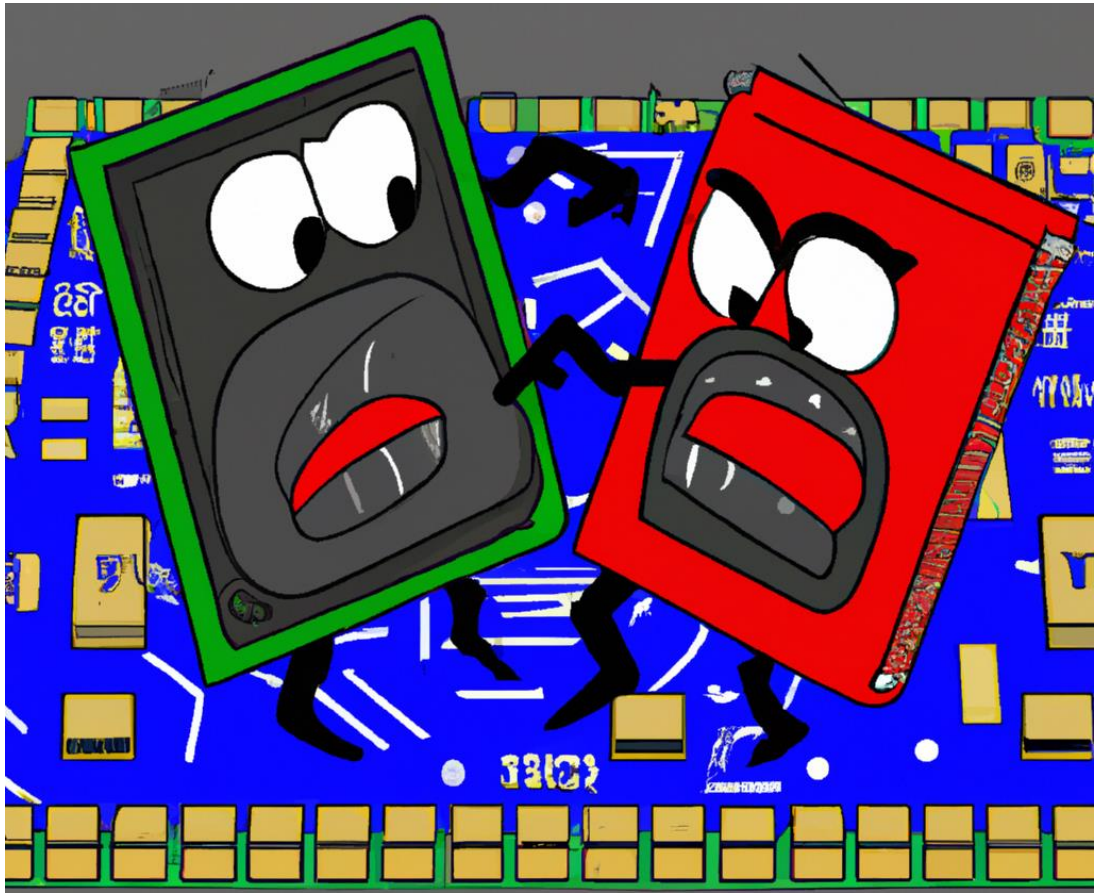
Power Efficiency: x86 vs ARM

(recap and outlook)



Outline

(transitional talk)

- Power comparison **x86_64** & **arm64**
  - Available hardware (Glasgow + Leichester)
  - Power reading & exporter tools
  - Benchmarks & results (ATLAS)
  - Limitations
- Outlook
  - New hardware at Glasgow
  - Looking forward to HEP-Score

Original slides:

https://indico.cern.ch/event/1128343/contributions/4787174/attachments/2412950/4129612/PowA_GridPP47.pdf

Abstract (old)

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 two remote **arm64** machines and two 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, compiled C code and two different types of ATLAS full Geant4 MT 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 ...

ScotGrid Glasgow:

Emanuele Simili, Gordon Stewart, Samuel Skipsey, David Britton

Special Thanks:

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

Available Hardware (old)

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)*

Power Readings

Power readings were achieved by two custom scripts, to collect and export metrics such as CPU, RAM and IPMI 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* ☹

Next time will use ROOT !

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

Benchmarks & Results (old)

Various benchmarks were attempted, where possible.
All benchmarks used some sort of multithreading (OMP, G4MT):

- Prime number sieve (BASH script): prime numbers up to 1 M (*78'498 primes*)
- Prime number sieve (C with *OMP*): prime numbers up to 100 M (*5'761'455 primes*)
- Large Matrix Multiplication (C with *OMP*): 20k * 20k random matrix (*int* & *double*)
- 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)	4.45 kW*h , 18 h	3.14 kW*h , 23 h	4.55 kW*h , 31 h	0.68 kW*h , 2 h
compiled	Prime Number Sieve (C + OMP)	1.2 kW*h , 5 h	2.3 kW*h , 13 h	1.5 kW*h , 9 h	0.8 kW*h , 2 h
	Large Matrix Multiplication (int)	0.14 kW*h , 30 min	0.37 kW*h , 2 h	0.33 kW*h , 2 h	0.45 kW*h , 1 h
	Large Matrix Multiplication (float)	0.12 kW*h , 30 min	0.2 kW*h , 1 h	0.3 kW*h , 2 h	0.27 kW*h , 45 min
AthSimulation	ATLAS TTbar (1k events)	0.2 kW*h , 35 min	skipped	not possible	0.23 kW*h , 35 min
	ATLAS TTbar (10k events)	0.86 kW*h , 2 h 30 min	skipped	not possible	0.95 kW*h , 2 h 30 min
	ATLAS Decaying Charginos (10k events)	0.55 kW*h , 1 h 30 min	skipped	not possible	0.68 kW*h , 1 h 45 min

I will not present all sets of results here. Please refer to the original slides presented at GridPP47.

About ATLAS

We tried full ATLAS simulation as our HEP benchmark. It has been challenging:

- **Athena** is a colossal piece of software, and we had very limited experience with it **(*)**

- It was not easy to run it standalone by a non root user …
  - It relies on CVMFS and picks up packages from several repos
  - It uses Singularity (we found it easier to run in a container)

- As a non-initiated ATLAS user, it has been hard to find a stable version …
  - Stable releases are not compiled for ARM (https://bigpanda.cern.ch/globalview/)
  - I had to rely on *nightly builds*, which eventually disappear after a month **(#)**

- Also, not sure we used the best workload …
  - Full G4 *TTbar* and *Decaying Charginos* test jobs (see .sh)
  - Hand-crafted combination of flags o achieve multithreading
  - There was no validation of the produced data (ROOT file readable)

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

(*) Luckily, a new colleague has just joined ScotGrid Glasgow, he used to work in ATLAS. He will help to define an ATLAS workload that works on both architectures.

(#) Unless you bother the right people

Simulation Inputs

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
```

ATLAS Simulations

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

(nightly builds were available for both **x86_64** and **aarch64**)

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
...
```

First Results (ATLAS 1k)

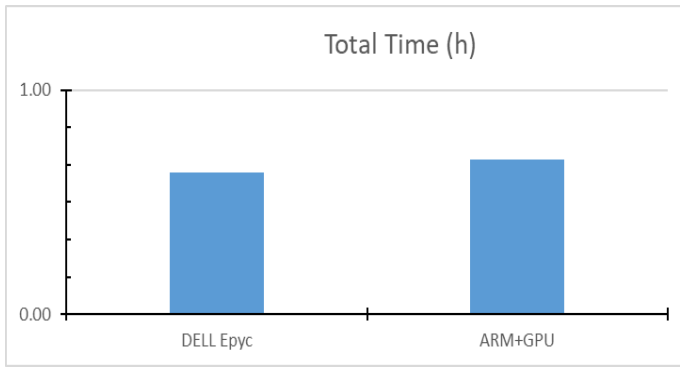
First look into an ATLAS simulation (TTbar, 1k events), comparing **DELL Epyc** with **ARM+GPU**

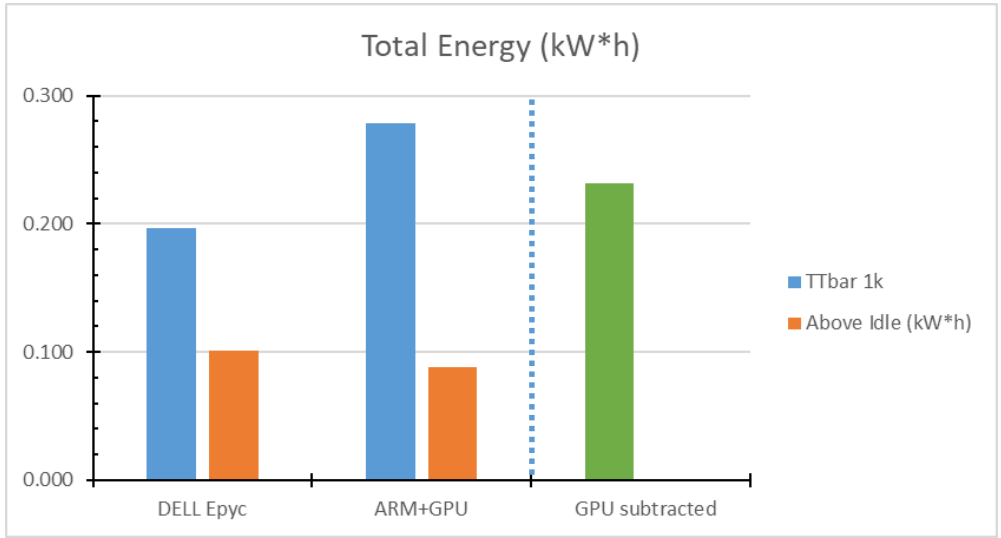
TTbar 1k

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

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

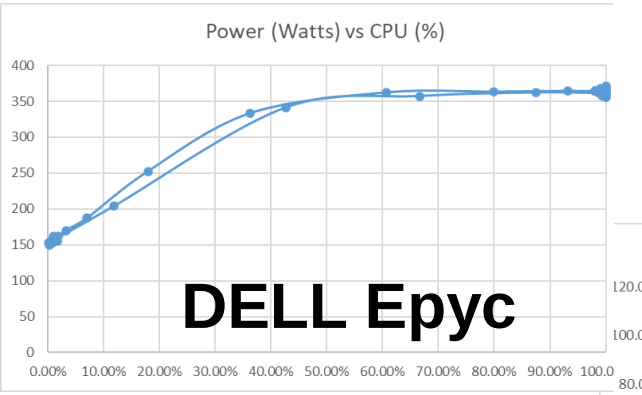
	DELL Epyx	ARM+GPU	
TTbar	1,000	1,000	**events**
Total time:	00 00:38:00	00 00:41:31	**min**
	2,280	2,491	**sec**
Per event:	2.2800	2.4910	**sec/event**
	0.44	0.40	**event/sec**
Total energy:	709,200.00	1,001,640.00	**Joules**
	0.20	0.28	**kW*h**
Energy/Event:	709.20	1,001.64	**J/event**
	0.1970	0.2782	**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**



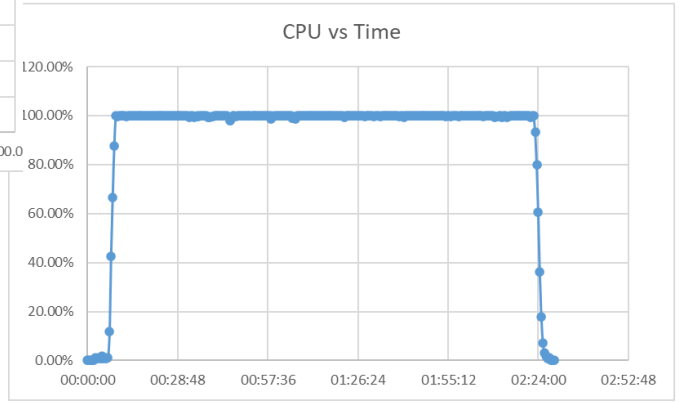


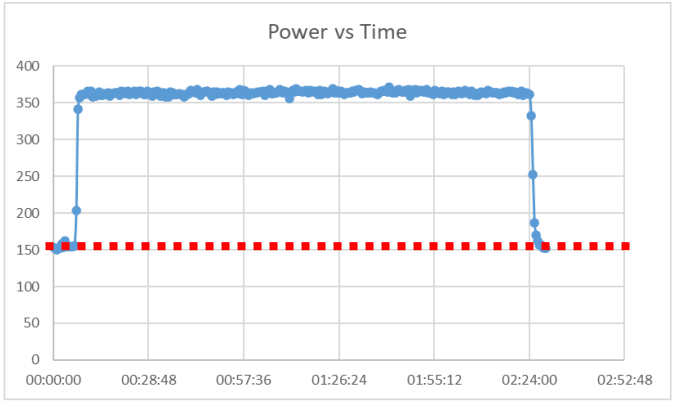
What a large idle !

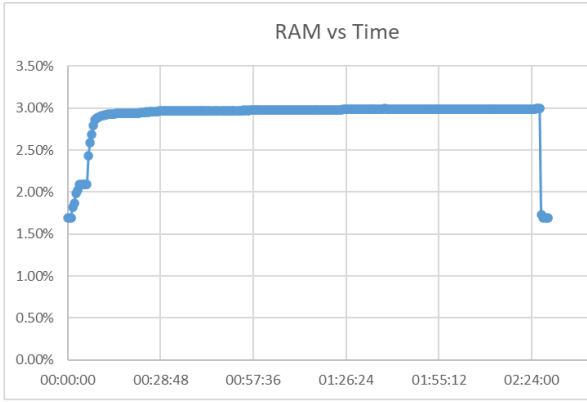
Job Profiles (ATLAS 10k)

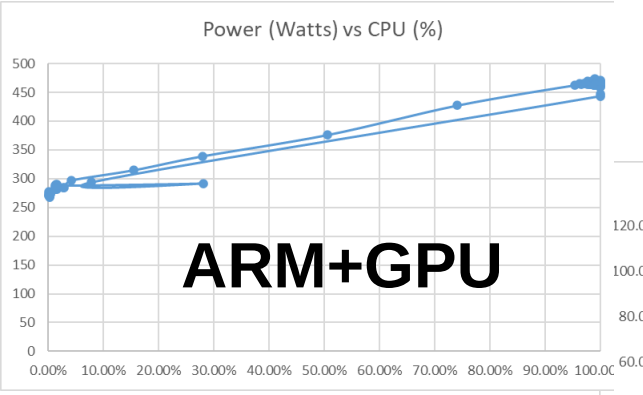


DELL Epyc

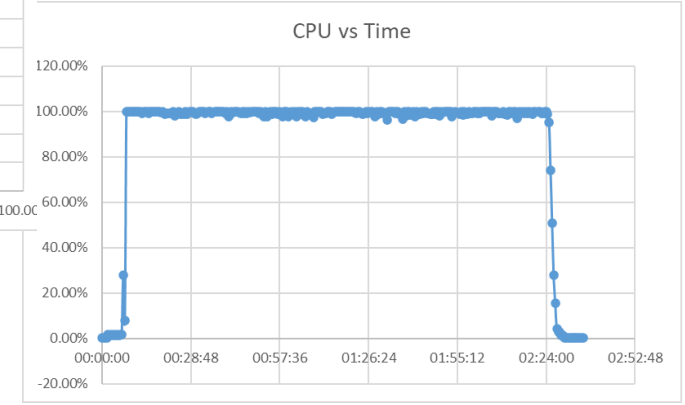


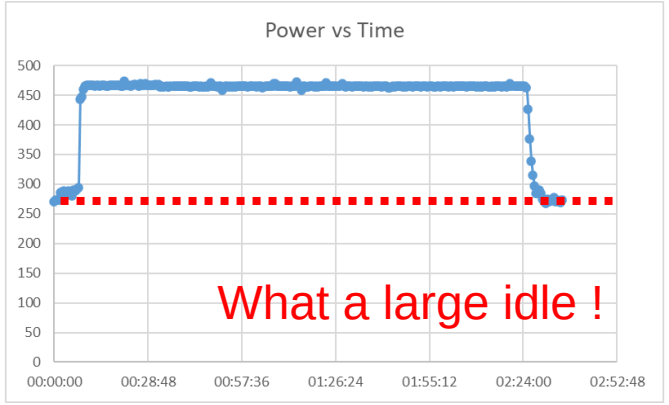




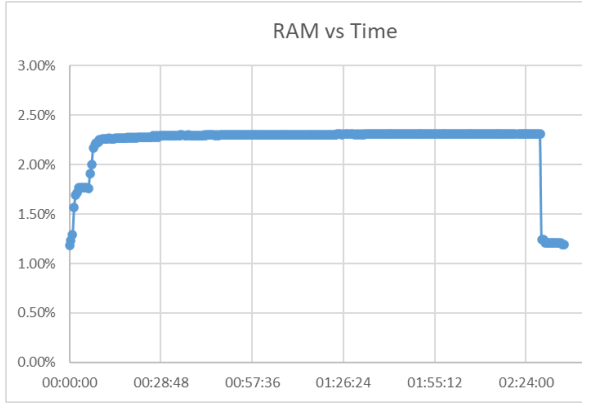


ARM+GPU





What a large idle !



Results (ATLAS 10k)

In both cases, the execution times are very similar, especially after subtracting the GPU.

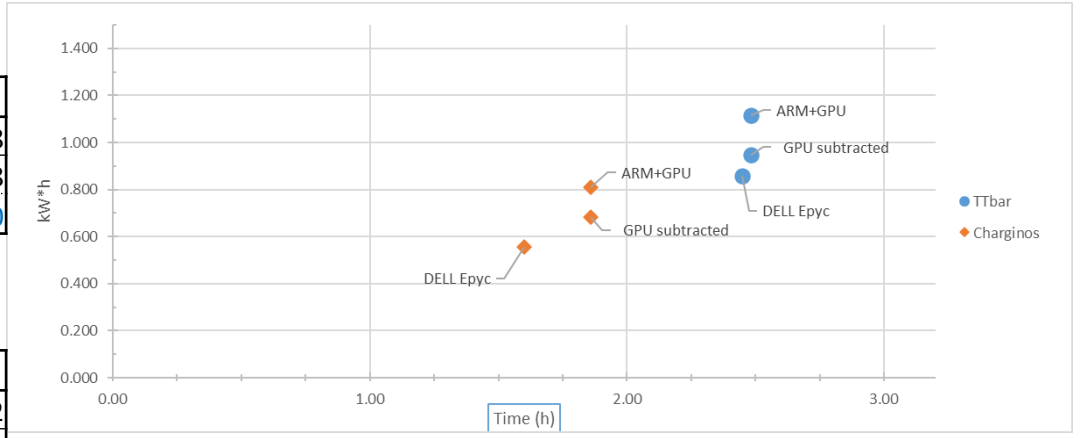
The **DELL Epyc** looks slightly more energy efficient.

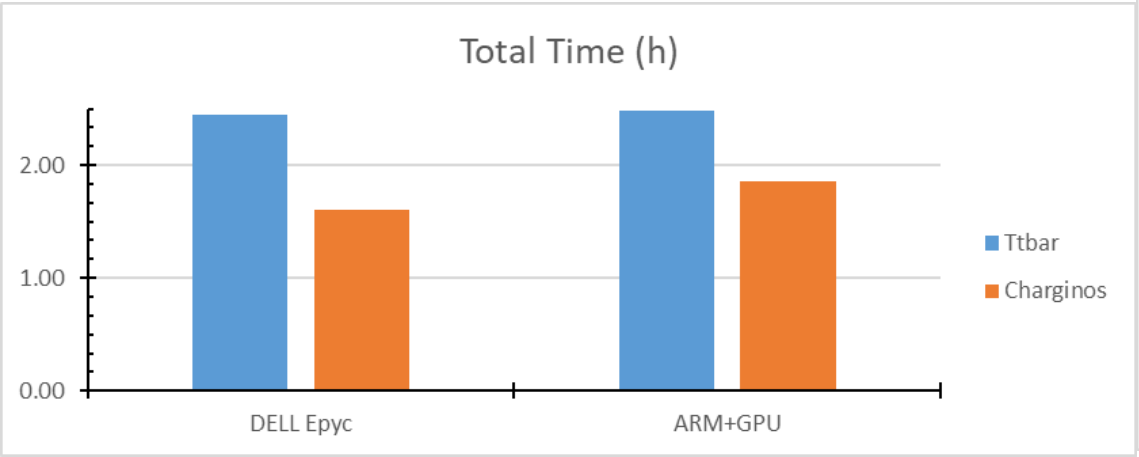
TTbar

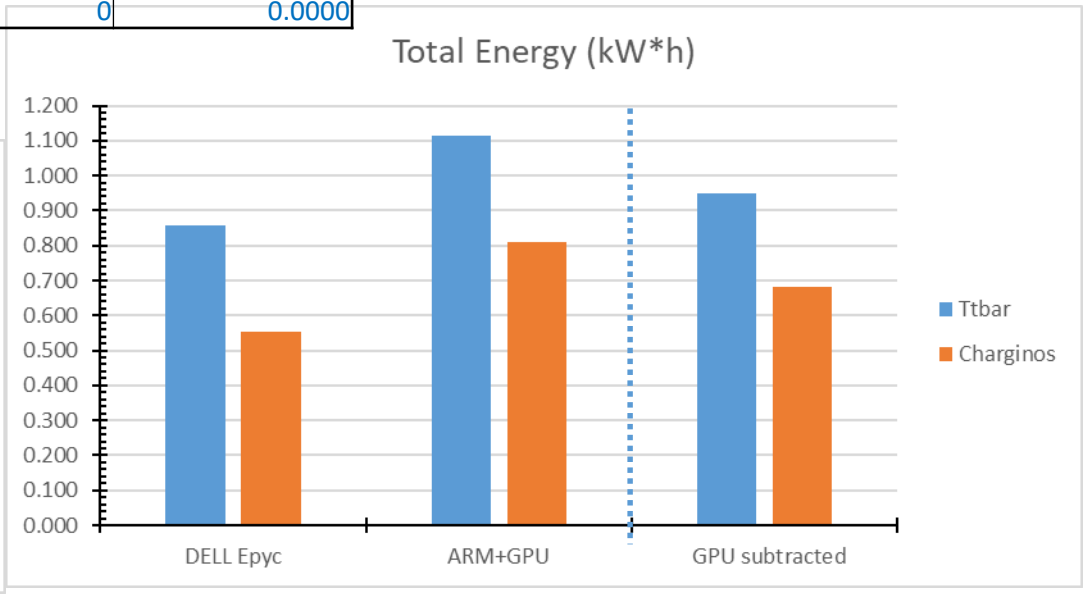
Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)	Above Idle (kW*h)
DELL Epyc	128	2.45	0.858	371	150		0.4908
ARM+GPU	80	2.48	1.114	474	274	67	0.4338
GPU subtracted		2.48	0.949	407	207	0	0.0000

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)

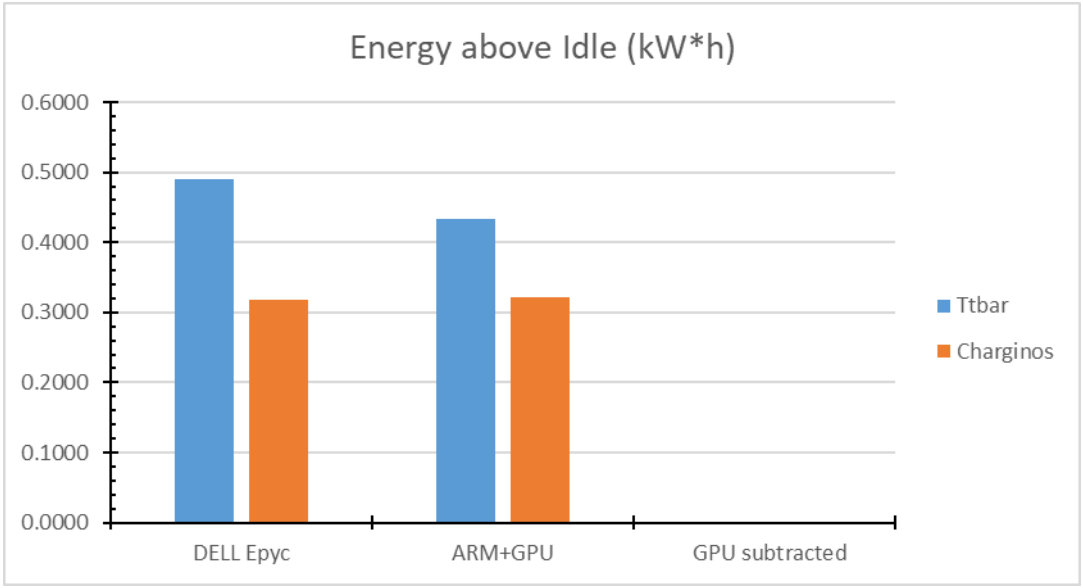
Or … to account for the different idle consumption, we can subtract it from the total and compare the *energy in excess* of idle:

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 power efficient.

However, this procedure might be questionable …

Limitations (old)

(#) 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 was just a normal user with limited privileges. Therefore, I had 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 had 2 powerful **Nvidia Ampere GPUs** just sitting and increasing the idle.

- Power measurements were 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 was no validation of power data (e.g., an external power-meter).

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

- Cooling was 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).

Outlook

To address the limitation of our previous study on efficiency and power consumption (ref. *GridPP47*), we have purchased two new servers with almost identical hardware specs but different CPU architectures: **arm64** and **x86_64**.

This will allow us to re-run the various benchmarks while comparing power consumption in a very controlled environment, where all variables can be accounted for (e.g., raw PDU power output, cooling, …), and where we have access to the full range of machine metrics (*node_exporter*).

As we have done before, we will run different benchmarks including BASH scripts, compiled C programs, ATLAS simulations (*) and possibly HEP-Score (**).

Results will hopefully make more sense than previous ones (***), and they will be presented at ACAT 2022 next month (****).

(*) This time we have ATLAS people to provide an actual workload
(**) ARM builds of some benchmarks are already available (CMS, ATLAS)
(***) E.g., we already see a huge difference in idle (AMD almost twice than ARM)
(****) Both servers arrived (with huge delay), and work is in progress

Purchased Hardware

x86_64: Single AMD EPYC 7003 series processor family

Server:	AMD EPYC 7003 UP Server System (SuperMicro)
CPU:	AMD EPYC 7643 48 core 96 Thread 2.3GHz processor
RAM:	256GB (16 x 16 GB) DDR4 3200MHz ECC Registered memory
Storage:	3.84TB Samsung PM9A3 M.2 (2280)

arm64: Single socket Ampere Altra MAX Processor

Server:	ARM 1U Rackmount Server system
CPU:	ARM Q80-30 80 core 210W TDP processor
RAM:	256GB (16 x 16 GB) DDR4 3200MHz ECC Registered memory
Storage:	3.84TB Samsung PM9A3 M.2 (2280)

Already available @ Glasgow:

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)*

… old and cranky

Benchmarks (new)

Available benchmarks (already implemented/tested)**:**

- Prime number sieve (BASH script)
- Prime number sieve (C with OMP)
- Large Matrix Multiplication (C with OMP) and/or other compiled operations
- Full G4MT ATLAS Simulation (~ in progress)

Other benchmarks (to implement/test)**:**

- HEP-Score / HEP Benchmark suite (~ if available)
- I/O bound jobs (e.g., run ATLAS analysis on a remote data set)
- CPU intensive Python tasks (e.g., PyTorch on CPU)
- Other experiments …

I am still open to any suggestions and criticism on what / how I should run !

HEP-Score & me

I started talking with the HEP-Score Working Group before the summer, because of mutual interest and partial overlap with my current research on power efficiency:

- I wish to use HEP-Score as a standard HEP workload to rate different architectures on power consumption and execution efficiency
- I can help testing the HEP-Score on different architectures (x86, ARM, and hopefully GPU - we already have one waiting :)
- It would be good to include the power readings in the standard output of HEP-Score (peak / idle power and integrated energy consumption for a given load) *

* **This is becoming increasingly important for future hardware procurement !**

HEP-Scores @ Glasgow

So far, I have been able to successfully run the HEP-score on all types of hardware at Glasgow (except ARM). Results submitted to the global Kibana dashboard at CERN.

Local HEP-score results (UKI-SCOTGRID-GLASGOW):

d20 (DELL PowerEdge C6525, 2020)	score: 2060.4333
d21 (DELL PowerEdge C6525, 2021)	score: 1983.4387
d22 (DELL PowerEdge C6525, 2022)	score: 2141.87
h17 (HP ProLiant DL60 Gen9, 2017)	score: 451.9846
h16 (HP ProLiant DL60 Gen9, 2016)	score: 368.671
s15 (SuperMicro SYS-6028TR-HTR, 2015)	score: 389.7718
GPU (DELL PowerEdge R7525 + 2x Nvidia A100, 2022)	score: 1906.9971 (CPU only)

I did not calculate the total score of our site just yet …

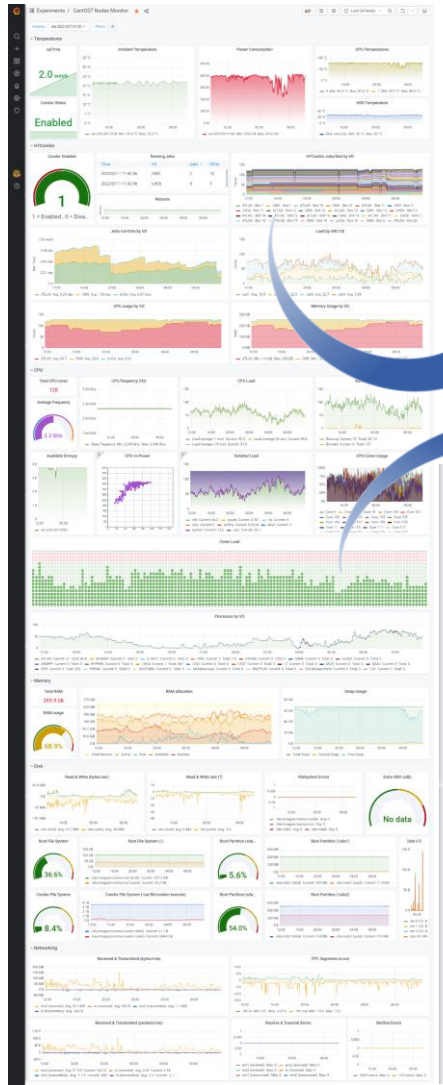
Thanks.

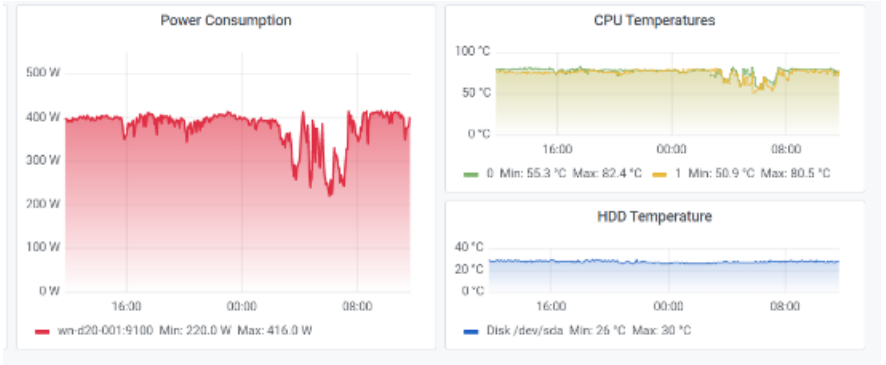
Visualization (local)

Local power readings and resource usage is extracted by *node_exporter* (which can be customized with any metric).

Data is colourfully visualised in a custom *Grafana* dashboard,

and can be exported to CSV format via the *Prometheus* API and analysed in ROOT





WorkerNode View

C Sieve

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

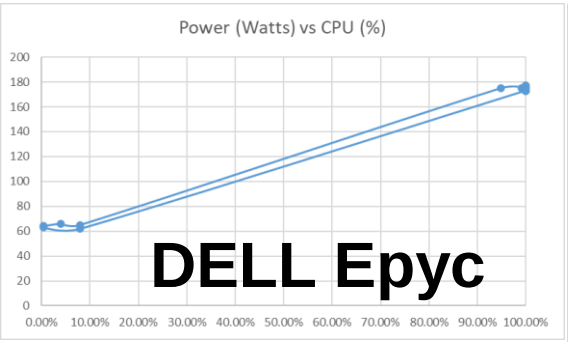
- Code from the web **(*)**
- Compiled on each machine with **gcc** and the OMP library
- It is executed while collecting metrics

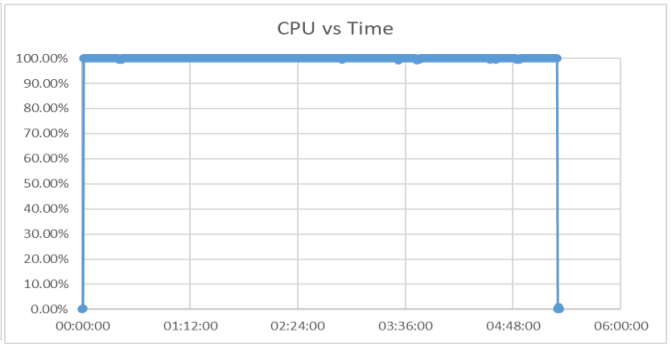
```
$ ./get2IPMI.sh > ipminfo.csv

$  gcc -fopenmp mtprime.c

$ ./a.out
```

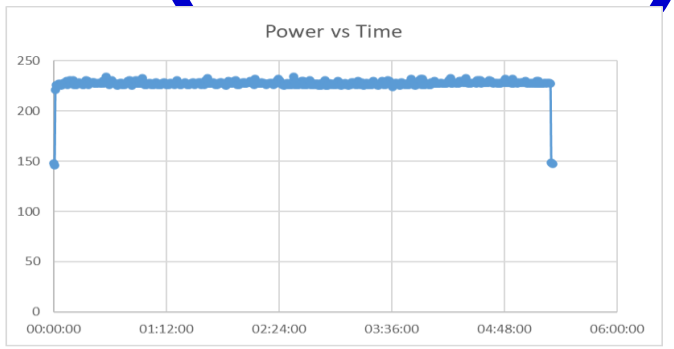
```
#include <omp.h>
...

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

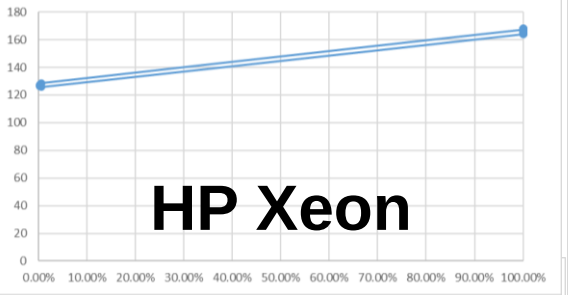
(*) https://stackoverflow.com/questions/9244481/how-to-get-100-cpu-usage-from-a-c-program

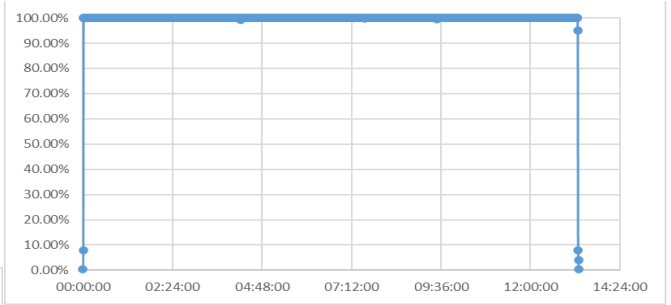
Job Profiles (C sieve)

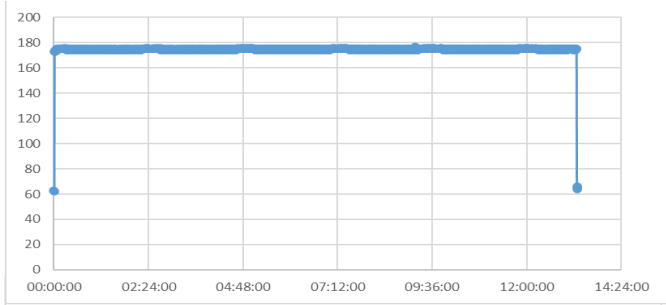


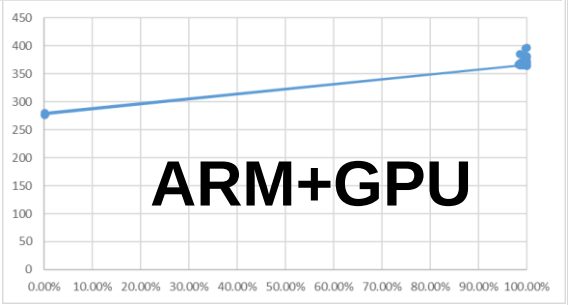


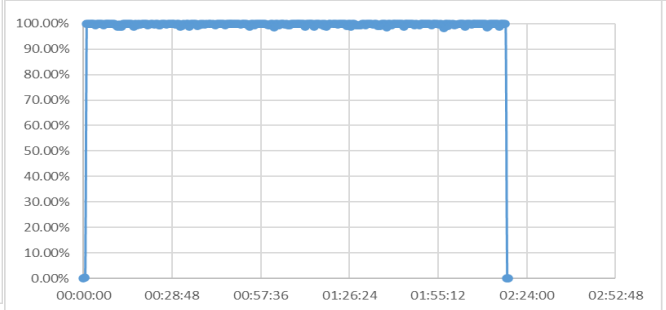


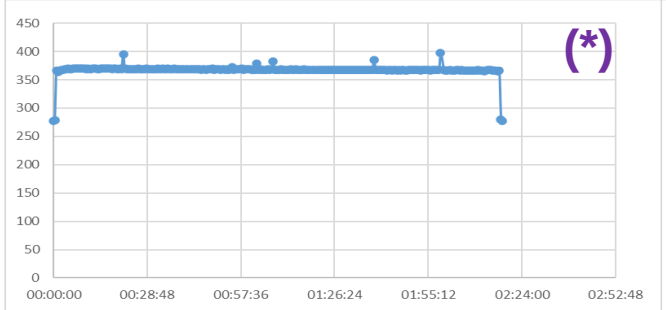


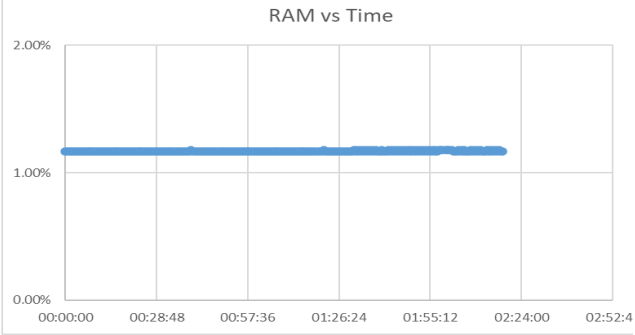


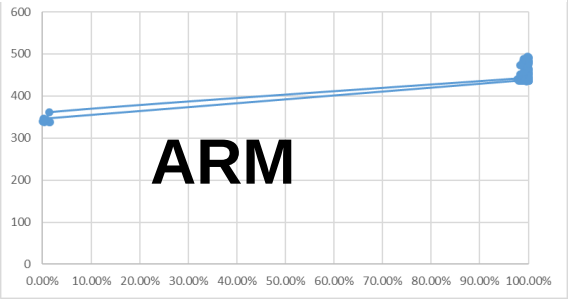


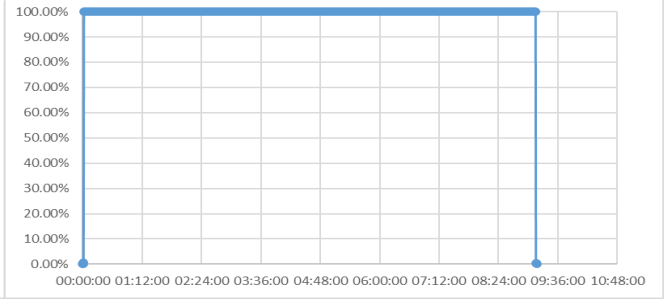


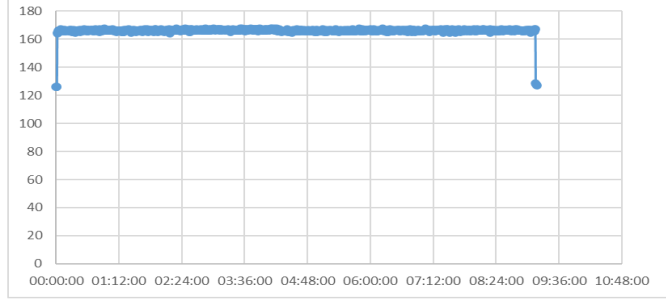






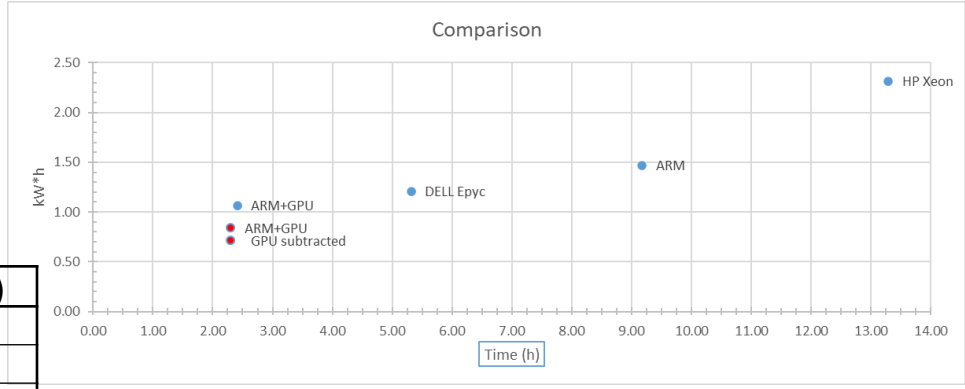






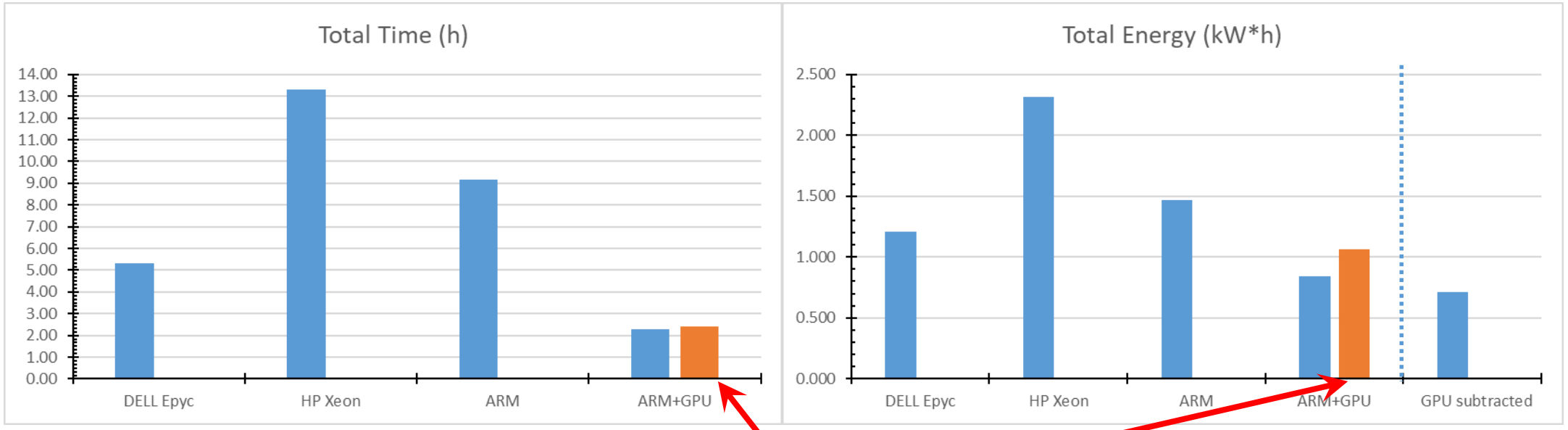
Results (C sieve)

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



Large Matrix Multiplication

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

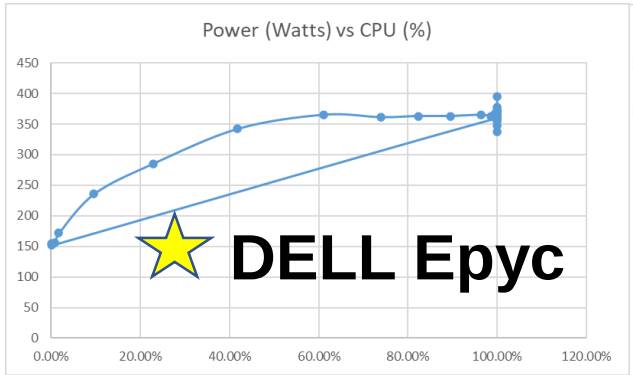
- Code is a modified version of a GitHub example **(*)**
- Compiled with the OMP library and *-mcmodel=large* flag
- Tried 2 types of matrices: integers and floating point (*double*)

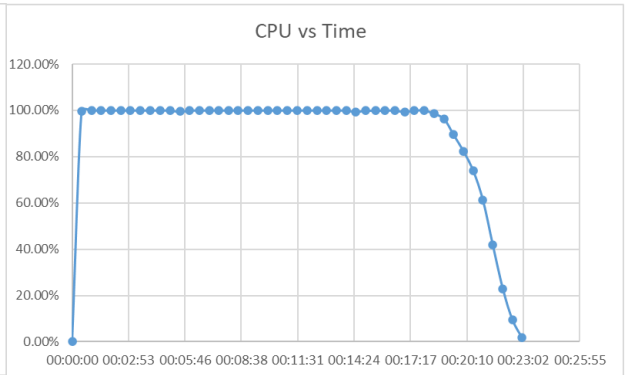
```
$ ./get2IPMI.sh > ipminfo.csv

$ gcc -fopenmp -mcmodel=large matmul.c

$ ./a.out
```

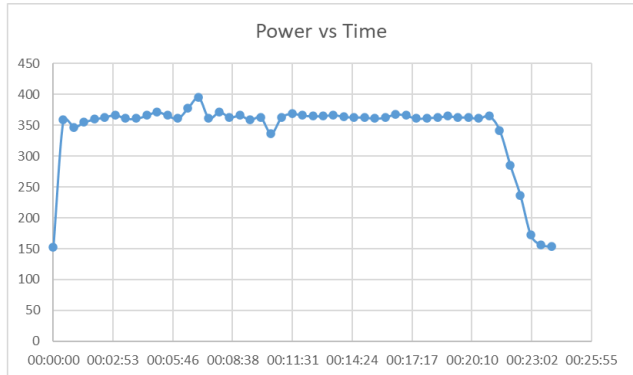
```
#include <omp.h>
...
#define N 20000
...

    #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];
```

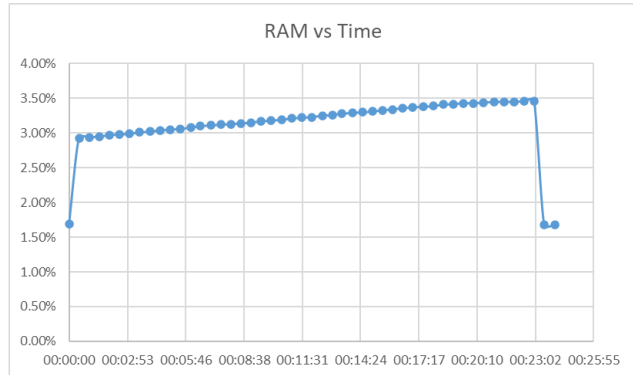
(*) https://gist.github.com/metallurgix/0dfafc03215ce89fc595

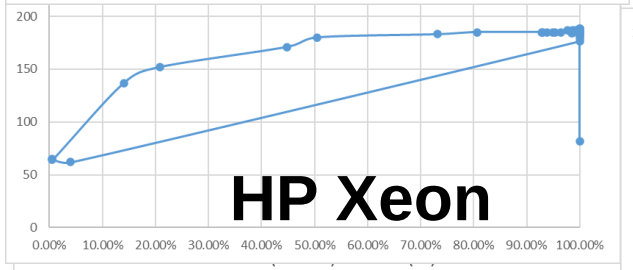
Job Profiles (mat. mult.)

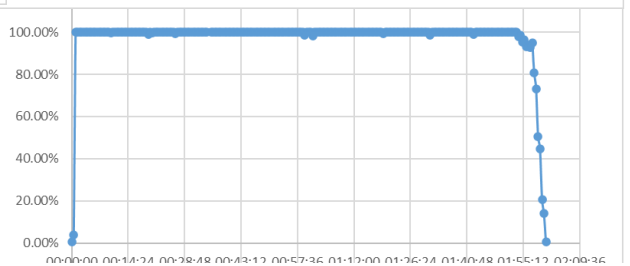


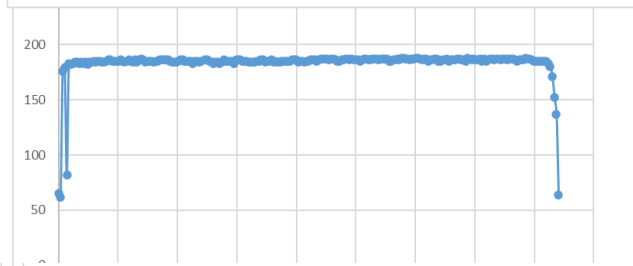


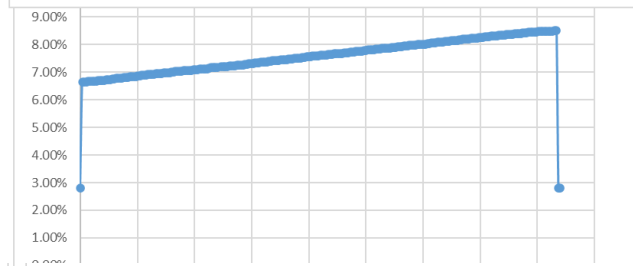


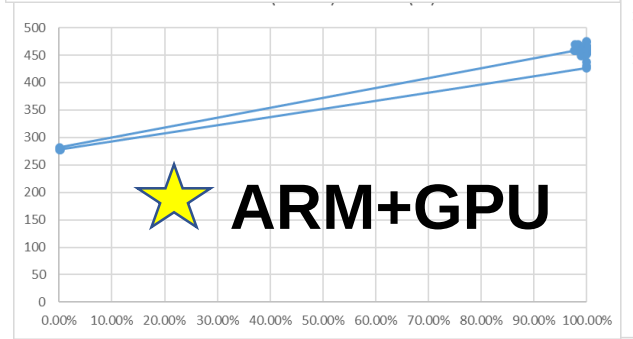


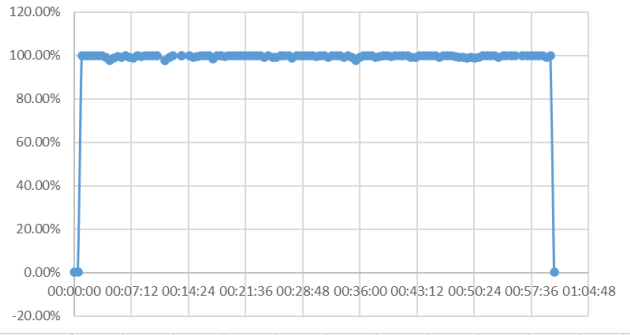


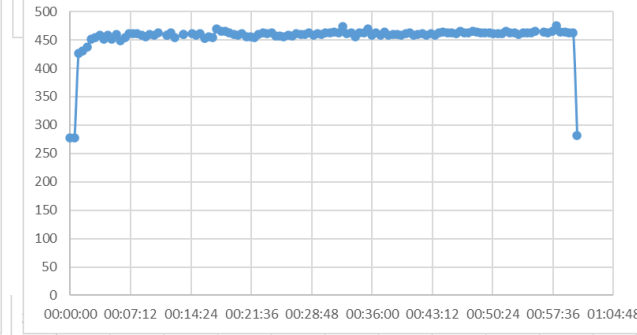


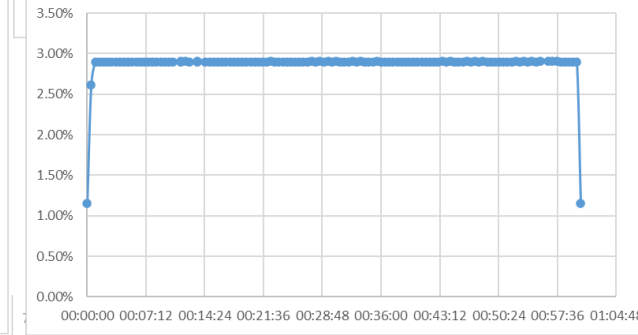


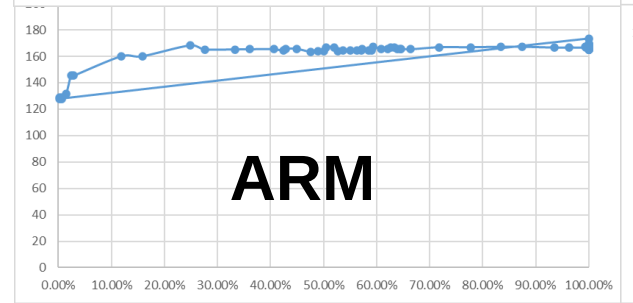


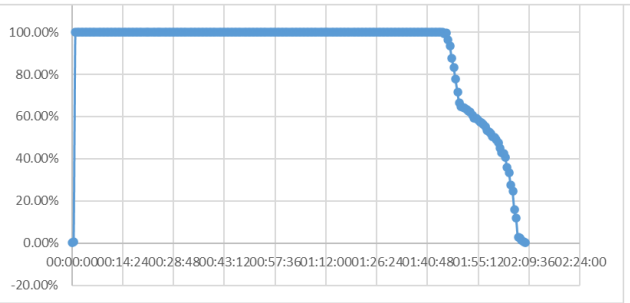


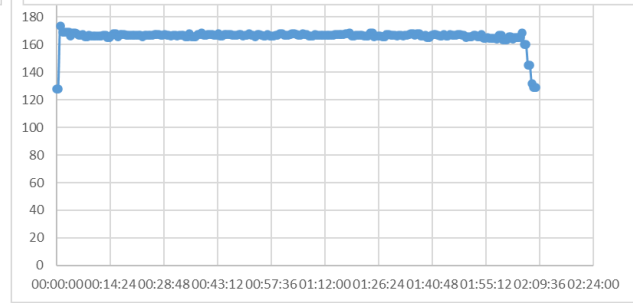


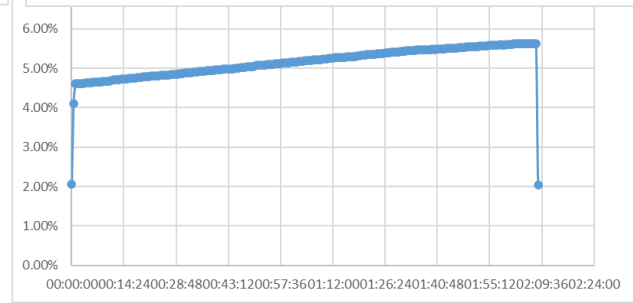












Results (mat. mult.)

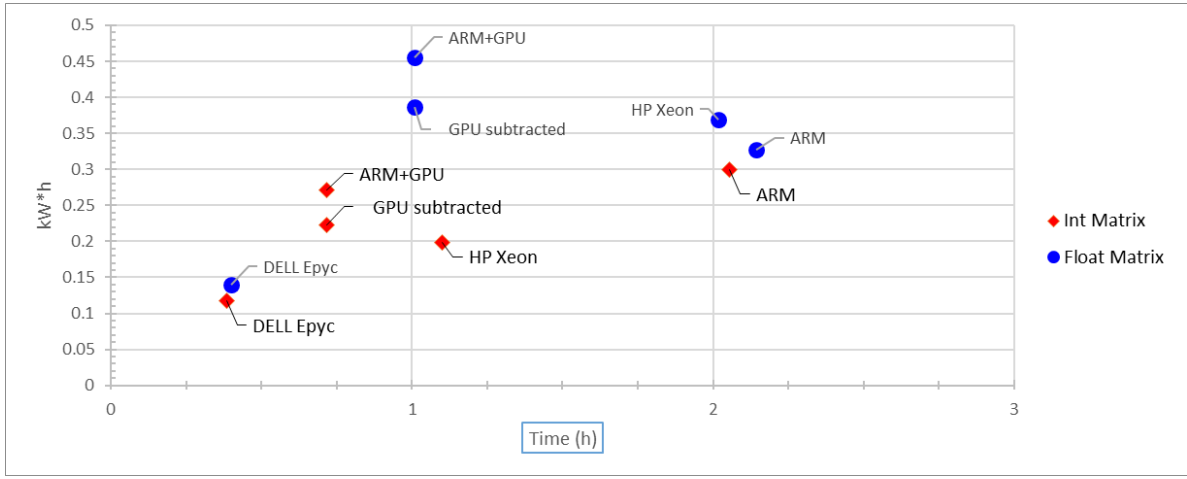
When memory usage is involved, the **DELL Epyc** machine outperforms the **ARM+GPU** in speed and energy efficiency …

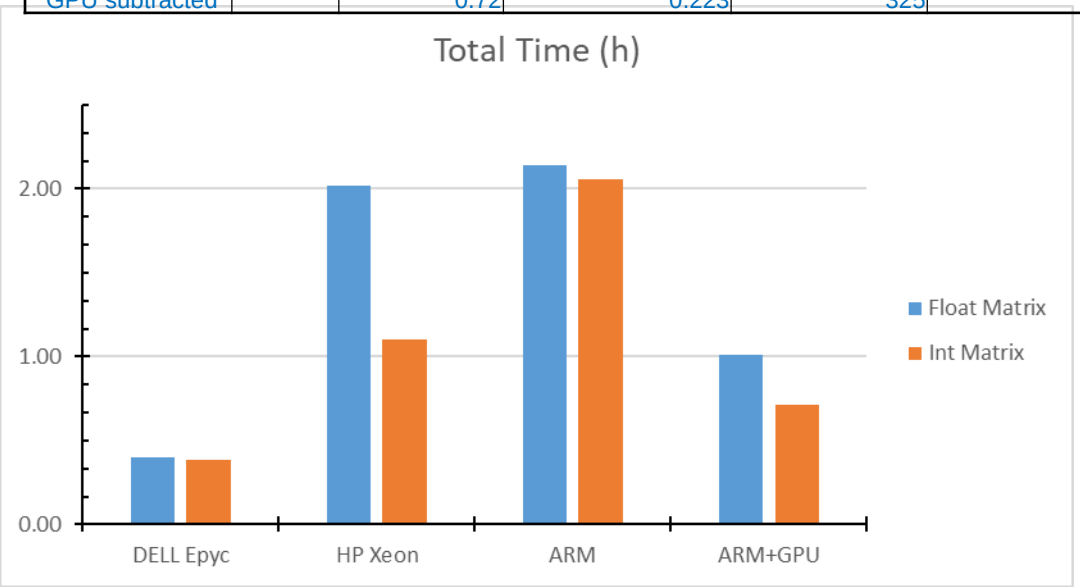
double Matrix

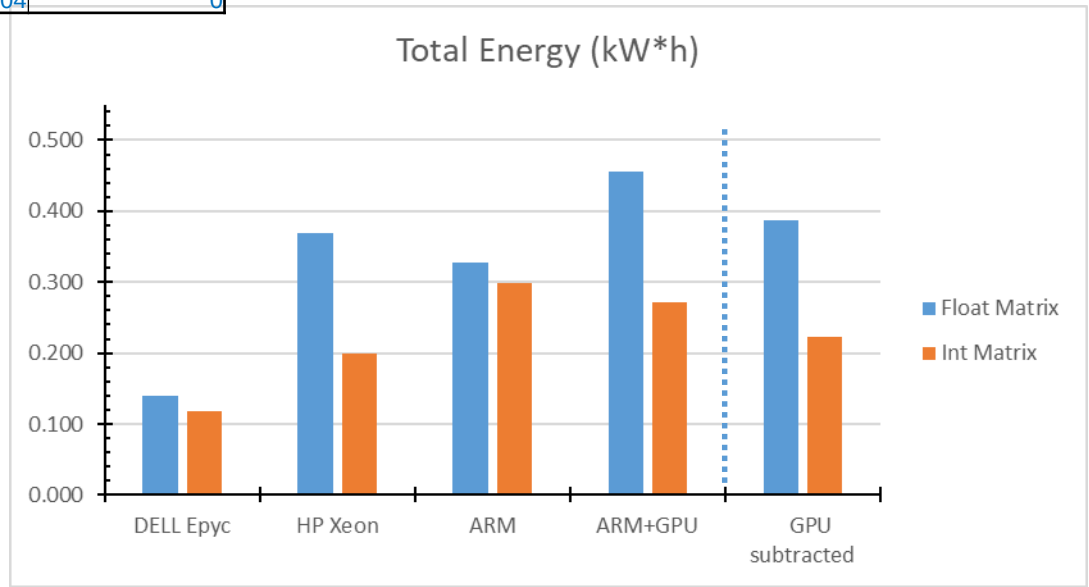
Machine	threads	Time (h)	Tot. Energy (kW*h)	Peak Power (W)	Idle (W)	GPU (W)
DELL Epyc	128	0.40	0.140	395	152	
HP Xeon	40	2.02	0.369	188	65	
ARM	224	2.14	0.327	173	128	
ARM+GPU	80	1.01	0.455	475	278	68
GPU subtracted		1.01	0.387	354	197	0

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







Power Readings, etc.

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:"`
Power (tx2mon): `tx2mon -T -d 10 -f $READFILE ; tail -2 ${READFILE} | head -1 | cut -d',' -f71`