Fine grained Energy Profiling of programs

Roblex NANA TCHAKOUTE

PhD Student
Centre de Recherche en Informatique (CRI)
Mines Paris - PSL Research University
PhD supervised by: Claude TADONKI (CRI), Petr DOKLADAL (CMM) and Youssef MESRI (CEMEF)

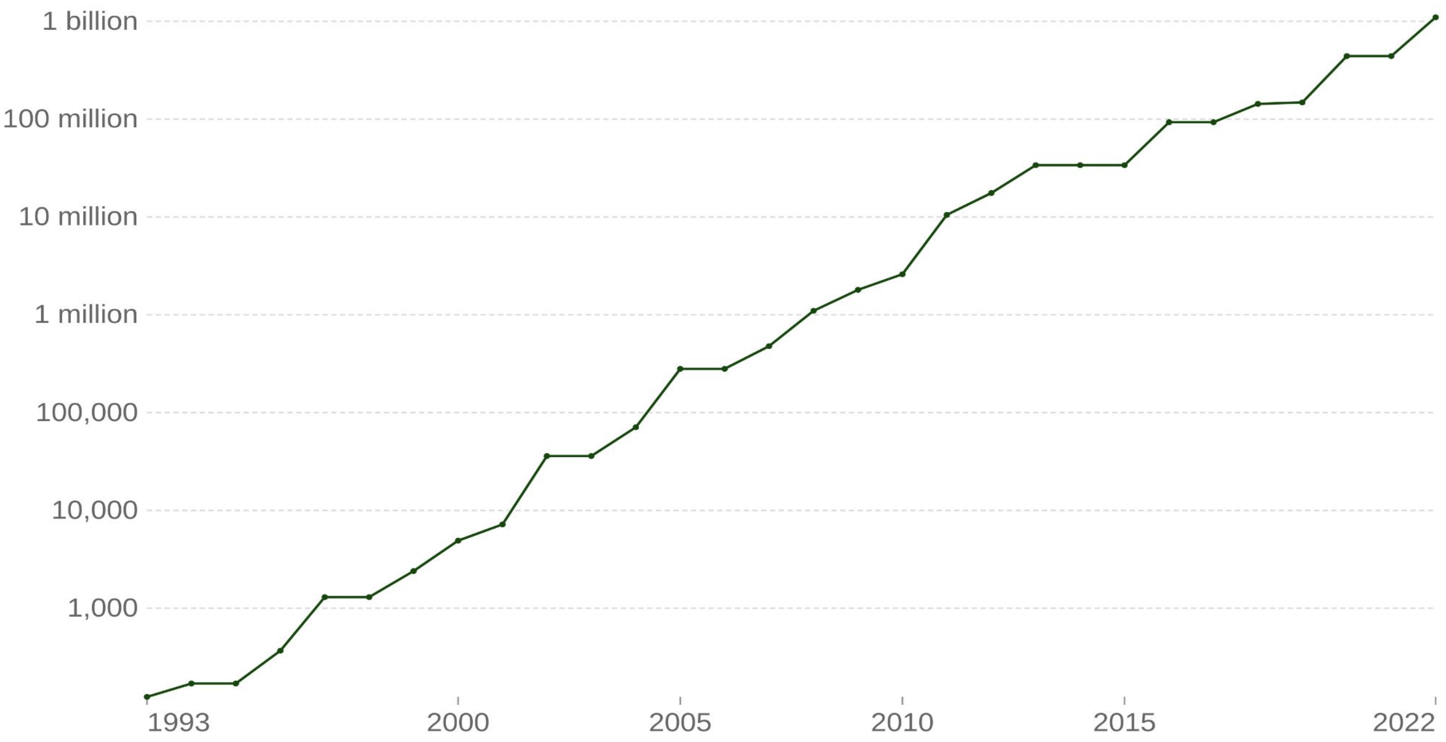
Date : 13/12/2023

contents

Context

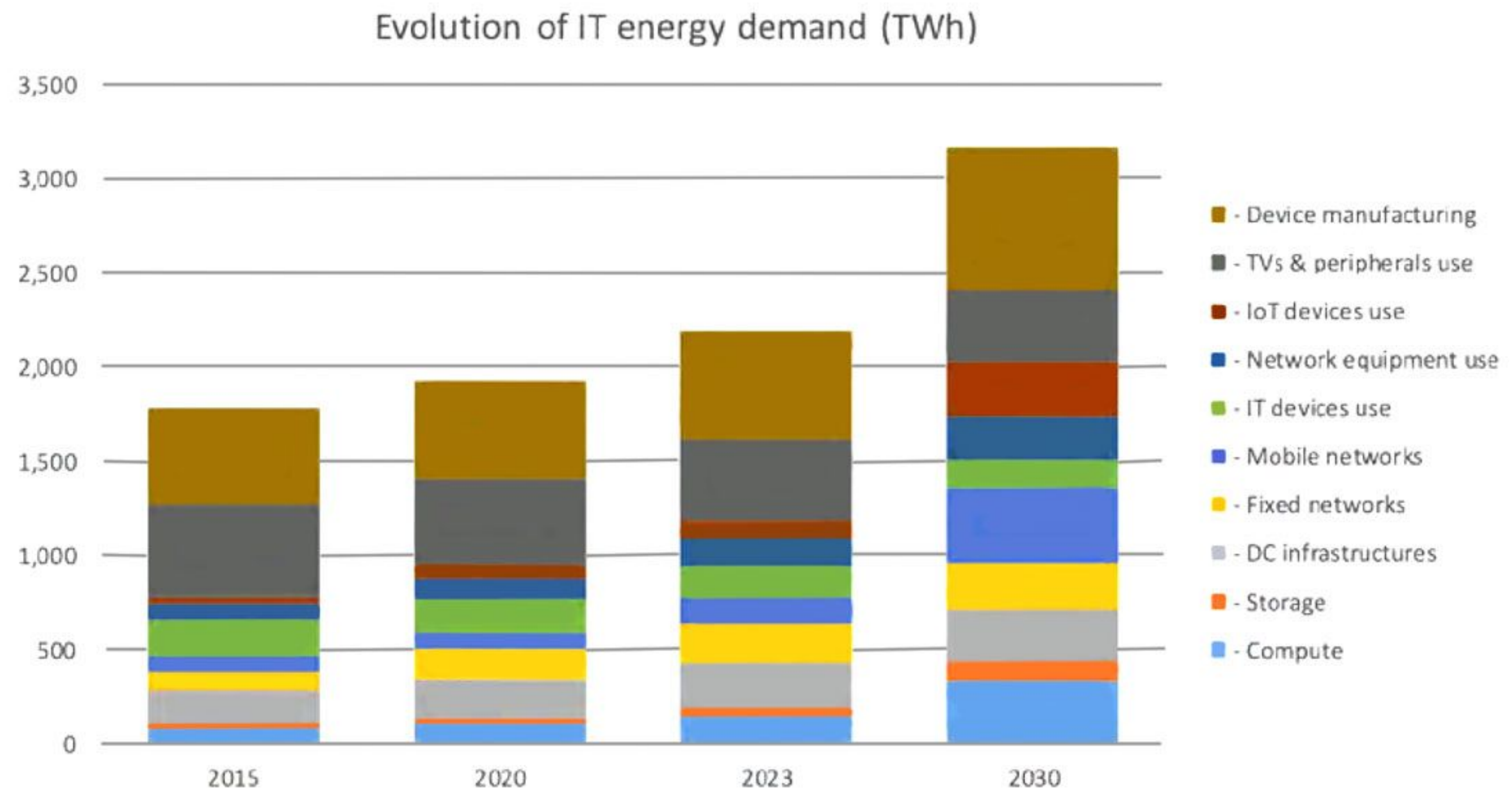
Computational capacity of the fastest supercomputers

The number of floating-point operations[1] carried out per second by the fastest supercomputer in any given year. This is expressed in gigaFLOPS, equivalent to 10^9 floating-point operations per second.



Data source: TOP500 Supercomputer Database (2023)

1. **Floating-point operation**: A floating-point operation (FLOP) is a type of computer operation. One FLOP is equivalent to one addition, subtraction, multiplication, or division of two decimal numbers.



Schneider Electric estimates that IT sector electricity demand will grow by 50 percent by 2030, reaching 3,200TWh, equivalent to 5 percent Compound Annual Growth Rate (CAGR) over the next decade. | © Image: Schneider Electric

Few facts

- **Training AI :** Its estimated energy consumption due to training GPT-3 is 1287 MWh and its carbon emissions are 552 tCO_2e *(tons of CO_2 equivalent emissions)*.
    - equivalent to driving 112 gasoline powered cars for a year

- **Inference AI :** BLOOM, consumed 914 kWh of electricity and emitted 360 kg for 18 days where it handled 230,768 requests ***(roughly 1.56 gCO_2e per request)***
    - 350 kgCO_2 = 1/3 Paris-New-York return

- **Embedded devices :** The AGX Orin is currently the most powerful board from Nvidia Jetson, with up to 275 TOPS and 60W TDP.
    - On full TDP, 60Wh battery will have 1h of life time. ***(Energy constraints for battery powered systems)***

ELECTRICITY COST PER HOUR FOR THE TOP FIVE SUPERCOMPUTERS.

Machine	Peak Perf.	Power	$/KWh	Total(K$)
FRONTIER	1.685 EFLOPS	21.1MW	0.150	3.165
FUGAKU	537.2 PFLOPS	29.9MW	0.219	6.548
LUMI	428.7 PFLOPS	6.02MW	0.198	1.192
LEONARDO	255.7 PFLOPS	5.61MW	0.561	3.147
SUMMIT	200.8 PFLOPS	10.1MW	0.150	1.515

CO_2 PER HOUR FOR THE TOP FIVE SUPERCOMPUTERS.

Machine	Peak Perf.	Power	Kg(CO_2)/KWh	Kg(CO_2)
FRONTIER	1.685 EFLOPS	21.1MW	0.379	7 997
FUGAKU	537.2 PFLOPS	29.9MW	0.479	14 322
LUMI	428.7 PFLOPS	6.02MW	0.132	795
LEONARDO	255.7 PFLOPS	5.61MW	0.372	2 087
SUMMIT	200.8 PFLOPS	10.1MW	0.379	3 828

General problem

Problem ?

- Computer activities uses more energy to provide more computation power
- Carbon is the consequence of energy consumption
  - Computer use energy and not carbon
  - Carbon footprint = Energy x Carbon Intensity
- Optimizing energy use and production is the way to reduce carbon footprint

Goal of optimization : Best trade-off between *<u>“Energy-Time-Memory”</u>*

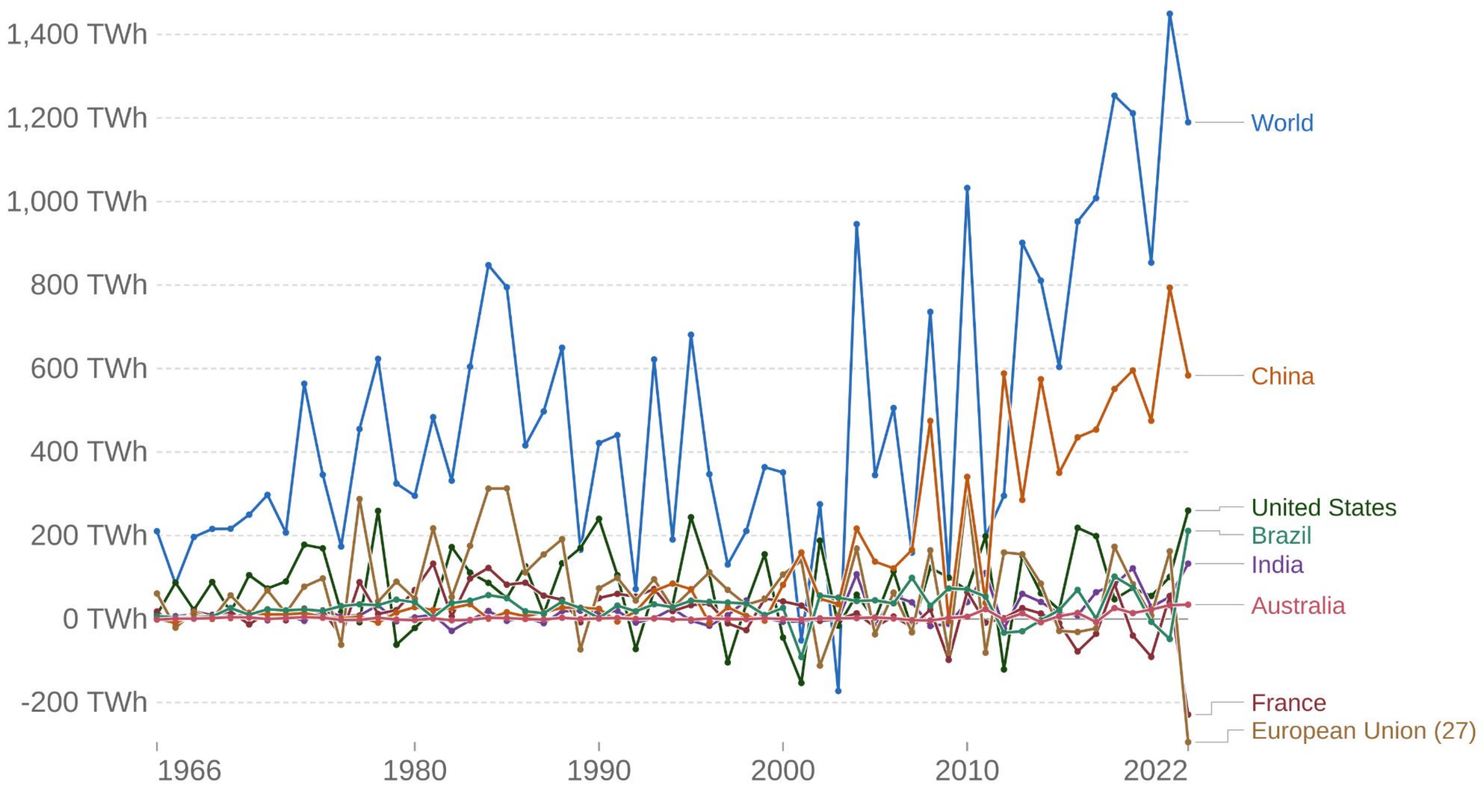
➔ 3-Dimensional optimization schema

➔ Main constraint for energy production : The source *(low-carbon sources)*

➔ Main constraint for energy use : The quantity *(should be minimized)*

Annual change in low-carbon energy generation

Shown is the change in low-carbon energy generation relative to the previous year, measured in terawatt-hours. This is the sum of energy from nuclear and renewable sources.



Data source: Energy Institute Statistical Review of World Energy (2023) OurWorldInData.org/energy | CC BY

Note: Primary energy is calculated using the 'substitution method', which accounts for the energy production inefficiencies of fossil fuels.

Energy activities in computer

Taxonomy of energy activities in computer

1 → 2 → 3 → 4 → 5

1. METRICS DEFINITION

Simples metrics : watt, joule, watt-hour, time, FLOPS, kilogram, etc.
Advanced metrics: FLOPS/watt, operation/watt, SWaP, DCIE, etc.

2. MESUREMENT & PROFILING

Out-band : wattmeters, multimeters, wattproff, HDEEM, DiG, etc.
In-bound : Integrated sensors (Intel's RAPL, NVIDIA-SMI), MSR registry, etc.

3. PREDICTION

Analytical Models (combinational techniques, instructions power consumption, etc.)
Machine learning based models prediction.

4. OPTIMIZATION

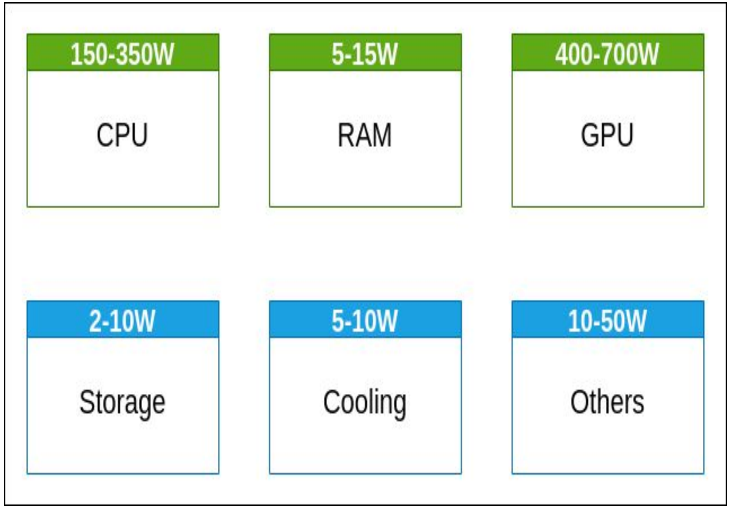
Static approaches : System design, code optimization, etc.
Dynamic approaches : DVFS, Processors states, Dynamic adaptation, etc.

5. EVALUATION & COMPARISON

Evaluate predictions : Compare predicted to measured values
Compare optimization techniques : for different optimizations methods, differents architectures, etc.

Motivations for software profiling (In-bound)

- Computer energy can be measured with Power Meter
- The most accurate way but, less helpful in optimization
- We need fine grained measurements to understand devices uses and them make optimization by devices
- We could also make program optimization



Mains energy hungry part within a modern computer server

News devices provides integrated sensors for fine grained software energy/power measurements

Energy profiling tools

SOTA Energy Profiling tools with hardware landscape

Support	Code carbon	EIT	Carbon tracker	Eco2AI	Tracarbon	Pyjoule	Perf	Likwid	PAPI	power gadget	Powertop
GPU support											
Nvidia GPU	✓	✓	✓	✓	✓	✓					
AMD GPU											
Intel GPU											
CPU and RAM supports											
Intel CPU	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AMD CPU				✓			✓				✓
RAM	✓	✓	✓	✓	✓						
OS support											
Linux	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	✓									✓	
Mac OS	✓	✓			✓					✓	
Others important characteristics											
Documentation	✓	✓			✓	✓	✓	✓	✓	✓	✓
Configurable	✓	✓									✓
code API	✓	✓	✓	✓	✓	✓		✓	✓		
AI oriented	✓	✓	✓	✓	✓						

Profiling tools

Key characteristics expected from of a profiling tool

- Programmability
  Should provides fine-grained control over energy profiling and allows developers to focus on specific parts of the codebase to optimize energy efficiency and performance (instrumentation and APIs)
- Flexibility
  To measure specific parts of the computer, allowing configurations, auto target hardware detection, and porting to other architectures.
- Standalone
  Easy to install, few dependences on others library and tools, minimum privileged rights for access
- Portability
  Compatibility across device generations, even within the same manufacturer (facilitate maintenance)
- Accuracy
  The tool does indeed measure the desired behavior and should be consistent across workloads

Reality with existing tools (why a new tool ?)

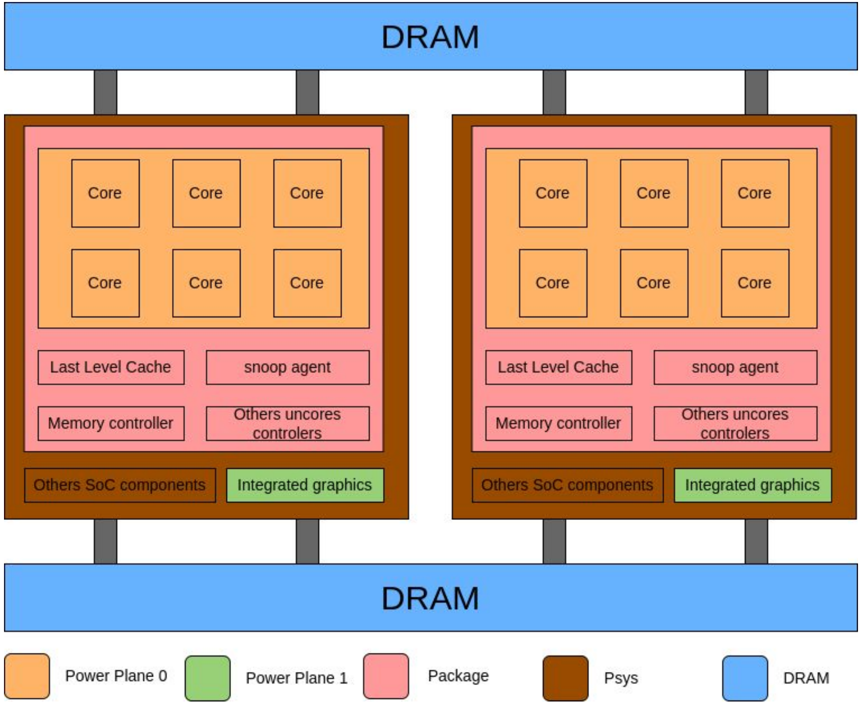
- Difficult to get, installed (*need for hand configuration*) and run
- Not reliable measurements (provide estimates - inconsistency)
- Lack of flexibility (device dependent - OS dependent)
- Lack of documentation (comprehension of the approach and outputs)

Thus our motivation to design a new energy measurement tool

Background

- Intel provided RAPL as embedded energy sensors for CPU grouped in power domains
- AMD provided similar ones for their CPU
- Nvidia GPU have Nvidia-SMI
- AMD GPU have ROCm-SMI
- And so on…

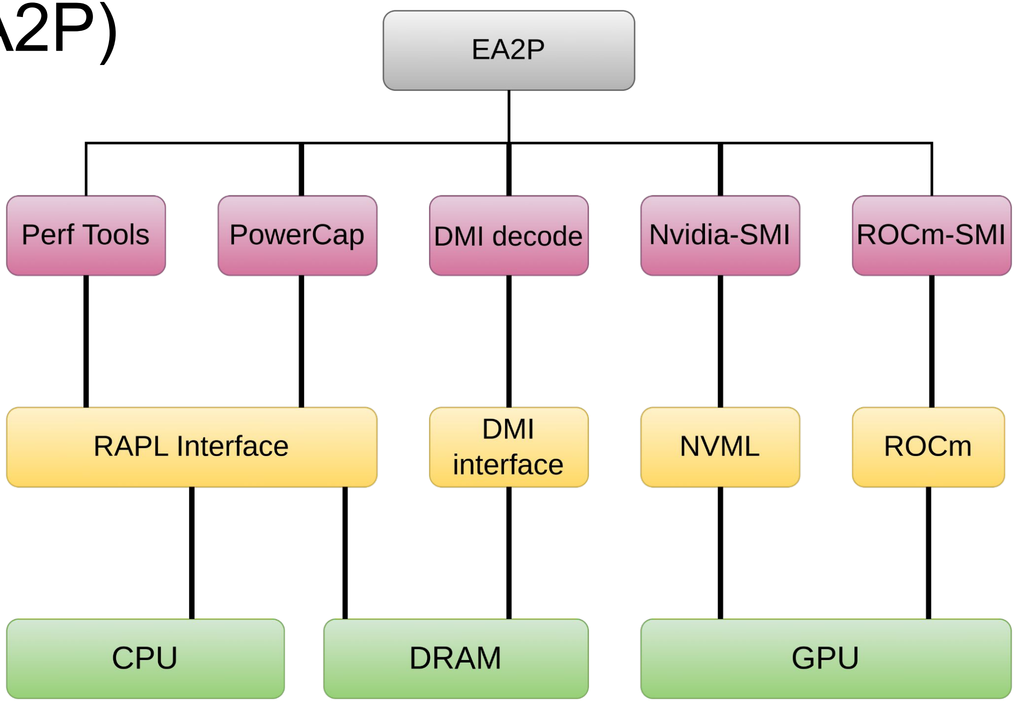
Each device manufacturer need to integrate embedded sensors into their design



Example of power domain in Intel RAPL

Presentation of our tool

Design overview of our tool : Energy Aware Application Profiler (EA2P)



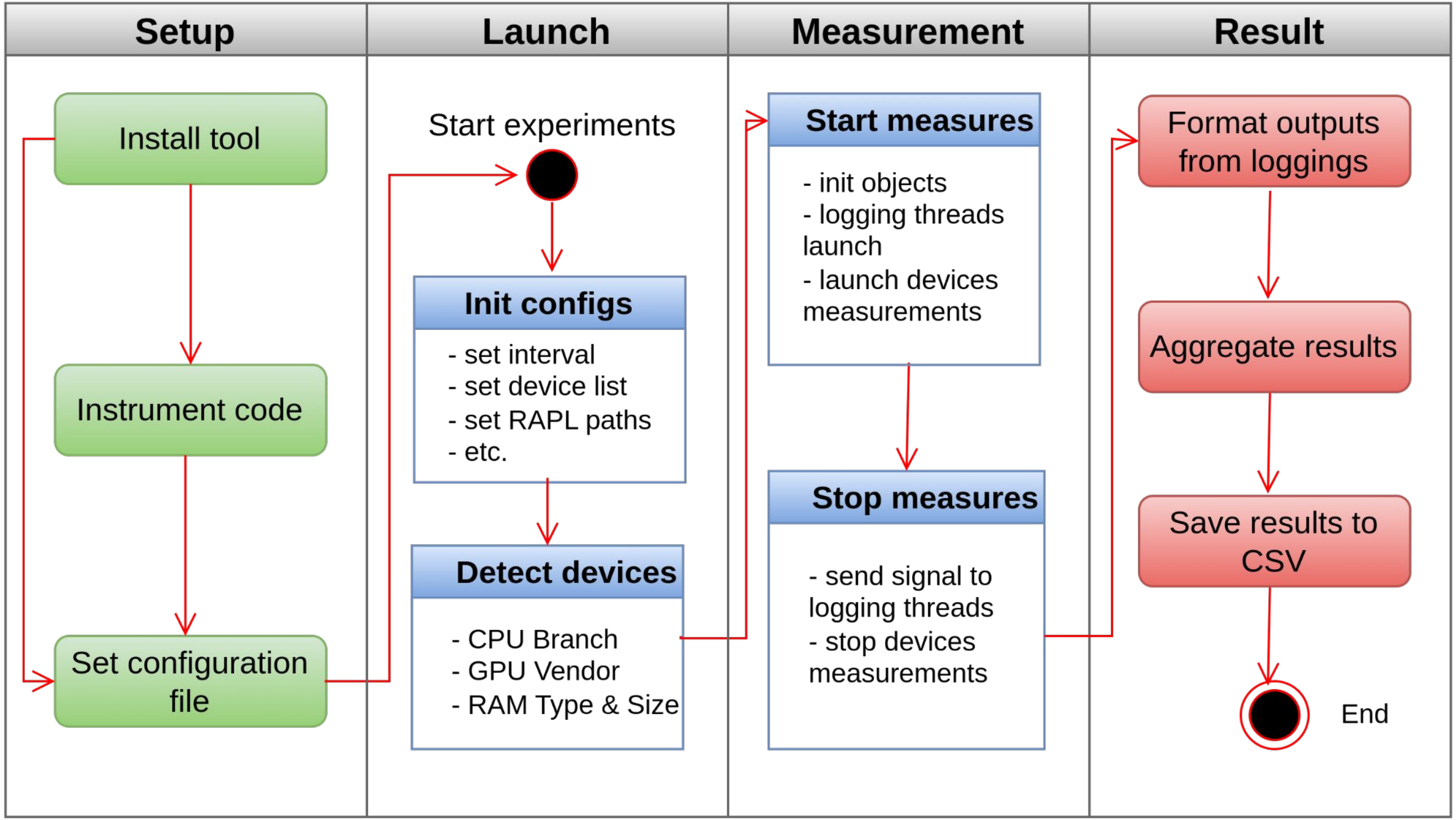
- our tool is written in Python
- we retrieve the values of the (power dedicated) registers through medium-level tools
- our tool can be used in a standalone *(external call)* form or through an API for programmability (internal call)
- our tool automatically detects the needed subtools for its execution (e.g. perf, PowerCap, …)

Few commands to access sensors values

There are **hardware sensors** that constantly get either the **power** or the **energy** of the device (or specific parts) while running and the measurements are stored into **specific registers**. (they are recent, otherwise we would go with rough and global estimations).
They are **essential** to get **power/energy informations**.

- Intel
  - `sudo sh -c 'echo -1 >/proc/sys/kernel/perf_event_paranoid'`
  - `sudo chmod -R a+r /sys/class/powercap/intel-rapl`
- AMD
  - `sudo sh -c 'echo -1 >/proc/sys/kernel/perf_event_paranoid'`
  - `perf stat —per-nodes -e power/energy-pkg/`
  - `rocm-smi --showpower      #for AMD GPU power reading`
- Nvidia
  - `nvidia-smi --query-gpu=power.draw --format=csv`
- RAM
  - `sudo chmod -R a+r /sys/firmware/dmi/tables`

Functional overview of EA2P



Sample usage in a Program

CLI API call

- **syntax** : $ python ea2p.py my_program
- **Example of call with C program** :

$ gcc -O3 -o matmul -fopenmp matmul.c
$ python ea2p.py ‘export OMP_NUM_THREADS=32;./matmul 8000’

With config file

```
from ea2p import Meter
config_path = “config.csv”
power_meter = Meter(config_path)

@power_meter.measure_power(
   package="time",
   algorithm="sleep",
)
def test_sleep(interval):
   time.sleep(interval)
test_sleep(180)          # runing
```

Code Instrumentation

```
from ea2p import Meter
power_meter = Meter()

@power_meter.measure_power(
   package="time",
   algorithm="sleep",
   data_type="",
   algorithm_params="",
)
def test_sleep(interval):
   time.sleep(interval)
test_sleep(180)          # runing
```

Sample config file

```
devices=gpu,cpu,ram
interval=0.01
output_file=experiment.csv
RAPL_FILE=/sys/class/powercap/intel/
energy_unit=wh
```

Experimental evaluation

Experimental evaluation : Goals

- **Tool Accuracy Assessment:** Validate the accuracy and precision of the energy profiling tool in measuring power consumption across different hardware components, including CPU, RAM, and GPU.
- **Energy Profiling Consistency:** Ensure the consistency of energy profiling results across multiple hardware platforms (AMD, Intel, and Nvidia).
- **Workload Characterization:** Profile various computational workloads, including CPU-intensive, GPU-intensive, and heterogeneous computing tasks, to evaluate the tool's ability to capture energy usage patterns accurately.
- **Cross-Platform Compatibility:** Assess the tool's compatibility with different hardware components (AMD and Intel CPUs, AMD and Nvidia GPUs) to ensure its versatility.

The testbed used

Applications:

- Sleep
- VGG16 with cifar10 TensorFlow dataset
- VGG16 with Stanford dogs TensorFlow dataset
- Parallel OpenMP multiplication with matrix size 8000x8000

name	Laptop	neowise	grouille	gemini
CPU name	core i9 12950HX	AMD EPYC 7642	AMD EPYC 7452	Intel Xeon E5-2698v4
GPU name	RTX 3080Ti	AMD MI50	Nvidia A100	Tesla V100
CPU TDP	55W	225W	155W (x2)	135W (x2)
GPU TDP	150W	300W x8	400W (x2)	300W (x8)
CPU threads	24	96	64 (x2)	40 (x2)
GPU memory	16GB	32GB (x8)	40GB (x2)	32GB (x8)
RAM size	32	512 GiB	128 GB	512 GB
NUMA	No	No	Yes	Yes

neowise, grouille and ***gemini*** are clusters from GRID5000. https://www.grid5000.fr/w/Grid5000:Home

Algorithms details

- VGG16 fine tuning (just train the last layer)
- Example of annotation for power measurement
- Main call for training

```python
def build_model(num_classes):
    inputs = tf.keras.layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    model = VGG16(include_top=False, input_tensor=inputs, weights="imagenet")

    # Freeze the pretrained weights
    model.trainable = False

    # Rebuild top
    x = tf.keras.layers.GlobalAveragePooling2D(name="avg_pool")(model.output)
    x = tf.keras.layers.BatchNormalization()(x)

    top_dropout_rate = 0.2
    x = tf.keras.layers.Dropout(top_dropout_rate, name="top_dropout")(x)
    outputs = tf.keras.layers.Dense(num_classes, activation="softmax", name="pred")(x)

    # Compile
    model = tf.keras.Model(inputs, outputs, name="VGG16")
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-2)
    model.compile(
        optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"]
    )
    return model
```

```python
model = build_model(num_classes=NUM_CLASSES)

@power_meter.measure_power(
    package="tensorflow",
    algorithm="VGG16",
    data_type="images",
    data_shape="(32,32,60000)",
    algorithm_params="batch_size=64,epochs=10,optimizer=Adam,loss='categorical_crossentropy'"
)
def train_model():
    model.fit(ds_train, epochs=epochs, batch_size=batch_size, validation_data=ds_test)

if __name__ == '__main__':
    train_model()
```

Energy reported values

- **psys :** Energy of the system on chip *(motherboard energy like in BMC counters with IPMI tools)*
- **package :** The CPU domain *(the CPU chip energy)*
- **uncore :** The integrated GPU energy of the package
- **cores :** The total consumption of all CPU cores of the package
- **gpu :** The consumption of GPU devices *(like Nvidia, AMD, ..)*
- **ram :** The energy of RAM domains
- **time :** The CPU elapsed time of application or instrumented code

Application	tool	package (Wh)	ram (Wh)	time (sec)
sleep	perf	2.27407	1.34291	183.787
	EA2P	2.1912	1.32991	180.274
VGG16 CIFAR-CPU	perf	27.62617	5.21861	464.698
	EA2P	28.52879	5.4077	495.096
VGG16 CIFAR-GPU	perf	1.61851	0.51481	68.425
	EA2P	1.21921	0.38869	52.459

CPU and DRAM validation on intel server "gemini" (Intel CPU)

Application	tool	package (Wh)	ram (Wh)	time (sec)
sleep	perf	4.78517	/	185.138
	EA2P	4.65467	4.85333	180.545
VGG16 CIFAR-CPU	perf	45.28731	/	557.001
	EA2P	45.57702	14.24	574.154
VGG16 CIFAR-GPU	perf	1.61832	/	45.058
	EA2P	1.21736	0.96	33.888

CPU and DRAM validation on AMD server "grouille" (AMD CPU)

Application	tool	cores (Wh)	uncore (Wh)	package (Wh)	psys (Wh)	ram (Wh)	time (sec)
Sleep	perf	0.00809	0.00048	0.14932	0.52005	/	180.029
	EA2P	0.008	0.00048	0.14917	0.52087	0.03116	180.192
VGG16 CIFAR-GPU	perf	0.08935	0.00138	0.2742	2.78056	/	72.626
	EA2P	0.05674	0.00132	0.22923	2.6726	0.01456	66.903
VGG16 CIFAR-CPU	perf	3.71593	0.00764	5.94994	11.0017	/	1476.905
	EA2P	3.69657	0.00783	5.95218	14.4883	0.29528	1478.121

CPU and DRAM validation on intel client "Laptop"

The energy of the whole system when no program is running can be non negligible. So take it into account in measurement as we can see with sleep test.

Application	tool	CPU (Wh)	GPU (Wh)	time(sec)
sleep	CodeCarbon	0.30538	0.98752	181.931
	EA2P	0.20417	0.82411	180.706
VGG16 CIFAR-GPU	CodeCarbon	0.22944	2.07726	67.993
	EA2P	0.23011	2.04792	67.757

GPU validation on Nvidia ("Laptop"). ***CPU is the energy of package domain***

Application	packages(Wh)	ram (Wh)	GPU0 (Wh)	GPU1 (Wh)	GPU2 (Wh)	GPU3 (Wh)	GPU4 (Wh)	GPU5 (Wh)	GPU6 (Wh)	GPU7 (Wh)	time (sec)
Sleep	2.19481	1.33308	2.17964	2.10399	2.12799	2.10432	2.10385	2.12957	2.10584	2.14317	181.038
VGG16 DOG-CPU	**28.52879**	5.4077	5.63378	5.41911	5.50514	5.41387	5.39961	5.49029	5.41896	5.52412	495.096
VGG16 DOG-GPU	1.21921	0.38869	**2.51989**	0.81177	0.81666	0.80432	0.81027	0.81626	0.80222	0.81376	52.459

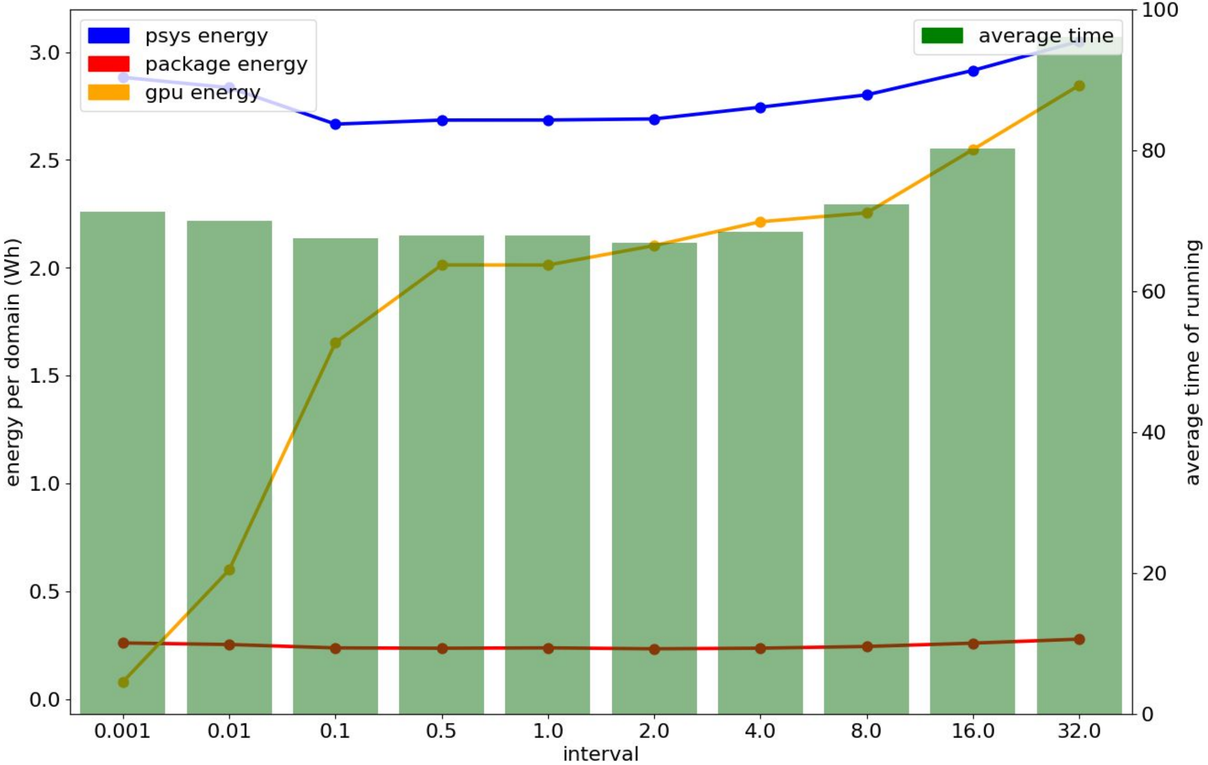
Multi GPU systems energy report "gemini" EA2P

Fine tuning VGG16 with Stanford dog dataset consume a total of more than 77 Wh for more than 9 minutes running on 80 threads Intel Xeon server with 8 Nvidia V100 GPU mounted.
The same program using GPU computing consume around 10 Wh for less than a minute of execution on the same machine. So 10x faster and 8x energy efficient

Sampling frequency influence

Application : VGG16 training on CIFAR10 with TensorFlow with batch size 64 and 10 epochs
CPU : Intel Core i9 12950HX (24 Threads)
RAM : 32 GB DDR5-4800
GPU : RTX 3080Ti, 16GB, GDDR6

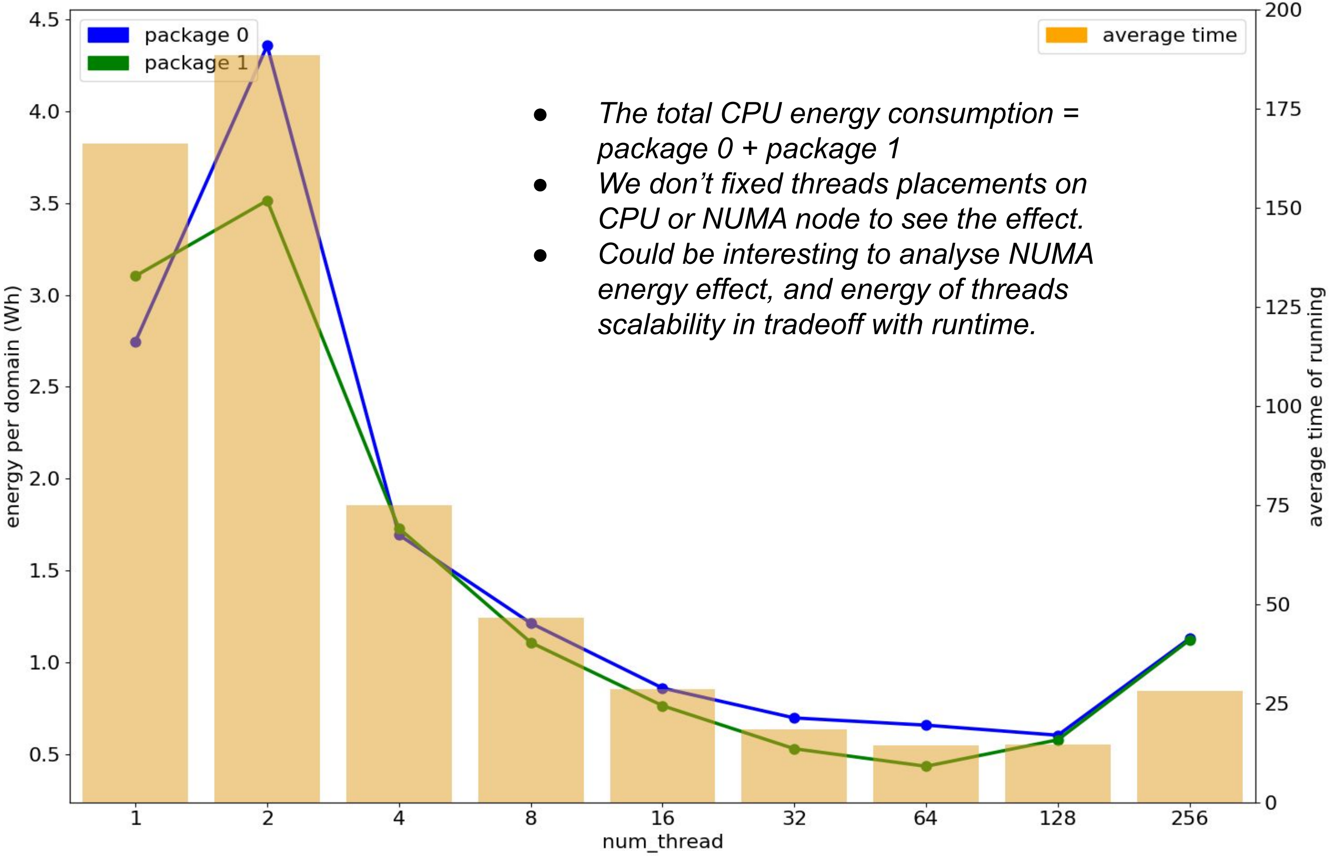
psys	package	gpu	time	interval
2.88300	0.25986	0.08007	71.28928	0.001
2.83491	0.25284	0.59910	69.90146	0.010
2.66597	0.23706	1.65259	67.43378	0.100
2.68465	0.23551	2.01294	67.88068	0.500
2.68499	0.23720	2.01260	67.91608	1.000
2.69010	0.23293	2.10269	66.90635	2.000
2.74465	0.23574	2.21340	68.46401	4.000
2.80177	0.24374	2.25440	72.26552	8.000
2.91496	0.25907	2.54813	80.16185	16.000
3.05029	0.27769	2.84596	96.12139	32.000



- Sampling frequency is the time between two query of energy values
- CPU (psys and package) energy and time are more correlated with sampling interval
- Normally, psys >= package+gpu since it's the entire board value
- GPU depend on Nvidia-smi which report the power and not the energy. So we notice consistency problem with low sampling intervals.
- Threads join from logging process is the problem of time overhead for big intervals

Multi-threading analysis

CPU : AMD EPYC 7452 (x2); **Threads** : 64 (x2), **CPU TDP** : 155W (x2) **RAM** : 128 GB; **Algorithm** : Matrix Multiplication; **Matrix size** : 8000x8000; OpenMP with - O3



- *The total CPU energy consumption = package 0 + package 1*
- *We don't fixed threads placements on CPU or NUMA node to see the effect.*
- *Could be interesting to analyse NUMA energy effect, and energy of threads scalability in tradeoff with runtime.*

package 0	package 1	time	num_thread
1.12894	1.12073	28.06818	256
0.60182	0.57692	14.53785	128
0.65672	0.43274	14.38633	64
0.69608	0.52816	18.34158	32
0.85876	0.76281	28.47718	16
1.21250	1.10733	46.66632	8
1.69303	1.72893	75.04626	4
4.35776	3.51411	188.61890	2
2.74486	3.10269	166.27935	1

Conclusion

- EA2P provide small overhead compared to Linux perf and codeCarbon tools
- provide fine grained results per device & power domains *(Intel)*
- Measurement for RAM, AMD GPU & CPU, Nvidia GPU, and Intel CPU
- Code Instrumentation API and CLI usages
- Provide Sampling frequency option to users.
- Automatic detection of device vendors and commands to use
- Possibility to select specific devices measurement *(Only subset of the system)*

Future works

- ➔ Investigate the FLOPS/Watt performance metrics
- ➔ use the tool to analyse the energy-time tradeoff of multi-threading computation
- ➔ Analyse the multi-GPU use in Deep learning training
- ➔ Apply optimization techniques (Mixed precision, quantization, etc.).
- ➔ Validate our RAM energy estimate
- ➔ Publication of a research paper for the tool

Thank you for your Attention !

Email : {roblex.nana_tchakoute, claude.tadonki, petr.dokladal, youssef.mesri}@minesparis.psl.eu

This research was supported by The Transition Institute 1.5 driven by École des Mines de Paris - PSL