.0000

Charginos

Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)	Above Idle (kW*h)
DELL Epyc	128	1.60	0.555	370	148		0.3182
ARM+GPU	80	1.86	0.811	465	263	69	0.3223
GPU subtracted		1.86	0.683	396	194	0	0.0000







Results (ATLAS 10k)

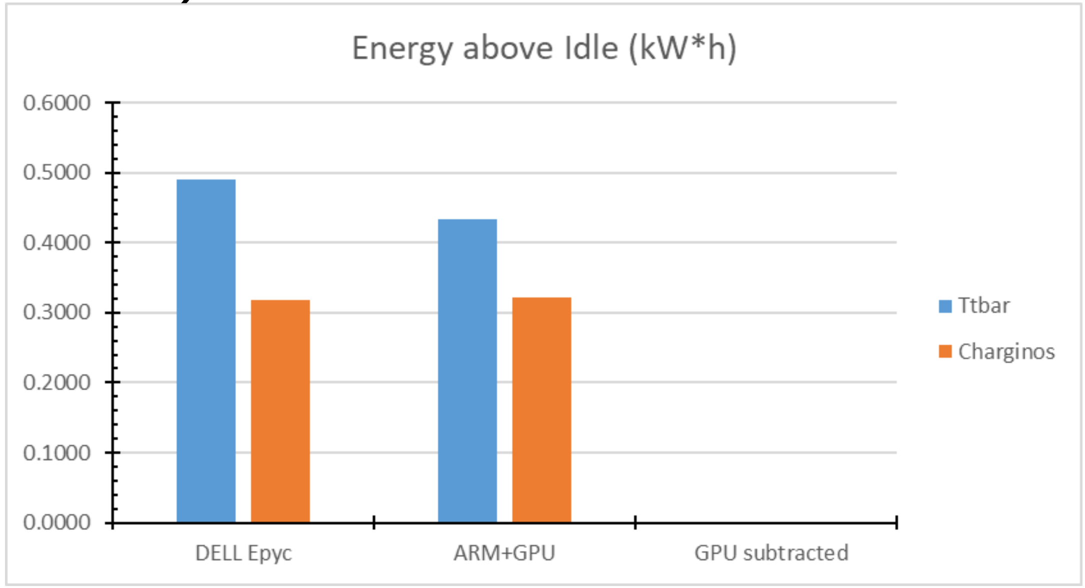
To account for the different idle consumption, we can subtract it from the total and compare the energy in excess of idle (above idle):

TTbar

Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)	Above Idle (kW*h)
DELL Epyc	128	2.45	0.858	371	150		0.4908
ARM+GPU	80	2.48	1.114	474	274	67	0.4338
GPU subtracted		2.48	0.949	407	207	0	

Charginos

Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)	Above Idle (kW*h)
DELL Epyc	128	1.60	0.555	370	148		0.3182
ARM+GPU	80	1.86	0.811	465	263	69	0.3223
GPU subtracted		1.86	0.683	396	194	0	



When we subtract the idle, the **ARM+GPU** machine looks slightly more efficient.

However, this procedure might be questionable …

More Results (ATLAS 1k)

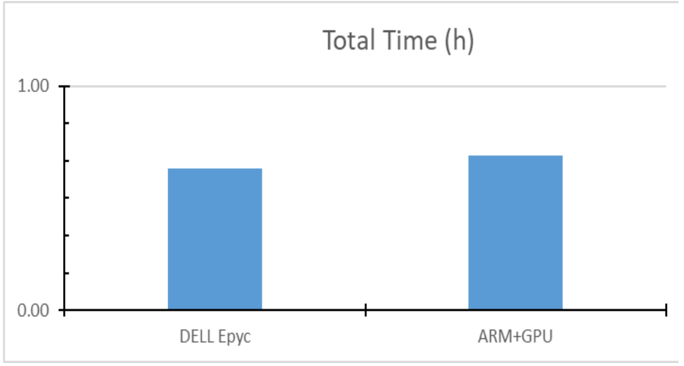
Here some more numbers from a clean run of a TTbar simulation (1k events) on **DELL Epyc** and **ARM+GPU**

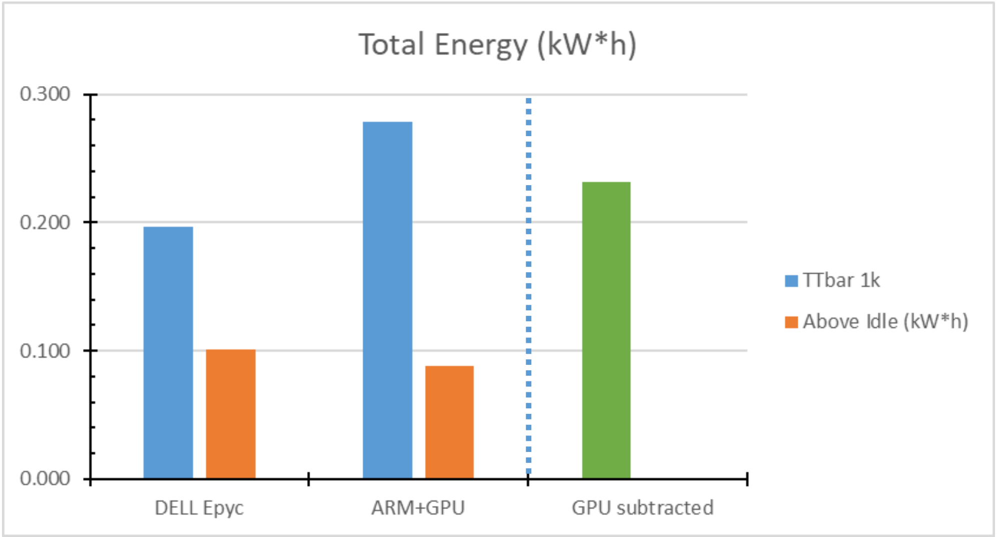
TTbar 1k

Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)	Above Idle (kW*h)
DELL Epyc	128	0.63	0.197	366	152		0.1007
ARM+GPU	80	0.69	0.278	468	275	67	0.0879
GPU subtracted		0.69	0.232	401	208	67	0.0000

	DELL Epyx	ARM+GPU	
TTbar	1,000	1,000	**events**
Total time:	00 00:38:00	00 00:41:31	**min**
	2,280	2,491	**sec**
Per event:	2.2800	2.4910	**sec/event**
	0.44	0.40	**event/sec**
Total energy:	709,200.00	1,001,640.00	**Joules**
	0.20	0.28	**kW*h**
Energy/Event:	709.20	1,001.64	**J/event**
	0.1970	0.2782	**W*h/event**
Average Power	307.01	402.27	**W**
Max Power	366.00	468.00	**W**
Min Power	151.00	268.00	**W**
Idle (estimate)	152.00	275.00	**W**
GPUs (average)		67.27	**W**
Idle subtracted:	152.00	275.00	**W**
Total energy:	362,640.00	316,615.00	**Joules**
	0.10	0.09	**kW*h**
Energy/Event: (*)	362.64	316.61	**J/event**
	0.101	0.09	**W*h/event**
Average Power	155.01	127.27	**W**
GPU subtracted:		67.27	**W**
GPU Total:		167,499.30	**Joules**
		0.05	**kW*h**
Total energy:		834,140.70	**Joules**
		0.23	**kW*h**
Energy/Event:		834.14	**J/event**
		0.23	**W*h/event**
Average Power		335.00	**W**

(*) The energy/event tends to decrease with a larger simulation. E.g., total energy of the 10k simulation is about 4 times the 1k …





Conclusions & Outlook

It was a nice exercise, but there aren’t many conclusions at this stage …

- ✓ Results show that the **ARM+GPU** machine is quicker at pure calculations, while **DELL Epyc** machines is generally faster and more power efficient when memory usage is involved (large matrices, ATLAS simulations)
- ✓ In general, the performance differences between comparable machines (**DELL Epyc** & **ARM+GPU**) are small for ordinary tasks
- ➢ This kind of comparison may be inconclusive (see “Limitations” slides)
- ➢ We did not involve ATLAS people as much as we could have (e.g., to select a proper software version and type of job to execute)
- ? If the study should continue, it would be good to have more similar hardware to compare (e.g., same machines with different CPU) and direct access to them, including the ability to use a physical metered PDU

Thanks.