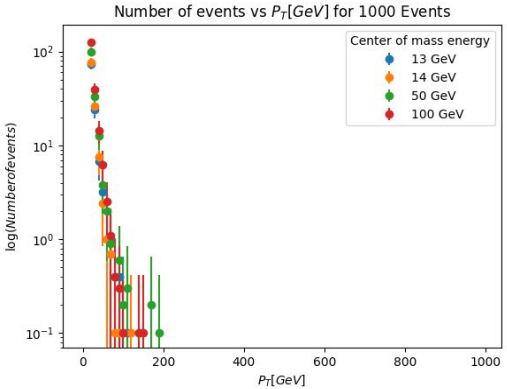
Cost of Running HEP Software

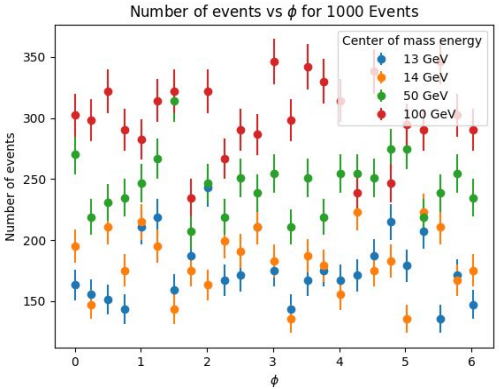
Akshat Gupta

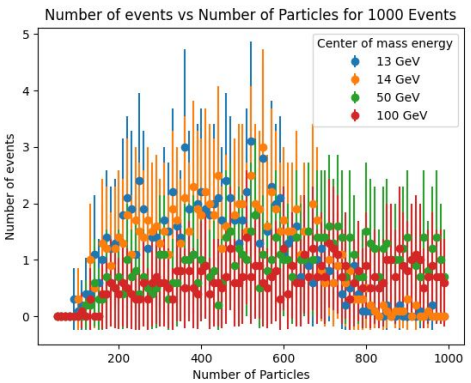
Progress Introduction

- All computations for 100 - 10000 were completed, 100000 events were not explored due to the lack of time
- New Graphs exploring the distributions of the different number of events in the different Centre of Mass (COM) Energies were made
- Scaphandre was explored on the same machine using Ubuntu installed on drive instead of WSL2 (reason for it is given later)
- 1000 Events were explored using the antikt-python package and antikt-julia package for both CodeCarbon and Scaphandre on the same environment

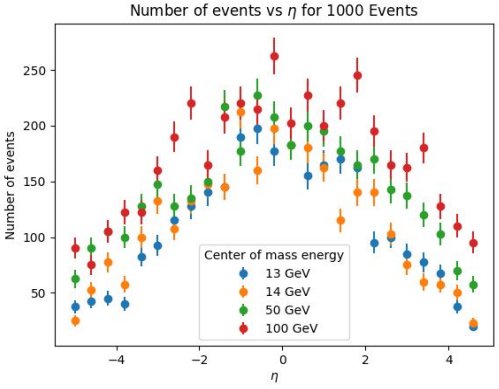
Dataset Distribution Visualisation

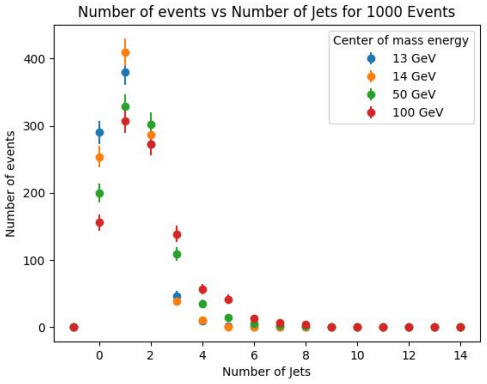






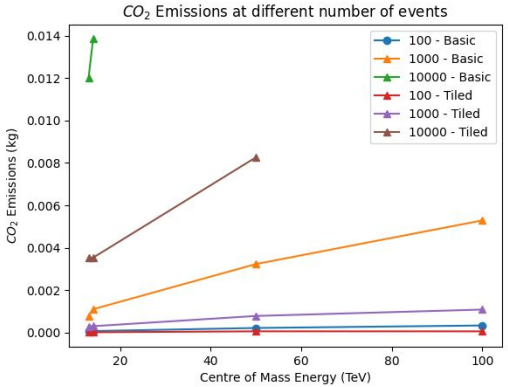
- Distributions are as expected
- The phi graph ranges from 0 to 2π instead of $-\pi$ to π
- The number of jets per event is as expected
- The pT graph shows much similarity between the two but is abruptly cut after 200 GeV
- These are generated using Pythia 8 with HardQCD on and Jet Min pT set to 20 GeV

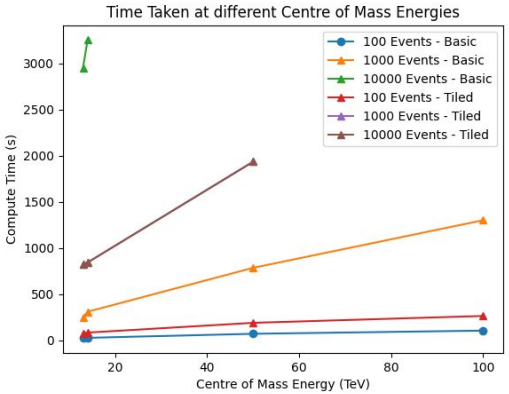


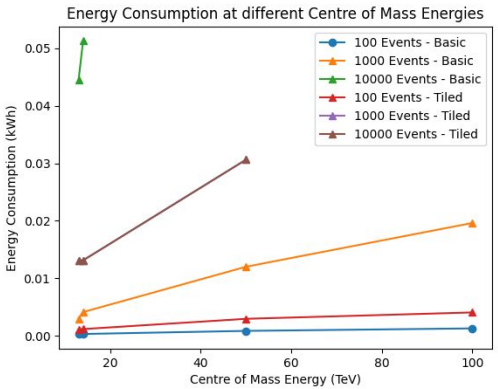


CodeCarbon - WSL2

- CodeCarbon is a package that monitors the energy usage of a process or the machine while that specific process is running
- It has no way of calculating carbon impact other than the amount of CO2 emitted during processing
- As such it relies on a linear factor that relates the energy used, the time the computation ran for which finally outputs the carbon emitted in kg.
- This is confirmed by the graphs on the right
- It uses a carbon equivalent factor to calculate its footprint [1]
- This is dependent on the different countries which are obtained from the International Energy Agency (IEA) [2]



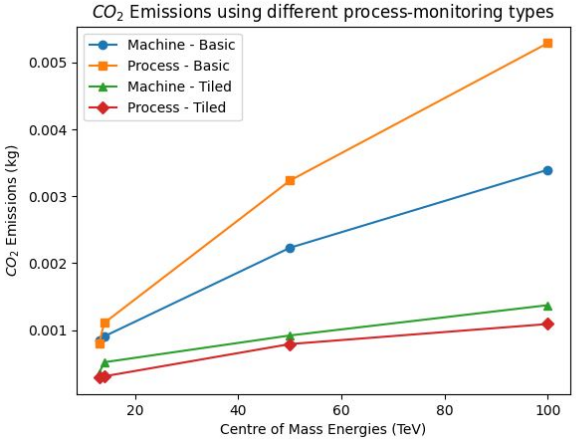


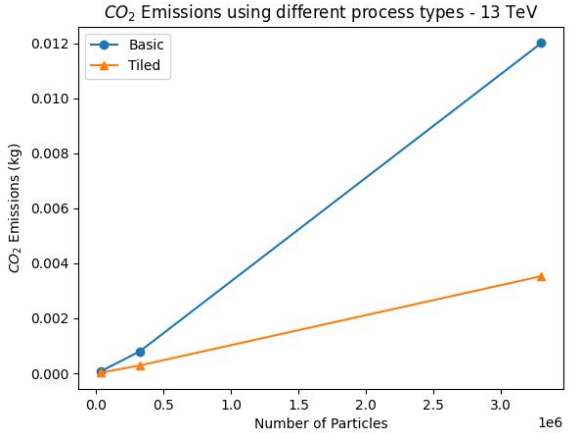


[1] https://mlco2.github.io/codecarbon/methodology.html
[2] https://www.iea.org/reports/global-energy-co2-status-report-2019/emissions

CodeCarbon - WSL2

- The difference between ‘Machine’ and ‘Process’ Monitoring Types was also explored.
- The only major difference was found to be the RAM energy usage that impacted the baseline (‘Process’) by a 25% increase
- This can be seen in the graph above
- A minor plot of the different antikt scripts was also made to confirm that the script is working as expected
- This can be seen in the graph below





Scaphandre

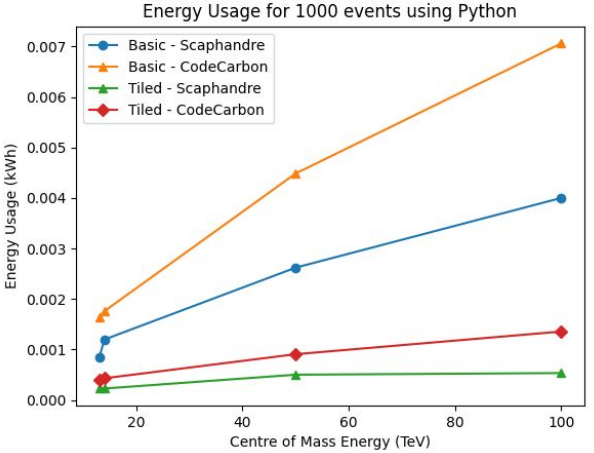
- Does not work with the initial environment (WSL2 + Windows 11)
- The compilation on windows is a bit complicated and has a few missing features. The list is incomplete.
- Required an installation of linux (dual booting) which in turn changed the environment and as such one set of events was rerun using CodeCarbon for Python and Scaphandre for Python and Julia
- Since CodeCarbon (CC) relies only on a constant factor to calculate our CO_2 Footprint, the numbers obtained from Scaphandre would be a direct drop-in in order to obtain our values
- Since Scaphandre (SC) uses accurate information reported by the CPU, it is much more accurate
- In this case, the GPU was turned off (due to missing drivers) and was not included in either CC or SC.
- However, since CC uses Nvidia Drivers to obtain the actual usage, it can be used in case we want to explore the involvement of the GPU (which is negligible).

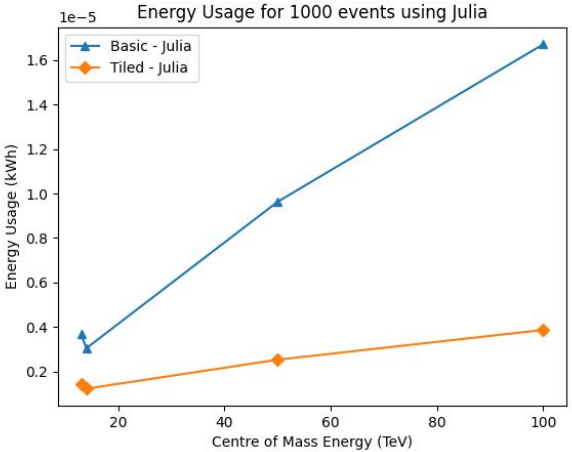
Scaphandre - Procedure

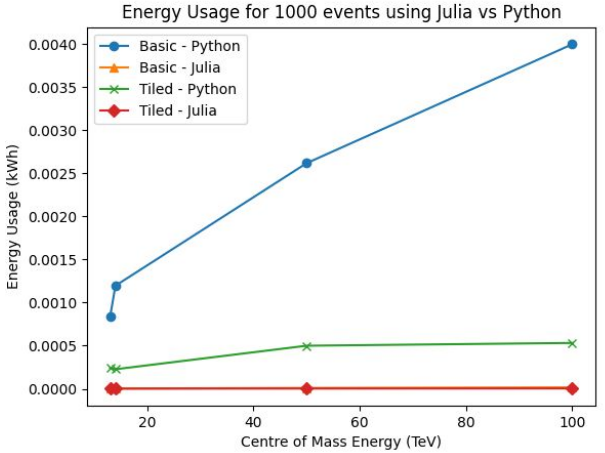
- Install Prometheus, Grafana and Scaphandre
- Set up scaphandre in Prometheus Exporter Mode
  - It is a mode that allows scaphandre to start a http server on localhost:8080 (can be changed) which Prometheus (localhost:9090) can access and understand (compatible)
- Using grafana and connecting it to Prometheus we can explore the different energy consumed in microwatts by the process (python3)
- This allows us to get an accurate result for the process
- This process was followed for 1000 Events Data for the different COM Energies
- In grafana, average of the energy consumption was noted as it fluctuates
- There is some uncertainty in the power consumption of 13/14 TeV datasets as they are processed faster than the reporting time

Scaphandre - Results

- Using just scaphandre to compare Julia vs Python, Julia is a clear winner in terms of energy usage
- We also notice that CodeCarbon has an approx 40-50% more energy usage for the same item due to using a constant for the CPU power instead of an accurate power usage
- In Julia, the tiled algorithm is the best algorithm which is to be expected
- Overall these results are acceptable and hence promote the use of Scaphandre over CodeCarbon







Conclusions

- Q1: Is Julia a better language than Python for AntiKt?
  - Yes
- Q2: How is the Carbon Footprint generally measured?
  - It uses a simple scaling that can be found online as mentioned previously
- Q3: Is the difference between different algorithms significant?
  - Yes, The basic algorithm is a lot slower than the tiled algorithm, and julia-tiled is better than all the different combinations
- Q4: Which Monitoring Software Package yields a better result?
  - Scaphandre is the better package compared to Code Carbon as it has bare metal host level energy monitoring that uses the computer's own drivers to determine - to a very good accuracy - the energy consumption
- Q5: Can we use this data to quantify the cost of running such an algorithm?
  - On its own this data is not enough to quantity such a metric as it does not take into account the amount of time need to write the code and compile it. It also does not take into account any other CO_2 emitting sources except the running of the code.
  - However, in a more holistic study, this would be represent a very important component.

Final Notes

- Main limiting factor in this experiment was the processing times.
- Scaphandre was quite difficult to install on WSL2, would want to find a good alternative for WSL2
- Uncertainties can be reduced by running the software multiple times and then getting an average
- These can also be reduced by using shorter time intervals for Scaphandre
- Next Step: finalise the repository with proper readme and notes to github repo