E4C INTERDISCIPLINARY CENTER

Measuring and optimizing the energy consumption of a simulation

François Trahay
Maître de conférence, Télécom SudParis

e4c.ip-paris.fr

Séminaire efficacité énergétique – 30/06/2022

Goals of this talk

- *Estimating the carbon footprint of a (weather) simulation*
  - *What are the source of GHG emission ?*
  - *Measuring the energy consumption of a program*
- *Reducing the carbon footprint of a simulation*
  - *Improving the performance of an application*

Estimating the carbon footprint of a simulation

Case study

- Warning: many estimates in the next slides

- ResNet50 training on ABCI supercomputer
  - 100 epochs over 512 GPUs
  - Each epoch takes ~30s
  - 4 GPUs per server

  → (100 x 30 x 512/4) / 3600 = **107 server . hour**

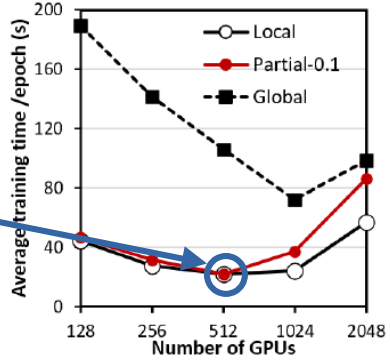


Fig. 9: Performance of ResNet50 with ImageNet-1K dataset.

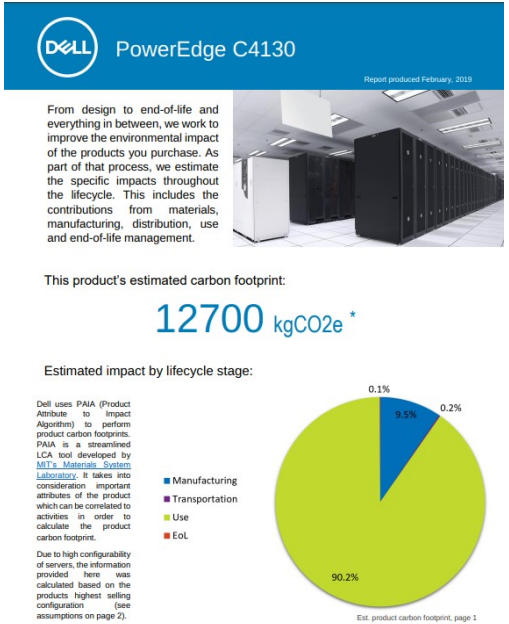
Source: *Why Globally Re-shuffle? Revisiting Data Shuffling in Large Scale Deep Learning*, IPDPS'22

Sources of carbon emissions

- Manufacturing of the supercomputer
  - Servers
  - Network (switches, cables, …) → assumption: neglictible
  - Cooling → could be reused
  - Building → could be reused
- Power usage
  - Servers
  - Network
  - Cooling

Carbon footprint of a server

- Emissions from production of a server
  - Some manufacturers (Dell, HP, Lenovo) provide *Product Carbon Footprint* documents for their servers
  - Includes the server manufacturing, transport, and recycling
  - Between 400 and 2000 kg eq CO2
  - Estimate: **1200 kg eq CO2 per server**
- Server lifespan:
  - Between 5 and 10 years
  - Estimate lifespan: **7 years**
- **Carbon footprint per hour:** 1200 / (7 x 365 x 24) = **0.02** kg eq co2/h



PowerEdge C4130

Report produced February, 2019

From design to end-of-life and everything in between, we work to improve the environmental impact of the products you purchase. As part of that process, we estimate the specific impacts throughout the lifecycle. This includes the contributions from materials, manufacturing, distribution, use and end-of-life management.

This product's estimated carbon footprint:

12700 kgCO2e *

Estimated impact by lifecycle stage:

Dell uses PAIA (Product Attribute to Impact Algorithm) to perform product carbon footprints. PAIA is a streamlined LCA tool developed by MIT's Materials System Laboratory. It takes into consideration important attributes of the product which can be correlated to activities in order to calculate the product carbon footprint.

Due to high configurability of servers, the information provided here was calculated based on the products highest selling configuration (see assumptions on page 2).

Est. product carbon footprint, page 1

Source: https://github.com/Boavizta/environmental-footprint-data

Power consumption of a supercomputer

- Servers
  - CPU: 50 – 150W (Thermal Design Power)
  - GPU: 300-400W
  - On ABCI : 2 CPUs+4 GPUs per server
    - 2 x 100 + 4 x 300 = **1400 W**
- Networking
  - difficult to quantify → ignored
- Cooling
  - Power Usage Effectiveness (PUE): 1.1 – 1.6
  - On ABCI: **1.1**
- **Power consumption:** 1400 x 1.1 = **1540 W**
- Carbon emission depends on the carbon intensity
  - In Japan: 0.416 kg CO_2 / kWh
- Carbon emission: 1.540 x 0.416 = **0.640 kg CO_2 / h**

RIKEN Center for Computational Science in Kobe

Source : https://bilans-ges.ademe.fr/documentation/UPLOAD_DOC_FR/index.htm?moyenne_par_pays.htm

Estimating the carbon footprint of a simulation

- ResNet50 training on ABCI: **107 server . Hour**
  - Manufacturing emissions: 107 x 0.02 = **2.14** kg eq co2 (3%)
  - Consumption emissions : 107 x 0,640 = **68.48** kg eq co2 (97%)
  - Total : **70.62** kg eq co2
    - ~ 366 km en voiture (Paris-Nantes)
- Reducing the carbon footprint of a simulation
  - `execution_time x ( (Manufacturing_cost / lifespan) + (server_power x PUE x carbon_intensity))`
  - ↓ carbon_intensity: moving to France (0.053 kg co2/kWh) → **7.81 kg eq co2** (~ CDG - Orly) → - 90 % CO2

    → Manufacturing : 2.14 kg (27%) / Consumption : 5.671 kg (72%)
  - ↓ execution_time: improve the program by 10 % → - 7 % CO2
  - ↑ lifespan: replace the supercomputer every 10 years → - 8 % CO2
  - ↓ server_power: reduce server energy by 10 % → - 7 % CO2

Source: https://monconvertisseurco2.fr/

Measuring the power consumption of a simulation

Available wattmeters

- Using hardware counters (Intel RAPL)
  - Measure the power consumption of CPU / Mem
  - Tools: perf, NVML, mpi-wattmeter, EZTrace
- IPMI
  - Measure the total power consumption of a server

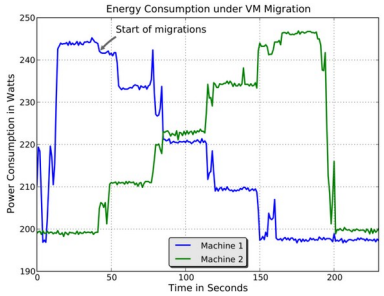
```
$ ipmitool dcmi power reading
    Instantaneous power reading:                   258 Watts
    Minimum during sampling period:                  1 Watts
    Maximum during sampling period:                744 Watts
    Average power reading over sample period:      260 Watts
```

- PDU/External wattmeters
  - Allows to collect measurements for several servers
  - Eg Grid5000

```
$ perf stat -e power/energy-pkg/,power/energy-ram/ ./bin/bt.A.x
[…]
 Performance counter stats for 'system wide':
            504.95 Joules power/energy-pkg/
             62.94 Joules power/energy-ram/
       3.003041620 seconds time elapsed
```

https://github.com/trahay/mpi-wattmeter

```
$ mpirun -np 4 mpi_wattmeter  ./bin/bt.A.4
[...]
============================
MPI-Wattmeter summary
============================
#Node:rank      Source          Joules          Watt.hour
3a401-01:0      energy-cores    216.070618      0.060020
3a401-01:0      energy-pkg      252.904602      0.070251
3a401-01:0      energy-ram      13.127563       0.003647
3a401-04:3      energy-cores    196.919067      0.054700
3a401-04:3      energy-pkg      233.507263      0.064863
3a401-04:3      energy-ram      12.370789       0.003436
Total:
#Source         total(j)        avg(j)          min(j)          max(j)
energy-cores    823.794678      205.948669      194.357422      216.447571
energy-pkg      970.565796      242.641449      230.965149      253.188782
energy-ram      50.890930       12.722733       12.370789       13.127563
```



Source : Laurent Lefèvre, EcoInfo 2012

Reducing the power consumption of a server

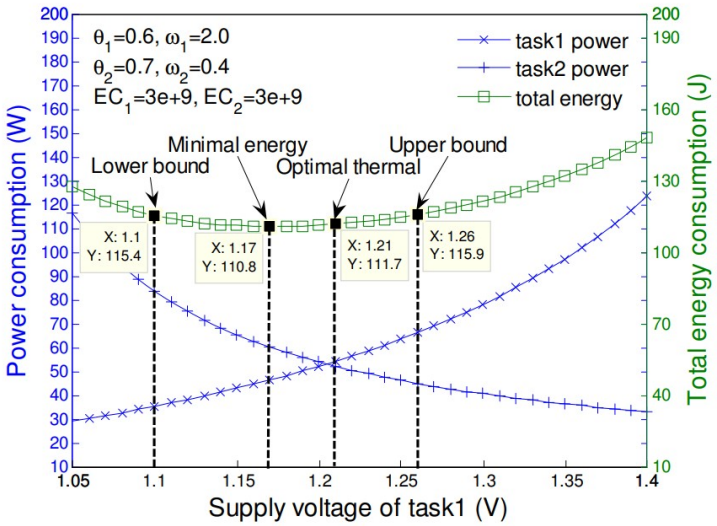
Frequency scaling

Energy = Power x Execution_Time
Power = f(frequency)

- Dynamic Voltage and Frequency Scaling (DVFS)
  - Reduce CPU frequency

    → Reduce Power consumption
    → May increase Execution_Time

  → Energy saving if Δexecution_Time < ΔPower



Source : Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In ISQED'07

Improving the power efficiency of an application

Impact of the architecture of computing resources

Power efficiency

- *Power efficiency measured in Gflops/Watt*
  - *Green 500 ranking of supercomputers*

Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	Power (kW)	Energy Efficiency (GFlops/watts)
1	29	**Frontier TDS** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	120,832	19.20	309	62.684
2	1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	21,100	52.227
3	3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	1,110,144	151.90	2,942	51.629

CPU vs GPU

- *Typical power efficiency*
  - *Nvidia A100 GPU (peak) : 9.7 TFlops / 400 Watt → 24.25 GFlops/Watt*
  - *AMD MI250X GPU (Peak) : 95.7 TFlops / 560 Watt →* ***170.9*** *GFlops / Watt*
  - *AMD EPYC 7742 CPU (peak) : 2.3 TFlops / 225 Watt →* ***10.2*** *GFlops / Watt*

- *Exploiting GPUs is crucial for power efficiency*
  - *Need to port the application to CUDA / OpenCL / …*

Exploiting vector instructions

- *AMD EPYC 7742 CPU (peak) : 2.3 TFlops / 225 Watt → **10.2** GFlops / Watt*
  - *Peak performance achieved when using vector instructions (eg AVX512, AVX2, …)*
- *Vector instructions*
  - *a Single Instruction processes Multiple Data (SIMD)*

```
for(i=0; i<size; i++) {
    C[i] = A[i] * B[i];
}
```

```
for(i=0; i<size; i+= 8) {
    *pC = _mm_mul_ps(*pA, *pB);
    pA++; pB++; pC++;
}
```

- *Compile with **-O3** !*

```
$ g++ […] -O0 -o lulesh2.0
$ ./lulesh2.0 -s 20
Elapsed time          =        3.6 (s)
```

```
$ g++ […] -O3 -o lulesh2.0
$ ./lulesh2.0 -s 20
Elapsed time          =        2.3 (s)
```

Improving the power efficiency of an application

Improving the application source code

Algorithm complexity

- Parallel complexity depends on
  - N: the problem size
  - P: the number of processors

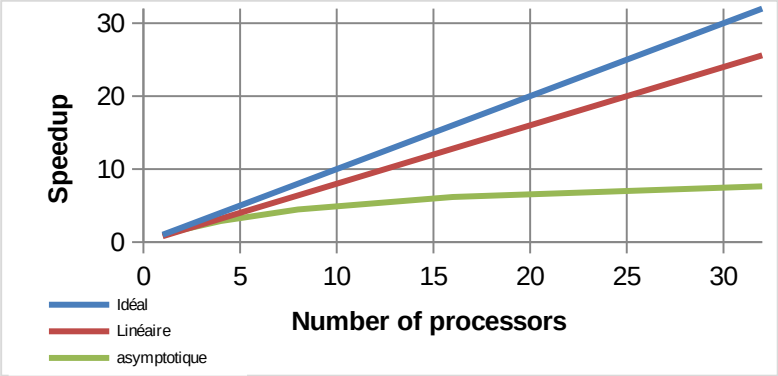
- Estimate the asymptotic complexity of the algorithm
  - If N >> P, improving the algorithm is more important than improving the parallelization
  - eg $O(N^2 / P) > O(N \log (N) / \frac{1}{2}P)$

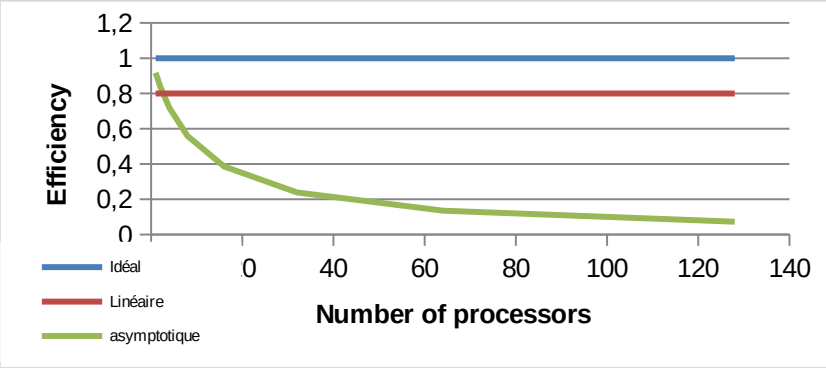
- Beware of the *hidden constant*
  - If N is small, $O(N^2) \sim= O(N \log(N))$

Scalability of a parallel program

- Speedup of a parallel program
  - D_s : duration of the sequential program
  - D_p : duration of the parallel program running on p processors
  - $S_p = D_s / D_p$: speedup on p processors
  - $E_p = S_p / p$: parallel efficiency on p processors

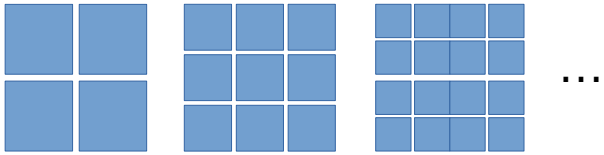
  - Good estimate of the potential for improvement



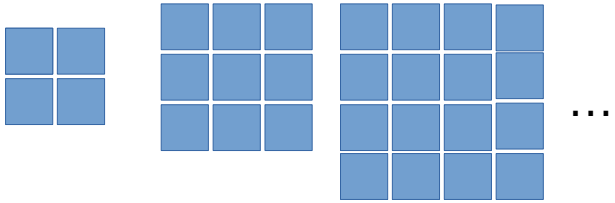


Strong scaling vs weak scaling

- Strong scaling study
  - Study how performance scales for a fixed problem size
  - How to solve problems faster ?
  - Ultimately, the computation becomes too small, and performance degrades

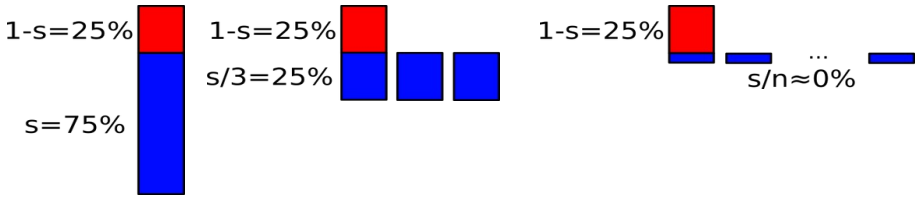


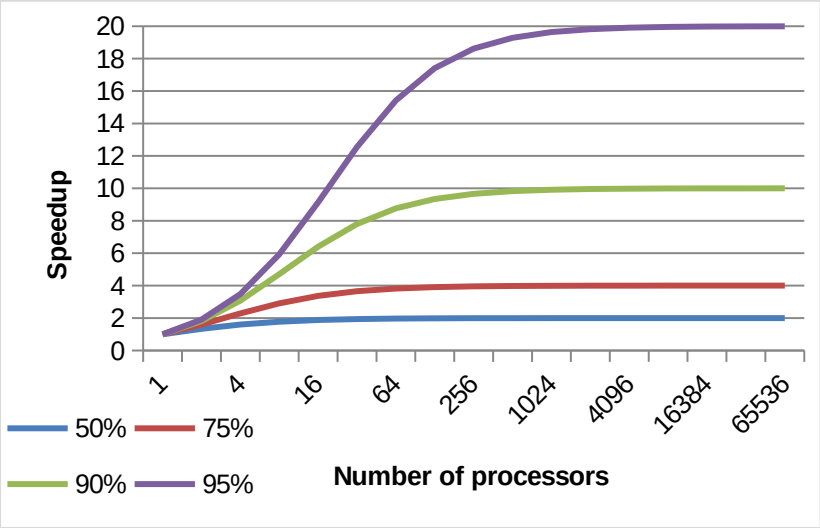
- Weak scaling study
  - Study how performance scales with a constant problem size per processor
  - How to solve bigger problems ?



Scalability of a parallel program

- Amdahl's law
  - Maximum achievable speedup
  - *s* = parallel part of the program
  - *1-s* = sequential part of the program
  - $r = \frac{1}{1 - s + \left(\frac{s}{p}\right)}$





→ **What limits the scalability is the sequential part of the program**

Sources of parallel inefficency

- Sequential parts of the program
  - Initialization/finalization (eg. reading/writing files)
  - Synchronization between threads/process
    - MPI communication
    - Mutexes

→ Reduce the amount of communication, non-blocking communication, (N-B) collective communication, ...

→ lock-free synchronization, reduction, ...

- Concurrent access to shared resources
  - Network, disk, memory, …
- Load imbalance

→ OpenMP schedule(dynamic), ...

Tools for performance analysis

Very coarse grain performance analysis `time`

- Outputs timing statistics for executing a command.
  - `Real` : time difference between the start date and the end date
  - `User` : total CPU time consumed by thread in user space
  - `Sys` : total CPU time consumed by thread in kernel space

```
$ time ./bin/dc.W.x
…
real    0m9,745s
user    0m31,930s
sys     0m3,509s
```

- Can be used for :
  - Computing speedup
  - Detecting I/O intensive applications (if `sys` is high)
  - Detecting a lack of parallelism (`user` should be roughly `real*nprocs`)

Coarse grain performance analysis
Profiling tools (eg `perf`)

- Show which functions takes most of the CPU time
- Collecting samples
  - Use the CPU sampling mechanism to know which instruction is being executed
  - Can record the callgraph (see `-g`)
- Many other cpu profilers exist
  - `gprof, oprofile, valgrind, …`
- Profilers for parallel applications
  - `mpiP, ARM MAP, ...`

```
$ perf record ./bin/dc.W.x
…
[ perf record: Woken up 21 times to write data ]
[ perf record: Captured and wrote 5,637 MB perf.data (147114 samples) ]

$ perf report
```

```
Samples: 152K of event 'cycles', Event count (approx.): 133697331175
Overhead  Command  Shared Object        Symbol
  33,34%  dc.W.x   dc.W.x              [.] KeyComp
  27,02%  dc.W.x   dc.W.x              [.] TreeInsert
   6,45%  dc.W.x   dc.W.x              [.] WriteViewToDiskCS
   6,24%  dc.W.x   libc-2.31.so        [.] _IO_fread
   4,96%  dc.W.x   libc-2.31.so        [.] _IO_fwrite
   3,13%  dc.W.x   libc-2.31.so        [.] __memmove_avx_unaligned_erms
   1,84%  dc.W.x   [kernel.kallsyms]   [k] copy_user_enhanced_fast_string
   1,78%  dc.W.x   dc.W.x              [.] SelectToView
   1,07%  dc.W.x   [kernel.kallsyms]   [k] do_syscall_64
   1,01%  dc.W.x   libc-2.31.so        [.] _IO_file_xsgetn
   0,94%  dc.W.x   libc-2.31.so        [.] _IO_file_xsputn@@GLIBC_2.2.5
   0,78%  dc.W.x   dc.W.x              [.] RunFormation
   0,43%  dc.W.x   [kernel.kallsyms]   [k] clear_page_erms
   0,35%  dc.W.x   [kernel.kallsyms]   [k] syscall_return_via_sysret
   0,34%  dc.W.x   [kernel.kallsyms]   [k] entry_SYSCALL_64
   0,25%  dc.W.x   [kernel.kallsyms]   [k] __list_del_entry_valid
```

```
$ mpirun -np 8 --env LD_PRELOAD=libmpiP.so ./lulesh2.0  -s 10
[…]
$ cat ./lulesh2.0.8.50835.1.mpiP
Task    AppTime    MPITime     MPI%
   0       4.37      0.309     7.08
   1       4.37      0.321     7.37
   2       4.36      0.383     8.77
   3       4.36      0.648    14.85
   4       4.36      0.529    12.11
   5       4.36      0.526    12.06
   6       4.36      0.603    13.81
   7       4.37      0.518    11.88
   *       34.9       3.84    10.99
```

https://github.com/LLNL/mpiP

Fine grain performance analysis
clock_gettime

- Manual timing of parts of the code
  - Precise timing/variation measurement
- Need a clock
  - `gettimeofday()`
    - Precision : 1µs, overhead : 20 ns
  - `clock_gettime()`
    - Precision : 1 ns, overhead : 10-200 ns
  - `MPI_Wtime()`
    - Can return a global clock

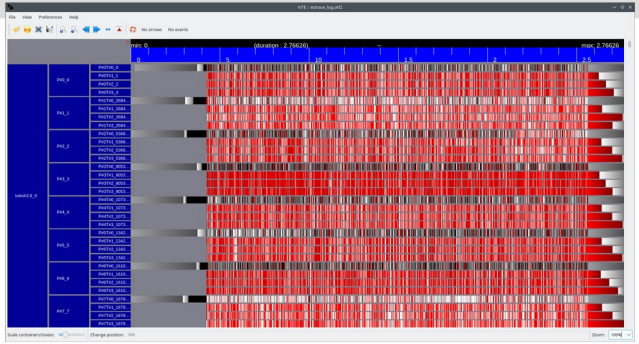
```
/* collect samples for all the threads */
for(int i=0; i<nthreads; i++) {

  size_t read_size, write_size;
  copied_size = 0;
  get_tick(&t1);
  numap_sampling_read_stop(thread_ranks[i].sm);
  get_tick(&t2);
  __process_samples(thread_ranks[i].sm, ACCESS_READ);
  get_tick(&t3);
  read_size = copied_size;
  numap_sampling_resume(thread_ranks[i].sm);
  get_tick(&t4);


  read_stop_duration += time_diff(t1, t2);
  process_samples_duration += time_diff(t2, t3);
  sampling_resume += time_diff(t3, t4);
```

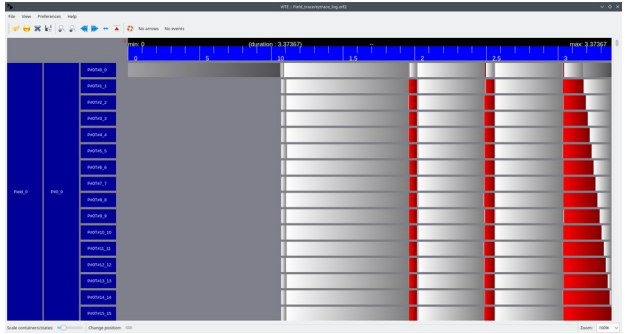
Fine grain performance analysis

Tracing tools (eg. VTune, EZTrace, …)

- Intel VTune, Scalasca, EZTrace, Extrae, ...
- Dynamic representation of the program behavior
- Execution trace :
  - Timestamped list of events
- Post-mortem analysis
  - Using visualization tools (eg. ViTE)
  - Using trace analysis tools
- Pitfalls
  - Trace file can be huge (eg. GB)
  - May alter the program performance/behavior

```
$ mpirun -np 8 eztrace -t mpi ./lulesh2.0  -s 10
[…]
$ vite lulesh2.0_trace/eztrace_output.otf2
```





Conclusion

Conclusion

- Power consumption accounts for 70 % of a simulation carbon footprint
- Many problems can degrade the performance of applications
- Many tools/techniques exist to investigate performance problems

- Open for collaboration
  - As part of a research project
    - Parallel & Distributed Systems group at Télécom SudParis
      https://www.inf.telecom-sudparis.eu/pds/
  - Call for master student projects
    - Contact : François Trahay -- https://trahay.wp.imtbs-tsp.eu/

Performance analysis tools

- Profiler
  - Arm MAP https://www.arm.com/products/development-tools/server-and-hpc/forge/map
  - mpiP https://github.com/LLNL/mpiP
  - perf

- Tracing tools
  - Intel Vtune https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html
  - Scalasca https://www.scalasca.org/
  - Extrae https://tools.bsc.es/extrae
  - EZTrace https://eztrace.gitlab.io/eztrace/
  - Darshan IO tracer https://www.mcs.anl.gov/research/projects/darshan/

- Other useful links
  - Performance Optimisation and Productivity (A Centre of Excellence in HPC) https://pop-coe.eu/partners/tools
  - PRACE training portal https://training.prace-ri.eu/
  - Estimation de l’empreinte carbone d’une heure.coeur de calcul https://hal.archives-ouvertes.fr/hal-02549565v4/document