Université de Pau et des Pays de l'Adour

Compte rendu des travaux du
TP N°2

Mesures énergétiques des codes sources et éco-conception

04 octobre 2024

UE : Numérique Responsable

Réalisé par :

M. ABECASSIS Hugo
M. ZOUHAIR Guilhem

Encadré par :

M. NOUREDDINE Adel

Master 2 Technologie de l'Internet - Année 2024 - 2025

Table des matières

Table des figures

1 Introduction

Dans le cadre de l'Unité d'Enseignement **"Numérique Responsable"** du cursus Master 2 Technologie de l'Internet à l'Université de Pau et des Pays de l'Adour, nous avons entrepris une série de travaux pratiques visant à prendre conscience de **l'impact environnemental des technologies numériques** et de comprendre comment **optimiser cet impact**.

En effet, plus nous avançons dans notre ère numérique, plus l'impact environnemental lié aux technologies devient préoccupant. Avec la montée en puissance des appareils connectés, des centres de données et du cloud, **la consommation énergétique du domaine numérique ne cesse de croître.** De plus, l'omniprésence des technologies dans nos vies entraîne également une augmentation constante de la demande en électricité, et donc des émissions de CO2.

Il devient alors impératif de réfléchir à des stratégies pour **réduire cette empreinte écologique.**

L'objectif des travaux pratiques que nous menons est de **sensibiliser** à ces enjeux tout en explorant des solutions concrète pour **optimiser** la consommation d'énergie des dispositifs électroniques.

2 Objectifs du TP

L'objectif du TP N°2 est similaire à celui du TP N°1. Une des principales différence est qu'au lieu d'utiliser l'outil PowerSplyCli[1], nous utiliserons pour faire nos mesures l'outil PowerJoular[4] et JoularJX[3]

3 Mesure au repos

Comme pour le TP n°1, nous avons **mesuré la consommation de la puissance de l'ordinateur au repos** afin d'obtenir une ligne de base de consommation énergetique au repos.

Pour ce faire, nous avons utilisé l'outil **PowerJoular**[4] afin de relever la consommation énergétique pendant 60 secondes.

Nous avons ensuite utilisé ces données, au format CSV, pour générer plusieurs graphiques permettant de faire des analyses.

Les fichiers (programme, donnée CSV, graphiques...) sont accessible ici : `https://github.com/Guiguizou/numerique_responsable/tree/main/tp2/part1`.

En exécutant notre programme Python avec les données des mesures de la puissance au repos contenues dans le fichier "idle.csv", nous obtenons le résultat suivant :

— **Puissance moyenne :**

$$\approx 18.93W$$

— **Energie totale consommé**

$$\approx 1113.43Joules$$

```
PS C:\Users\guigu\Desktop\tp1\numerique_responsable\tp2\part1> python .\script_idle.py
Consommation moyenne au repos : 18.93 W
Énergie totale consommée : 1113.43 Joules
Puissance totale consommée : 1059.89 W
```

FIGURE 1 – Console lors de l'exécution du programme

Le graphique ci-dessous illustre **l'évolution de la puissance consommée** (en Watt) par un ordinateur au repos, sur une durée d'environ 60 secondes. On observe une variabilité importante dans la consommation énergétique au cours du temps, avec des fluctuations oscillant entre environ 20.8W et 24.03W.

Néanmoins, on constate que la courbe tend à se stabiliser autour de 21.61W a 23.34W.

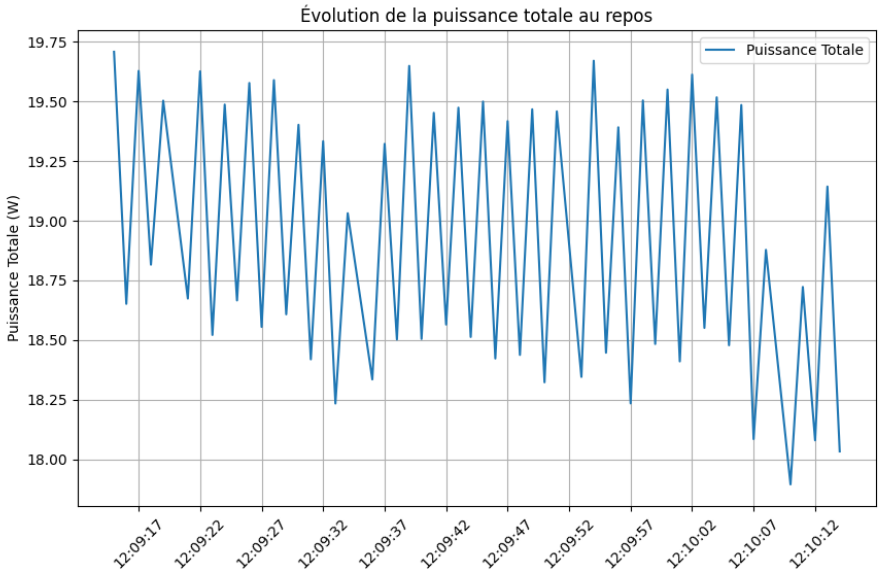

FIGURE 2 – Mesure de la puissance au repos

4 Mesure en activité

L'un des objectifs de ce TP N°2 est de **mesurer la consommation de l'ordinateur lorsqu'il est en activité**.

L'activité dans le TP sera représenté par une même vidéo (`https://cdn.kde.org/promo/Announcements/Plasma/5.20/Video.webm`) qui sera lancé sur deux navigateurs :

— Google Chrome
— Mozilla Firefox

Nous mesurons simultanément la consommation de puissance de l'ordinateur à l'aide de PowerJoular[4] selon trois approches distinctes :

— **Méthode 1 : Mesure général** : consiste à mesurer la puissance global de l'ordinateur

— **Méthode 2 : Mesure à l'aide du PID** : à l'inverse de la 1ère partie, on mesure la puissance consommé uniquement par le processus de l'application que l'on souhaite (ex : 56097)
— **Méthode 3 : Mesure à l'aide du nom de l'application** : similaire à la 2ème partie, mais au lieu d'utiliser l'identifiant d'un processus on utilise le nom de l'application (ex : Firefox)

Pour chaque méthode, nous avons réalisé trois benchmarks afin d'augmenter notre nombre de donnée et d'avoir une moyenne de la consommation plus réaliste.

Toutes les données sont accessible ici : `https://github.com/Guiguizou/numerique_responsable/tree/main/tp2/part1/browser`

4.1 Méthode 1 : Mesure général

Dans cette première méthode, nous mesurons la puissance totale consommée par l'ordinateur, et non celle du navigateur uniquement. Pour ce faire, nous avons redémarré l'ordinateur et fermer toutes les applications afin de ne laisser que Google Chrome ou Mozilla Firefox d'ouvert et de ne pas avoir d'application en arrière plan qui consomme de la puissance.

La commande PowerJoular que nous avons utilisé ici est **"powerjoular -f file.csv"**. Nous avons executé l'outil PowerJoular trois fois pour chaque navigateur, en enregistrant les données comme suit :

— "firefox-1.csv", "firefox-2.csv" et "firefox-3.csv"
— "chrome-1.csv", "chrome-2.csv" et "chrome-3.csv"

4.1.1 Analyse des données

A l'aide d'un script python et de la bibliothèque matplotlib[2], nous avons pu analyser les fichiers CSV afin d'obtenir les graphiques suivants :

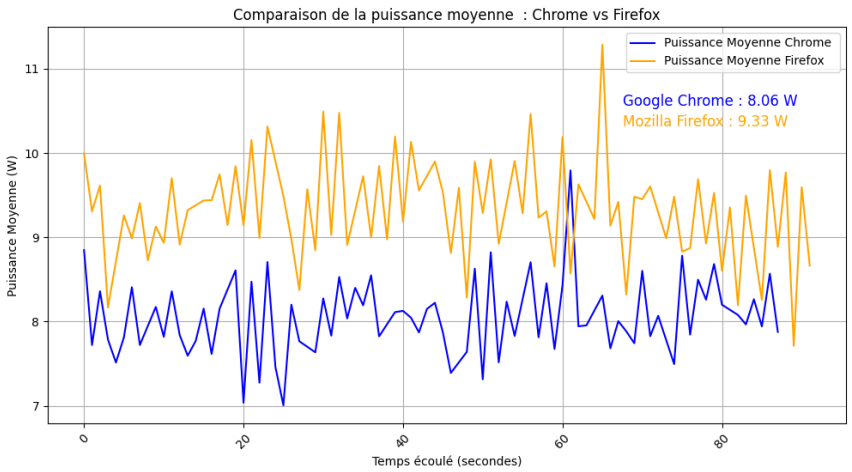

FIGURE 3 – Graphique de comparaison des puissances consommées par Firefox et Chrome

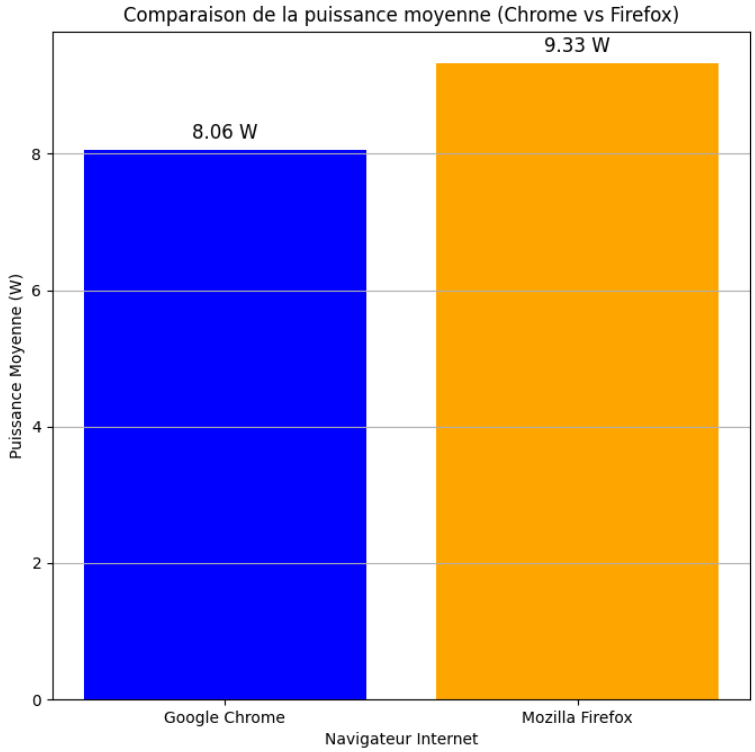

FIGURE 4 – Comparaison de la puissance moyenne de Firefox et Chrome

Les graphiques précédents illustrent la différence de consommation entre Google Chrome (en bleu) et Mozilla Firefox (en jaune). On constate que Google Chrome consomme en moyenne 8.06W tandis que Mozilla Firefox consomme en moyenne 9.33W.

Enfin, pour résumer nos graphiques, nous avons affiché toutes les consommations (puissance et énergie) sur le terminal (capture d'écran ci-dessous).

On peut donc conclure que Mozilla Firefox est plus énergivore que Google Chrome, même si la différence n'est pas forcément grande. On arrive à la même conclusion que lors du TP N°1.

```
PS C:\Users\guigu\Desktop\tp1\numerique_responsable\tp2\part1> python .\script_browser.py

Puissance moyenne globale de Idle : 18.93 W
Énergie totale consommée par Idle : 1113.43 Joules


Puissance moyenne globale de Chrome : 8.06 W
Puissance moyenne globale de Firefox : 9.33 W
```

FIGURE 5 – Terminal résumant l'analyse des données de la méthode 1

4.2 Méthode 2 : Mesure à l'aide du PID

L'avantage de powerjoular est qu'on peut également mesurer la puissance et l'energie consommé à l'aide de l'identifiant d'un processus spécifique (PID). Cela permet de se concentrer uniquement sur une application spécifique, au lieu de mesurer la consommation de tout l'ordinateur.

Cela est réalisable grâce à la commande 'powerjoular -p pid -f file.csv'.

Nous avons réalisé le même nombre de mesures que dans la partie précédente 4.1Méthode 1 : Mesure général. Sauf que nous avons utilisé la commande en spécifiant le PID de Google Chrome et de Mozilla Firefox afin de se focalisé que sur ces 2 processus.

Nous avons sauvegardé les données au format CSV comme suit :

— "firefox-1-pid.csv", "firefox-2-pid.csv" et "firefox-3-pid.csv"
— "chrome-1-pid.csv", "chrome-2-pid.csv" et "chrome-3-pid.csv"

4.2.1 Analyse des données

A l'aide de ces graphiques, on peut faire la même analyse que pour la figure "3.Graphique de comparaison des puissances consommées par Firefox et Chrome".

Mozilla Firefox possède une consommation de puissance plus importante que Google Chrome.

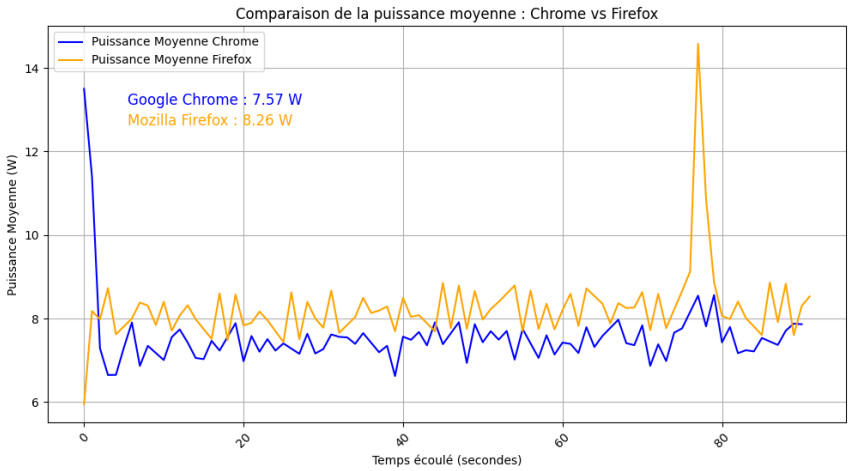

FIGURE 6 – Graphique de comparaison des puissances consommées par Firefox et Chrome avec la méthode PID

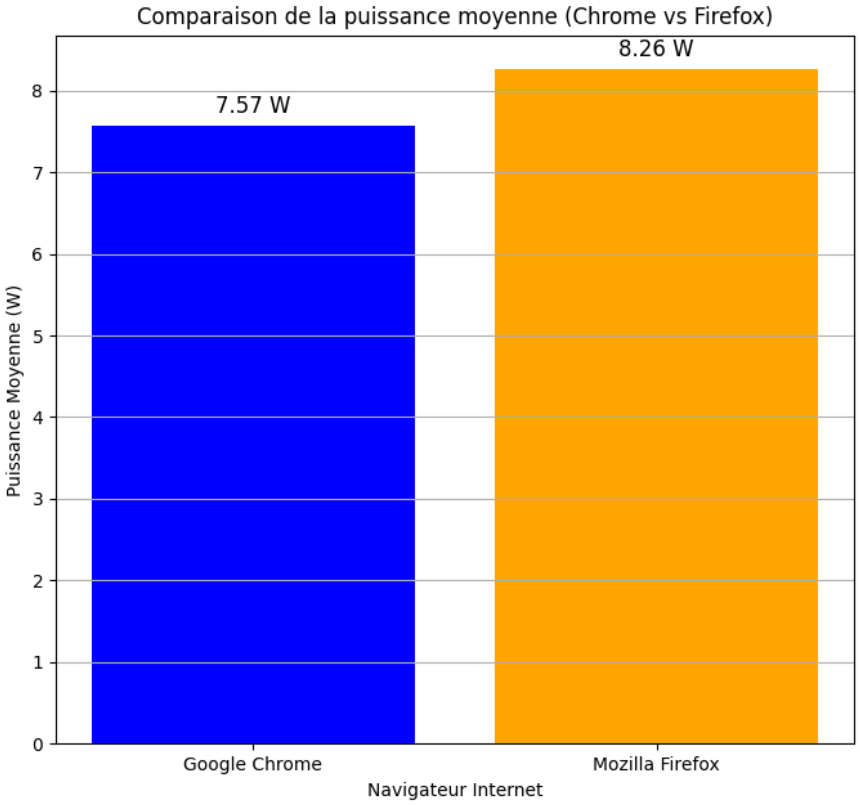

FIGURE 7 – Comparaison de la puissance moyenne de Firefox et Chrome avec la méthode du PID

Pour conclure sur la puissance consommé par les navigateurs en utilisant l'outil powerjoular associé au PID de l'application, nous arrivons à la même conclusion. En effet, Mozilla Firefox semble être de nouveau plus énergivore que Google Chrome.

Nous pouvons également aperçevoir une très légère baisse de l'energie totale consommé par les 2 navigateurs entre la Méthode 1 : Mesure général et la Méthode 2 : Mesure à l'aide du PID

```
PS C:\Users\guigu\Desktop\tp1\numerique_responsable\tp2\part1> python .\script_browser_pid.py

Puissance moyenne globale de Idle : 18.93 W
Énergie totale consommée par Idle : 1113.43 Joules

Puissance moyenne globale de Chrome : 7.57 W
Énergie totale consommée par Chrome : 675.15 Joules

Puissance moyenne globale de Firefox : 8.26 W
Énergie totale consommée par Firefox : 751.84 Joules
PS C:\Users\guigu\Desktop\tp1\numerique_responsable\tp2\part1>
```

FIGURE 8 – Terminal résumant l'analyse des données avec la méthode PID

4.3 Méthode 3 : Mesure à l'aide du nom de l'application

La 3ème méthode consiste à mesurer la consommation d'une application spécifique en utilisant son nom.

Par exemple, pour Firefox, nous utiliserons la commande : 'powerjoular -a firefox -f firefox-1-appname.csv'.

Nous avons donc réitérer toutes nos mesures en utilisant désormais cette commande. Et les données ont été enregistré comme suit :

- "firefox-1-appname-firefox.csv", "firefox-2-appname-firefox.csv" et "firefox-3-appname-firefox.csv"
- "chrome-1-appname-chrome.csv", "chrome-2-appname-chrome.csv" et "chrome-3-appname-chrome.csv"

4.3.1 Analyse des données

Pour changer et tester une nouvelle manière d'analyser les données, nous nous sommes désormais basé sur l'utilisation du CPU et non sur la consommation totale. De plus, nous avons aussi fait la comparaison de l'énergie consommée (en Joules) et non de la puissance.

En procédent de cette manière, on parvient à la même conclusion que les méthodes précédentes. En effet, Google Chrome possède une consommation nettement inférieur à celle de Mozilla Firefox pour la même utilisation comme le démontre les graphiques suivants :

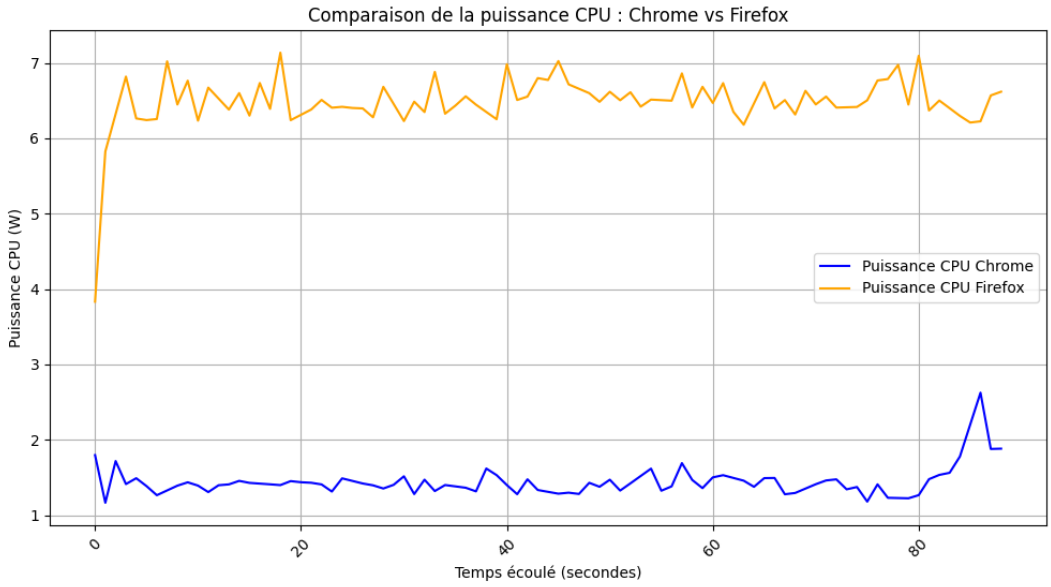

FIGURE 9 – Graphique de comparaison des puissances consommées par Firefox et Chrome avec la méthode AppName

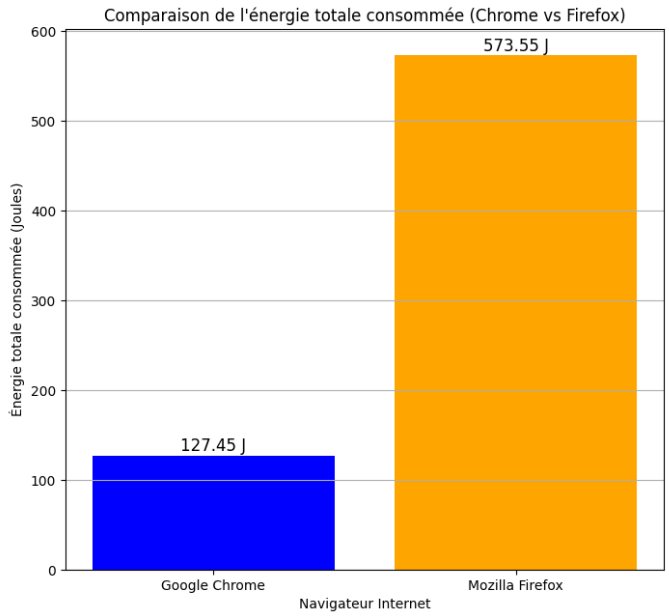

FIGURE 10 – Comparaison de la puissance moyenne de Firefox et Chrome en fonction de la puissance au repos avec la méthode AppName

```
PS C:\Users\guigu\Desktop\tp1\numerique_responsable\tp2\part1> python .\script_browser_appname.py

=== Résultats de l'analyse ===
Google Chrome :
 - Puissance moyenne : 1.44 W
 - Énergie totale consommée : 127.45 Joules

Mozilla Firefox :
 - Puissance moyenne : 6.49 W
 - Énergie totale consommée : 573.55 Joules

PS C:\Users\guigu\Desktop\tp1\numerique_responsable\tp2\part1>
```

FIGURE 11 – Terminal résumant l'analyse des données avec la méthode AppName

4.3.2 Conclusion sur les méthodes de mesure de l'activité

Pour mesurer la consommation de l'ordinateur en activité, nous avons utilisé trois méthodes différentes :

- **Méthode 1 : Mesure général de la consommation globale de l'ordinateur** : "powerjoular -f file.csv"
- **Méthode 2 : Mesure à l'aide du PID** : "powerjoular -p pid -f file.csv"
- **Méthode 3 : Mesure à l'aide du nom de l'application** : "powerjoular -a appname -f file.csv"

Pour chacune des méthode on arrive à la même conclusion, Google Chrome est moins enérgivore que Mozilla Firefox. Suivant la méthode utilisée, on peut constater une différence entre les deux navigateurs internets plus ou moins grandes.

5 Comparaison des langages de programmations

Dans cette partie, nous utiliserons toujours l'outil PowerJoular[4] mais également l'outil JularJX[3], ainsi que l'algorithme de Ray Casting disponible ici : `https://rosettacode.org/wiki/Ray-casting_algorithm`.

Toutes les données de cette partie sont accessible ici `https://github.com/Guiguizou/numerique_responsable/tree/main/tp2/part2`

Le but ici est de comparer la consommation de la puissance lorsque l'ordinateur exécute le même algorithme mais dans différents langages de programmations.

Pour ce faire, nous avons modifié les programmes (Java, C, python...) afin d'avoir les mêmes valeurs en entrée. Nous avons également dupliquer la version Java pour obtenir dans l'une, une sortie similaire à la version C (1,0), et dans l'autre une sortie avec du texte (true, false) afin de voir l'impact qu'à également l'affichagage sur la consommation d'un programme

5.0.1 Benchmark

Pour faire nos mesures, nous avons implémenté un programme python qui permet d'exécuter chaque programme pendant 30 secondes. Nous avons fait une pause de 10 secondes entre les différents programmes afin d'isoler la consommation de la puissance.

Voici les graphique d'analyses :

Ce graphique représente la puissance moyenne, en Watt, des quatre programmes réalisés avec les trois benchmarks.

On constate que le programme écrit en Langage C est celui qui consomme le plus de puissance.

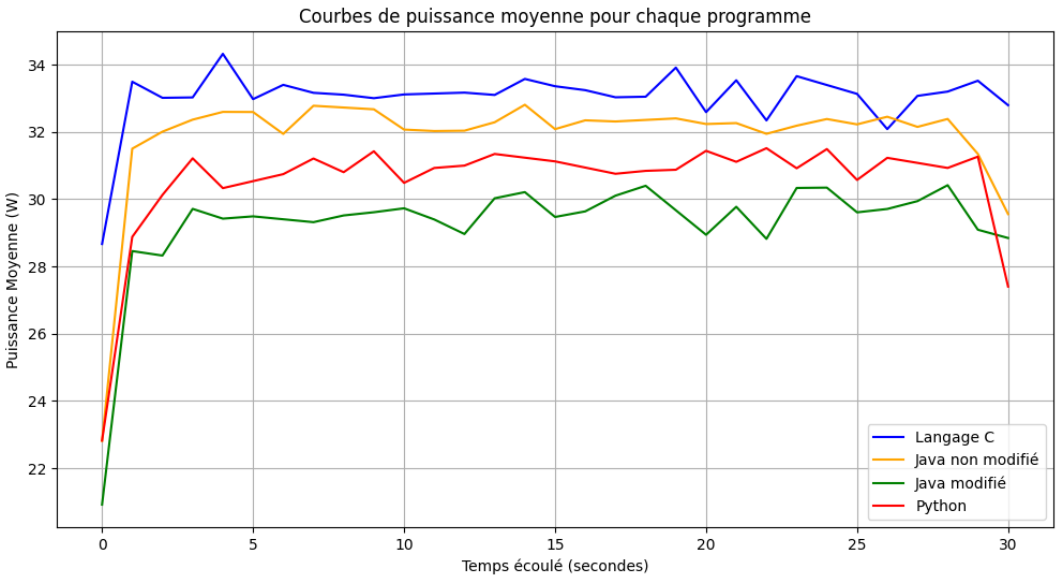

FIGURE 12 – Graphique de comparaison des puissances consommées par les différents langages de programmations

Le bar chart ci-dessous met en avant la différence des puissances moyennes et de l'énergie totale consommée par chaque programme.

Comme vu dans le graphique précédent, le programme écrit en C est celui qui consomme le plus à la fois au niveau de la puissance en Watt (couleur vert) mais également au niveau de l'energie en Joules (couleur bleu)

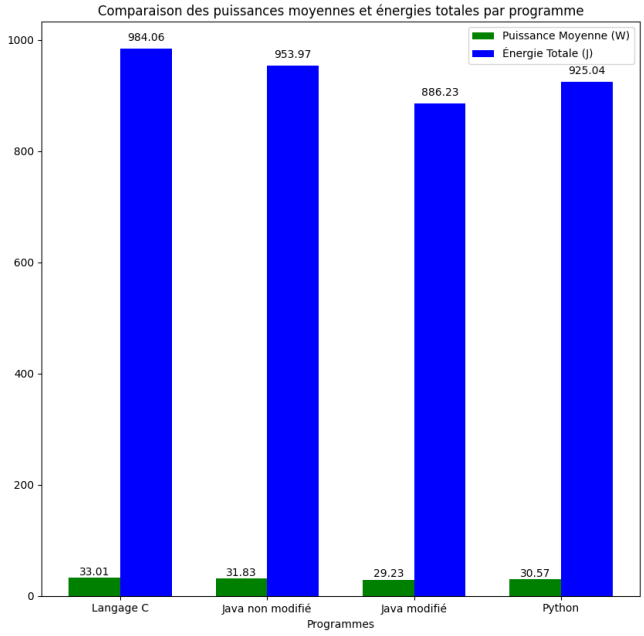

FIGURE 13 – Comparaison de la puissance moyenne et de l'énergie totale entre les langages de programmations

```
PS C:\Users\guigu\Desktop\tp1\numerique_responsable\tp2\part2> python .\scripts_raycasting_analyse.py

=== Résultats finaux ===
Langage C :
 - Puissance Moyenne (Total) = 33.01 W
 - Énergie Totale = 984.06 J
 - Puissance Moyenne (CPU) = 18.79 W
 - Énergie Totale CPU = 561.14 J

Java non modifié :
 - Puissance Moyenne (Total) = 31.83 W
 - Énergie Totale = 953.97 J
 - Puissance Moyenne (CPU) = 17.15 W
 - Énergie Totale CPU = 516.52 J

Java modifié :
 - Puissance Moyenne (Total) = 29.23 W
 - Énergie Totale = 886.23 J
 - Puissance Moyenne (CPU) = 15.18 W
 - Énergie Totale CPU = 462.84 J

Python :
 - Puissance Moyenne (Total) = 30.57 W
 - Énergie Totale = 925.04 J
 - Puissance Moyenne (CPU) = 16.03 W
 - Énergie Totale CPU = 486.98 J
```

FIGURE 14 – Terminal résumant l'analyse des données de comparaison entre les langages de programmations

5.0.2 "Source Code Power Consumption"

Dans cette partie, nous utiliserons JoularJX[3] afin de comparer la consommation des deux programmes Java de RayCasting. La première version correspond au programme de base c'est-à-dire avec un affichage de true ou false. Tandis que la seconde version affiche en sortie des 1 ou 0.

Voici les graphiques que nous avons pu générer à l'aide des données relevées :

On constate que le programme 2 (modifié) dont l'affichage a été allégé pour n'afficher que des 1 ou 0 au lieu de true ou false est moins énergivore que le programme 1.

En effet, comme on peut le visualiser sur les graphiques et le termianl, la puissance moyenne du programme est d'environ 29.31W contre 27.58W. Et concernant l'énergie totale consommée nous avons 862.76 Joules pour le premier programme contre 798.41 Joules pour le seconds.

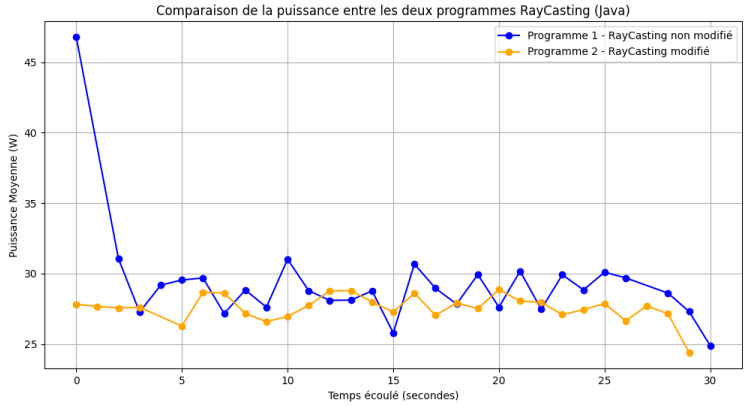

FIGURE 15 – Graphique des puissances consommées par RayCasting avec JoularJX

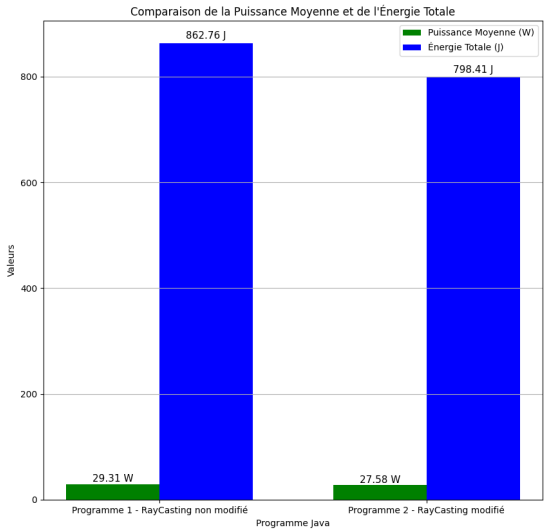

FIGURE 16 – Comparaison de la puissance moyenne et de l'énergie totale de RayCasting avec JoularJX

FIGURE 17 – Terminal résumant l'analyse des données de RayCasting avec JoularJX

6 Comparaison des collections

La dernière partie de ce TP consiste à comparer la consommation d'energie de plusieurs collections Java ("lists and maps").

Toutes les données de cette partie sont accessible ici `https://github.com/Guiguizou/numerique_responsable/tree/main/tp2/part2/javacollection`

Nous avons compiler et exécuter les différentes collections et mesurer à nouveau la consommation d'énergie à l'aide de JoularJX[3]. Chaque exécution permet de faire une insertion d'elément à la collection, d'accéder aux élements, et de supprimer les éléments.

Nous avons fait 2 séries de mesures distinctes.

- La 1ère série concerne les collections avec une seule et unique méthode pour faire les trois opérations
- La 2ème série de mesure s'est effectuée avec les méthodes séparée

Etant donnée que le nombre de donnée est assez important, pour ne pas surcharger le compte rendu nous avons fait la comparaison avec uniquement deux programmes.

6.1 ArrayList2.java

Lors de l'exécution du programme dans sa version originale et modifiée, nous avons inséré 1 000 000 éléments, et supprimer 500 000 éléments.

Voici le résultat obtenu à l'aide de JoularJX :

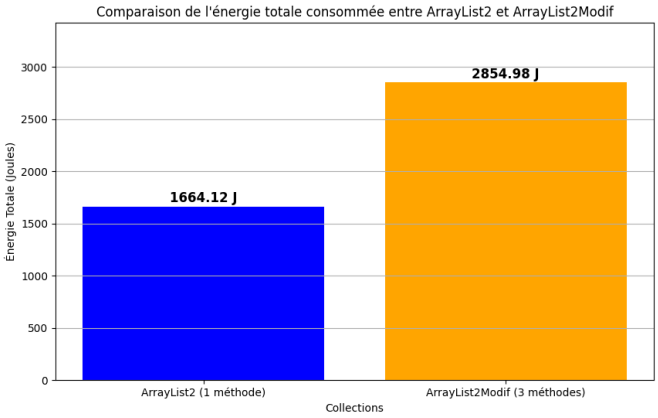

FIGURE 18 – Comparaison de l'énergie consommée entre ArrayList2 (1 méthode) et ArrayList2Modif (3 méthodes)

On constate que la version original qui regroupe les 3 opérations consomme moins que la version modifié qui sépare les 3 opérations.

6.2 LinkedHashMap2

On a répété la même opération pour le programme LinkedHashMap2 sauf que le nombre d'élément à été modifié. Nous avons inséré 10 000 000 éléments et supprimer les 5 000 000 premiers éléments.

On constate ici que pour un nombre d'éléments plus important que le ArrayList2, la consommation d'énergie est nettement inférieur.

De plus, On s'aperçoit que même si la différence est minime, la version qui sépare les 3 opérations consomme légèrement moins que la version qui réunit les 3 opérations, contrairement à ArrayList2.java. Pour gérer des listes et maps il sera donc mieux de prioriser le choix du LinkedHashMap2 que le ArrayList2

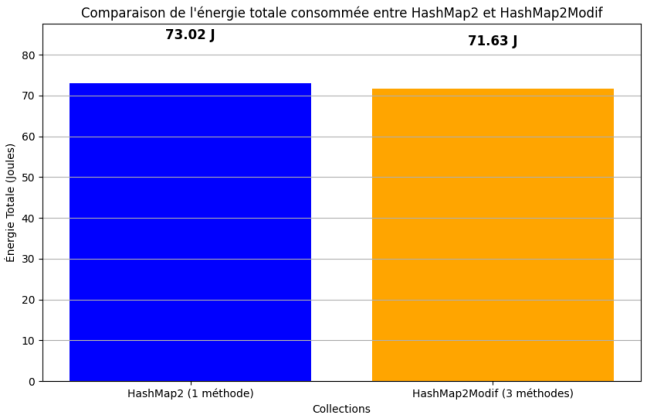

FIGURE 19 – Comparaison de l'énergie consommée entre LinkedHashMap2 (1 méthode) et LinkedHashMap2Modif (3 méthodes)

6.3 Conclusion

Pour le compte rendu, nous nous sommes arrêté à la comparaison de deux collections afin de ne pas le surcharger.

Nous avons pu observer une différence de consomme d'énergie importante entre ArrayList2.java et LinkedHashMap2.

De plus, on a également vu une différence avec les versions qui sépare les opérations. En effet, pour le ArrayList2.java la version qui réunit les 3 opérations consomme largement moins que la version qui les sépare. Tandis que pour LinkedHashMap2, c'est l'inverse, en séparant les opérations en 3 méthodes, nous avons pu réduire la consomme énergétique.

On peut donc en déduire avec nos résultats et quelques recherches, que la consommation d'énergie varie selon la structure et la gestion interne des collections. ArrayList2.java consomme moins d'énergie quand les méthodes sont combinées car cela réduit les surcharges liées aux appels de méthode et optimise les accès en mémoire. En revanche LinkedHashMap2 est plus efficace avec des méthodes séparées en raison de sa gestion optimisée des opérations et de l'ordre des élements. Pour résumé, LinkedHashMap2 consomme moins d'énergie que ArrayList2.java car il évite les coûts de réorganisation des élements lors des insertions et suppressions.

C'est pour cette raison qu'il est important de faire attention à la manière dont on programme, car de petites optimisations peuvent réduire significativement la consommation d'énergie et améliorer l'efficacité globale de l'application.

6.4 Conclusion

La réalisation de ce TP N°2 nous a permis de renforcer et d'enrichir les connaissances acquises durant le TP N°1.

En effet, en plus de mesurer la consommation d'énergie d'applications comme des navigateurs web ou de matériels non informatiques tels qu'un moniteur d'écran à l'aide d'un wattmètre et de l'outil PowerJoular[**?**], nous avons découvert l'outil JoularJX[**?**], qui nous a permis d'analyser la consommation énergétique de programmes informatiques. Cela nous a permis de mieux comprendre l'importance du 'Green Coding'. Grâce à JoularJX, nous pouvons tenter de minimiser l'impact environnemental des programmes et logiciels en identifiant quelles approches consomment le moins d'énergie.

Ainsi, nous avons constaté que, pour un même programme, la consommation énergétique peut varier significativement selon la manière dont il est codé (fonctions, méthodes, affichages, etc.).

6.5 Bibliographie

[1] PowerSpyCli — Adel Noureddine [en ligne], `https://github.com/joular/powerspycli?tab=readme-ov-file`.

[2] Matplotlib — Visualization with Python Documentation [en ligne], https://matplotlib.org.

[3] JoularJX - Adel Noureddine [en ligne], https://www.noureddine.org/research/joular/joularjx

[4] PowerJoular - Adel Noureddine [en ligne], https://www.noureddine.org/research/joular/powerjoular