# Analyse des impacts énergétiques

Louis Jézéquel-Royer
Andriy Parkhomenko
Etienne Gacel
Bastien Fabre
Mohamed Ali Lagha

27 Janvier 2021

## Contents

# 1 Introduction

Ce document a pour but d'initier une réflexion vis-a-vis de l'impact énergétique de notre projet, de mettre en avant nos choix et de nous faire prendre conscience de l'impact que ces derniers peuvent avoir. Nous allons séparer ce document en trois parties principales : l'impact énergétique de la compilation, la complexité du programme assembleur généré, et enfin la discussion de notre processus de validation.

# 2 Compilation

## 2.1 Présentation de nos choix

Durant ce projet, notre but principal a été d'avoir un compilateur fonctionnel, l'impact énergétique comme l'optimisation passant au second plan. En effet, avec plus de temps nous aurions mis en place une solution, tel que réaliser une deuxième passe afin de retirer des instructions inutiles, ou encore implémenter FMA pour réduire le nombre d'instruction.
Pour nous réduire notre impact écologique signifiait également optimiser notre code, afin d'enlever le plus de calculs inutiles possibles et de réaliser certaines tâches en diminuant le nombre d'opération. Une problématique qui aurait pu se poser était la seconde passe sur le code assembleur, permettant de supprimer les opérations redondantes (attribution de registre alors que la valeur voulue est déjà stockée par exemple). Une seconde passe, bien que réduisant le nombre de calculs effectués par le code assembleur lorsque ce dernier est exécuté, aurait sûrement rajouter des instructions dans le compilateur. Il aurait alors été particulièrement intéressant d'étudier le coût d'une autre passe et le comparer au coût actuel, voir implémenter les deux et permettre à l'utilisateur de choisir lequel utiliser. Par un exemple, si il a besoin d'utiliser toujours le même fichier assembleur sans avoir à recompiler, il va être évident qu'il va être mieux d'avoir un code assembleur plus optimisé.

# 3 Exécution

## 3.1 Présentation du programme de référence

Afin de tester si le code assembleur obtenu a un fort impact énergétique, nous avons eu l'idée de comparer la complexité théorique et la complexité du code assembleur d'un programme. Pour cela nous allons utiliser deux programmes triviaux : le parcours d'une liste chaînée et le double parcours de celle-ci. En théorie nous devrions avoir une complexité linéaire pour le premier et quadratique pour le second.

## 3.2 Résultats obtenus

Pour réaliser les tests nous avons exécuté nos programmes sur une série de valeur allant de 100 en 100 jusqu'à 3000, grâce à un script shell et un programme python.

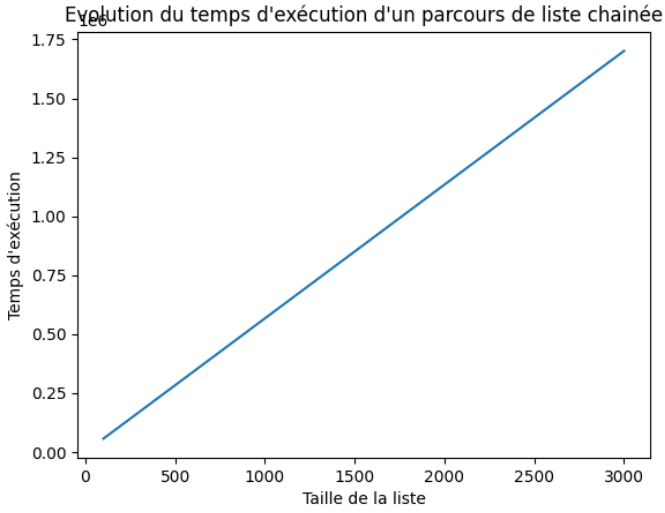


Figure 1: Parcours d'une liste chaînée

Nous observons que dans les deux cas nous obtenons une complexité proche de la théorie, en particulier pour le premier programme. Le second programme semble commencer avec une complexité quadratique puis tend bizarrement vers quelque chose de linéaire. Grâce à ces résultats nous pouvons affirmer que le code assembleur généré n'a pas un grand impact énergétique, c'est à l'utilisateur du compilateur de faire attention à ce que son programme ait un impact moindre.

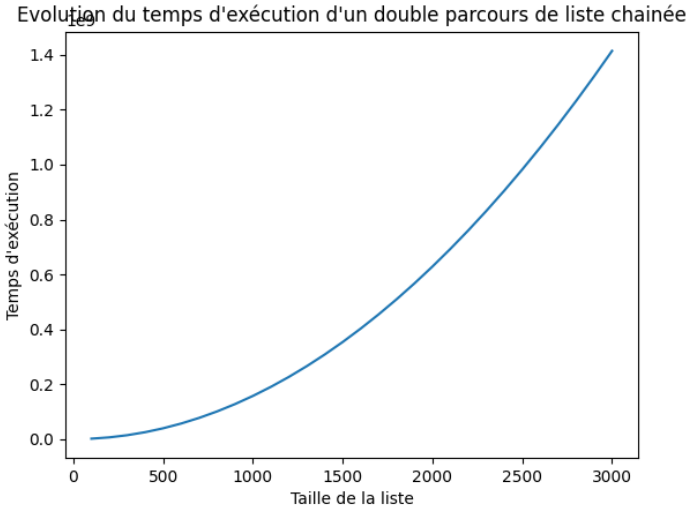


Figure 2: Double parcours d'une liste chaînée

# 4 Fonctionnement des test

Nous avons conçu un grand nombre de tests unitaires pour essayer de couvrir le plus de possibilités possibles. Lors d'un changement, il convenait souvent de lancer plusieurs tests afin de vérifier que les changements n'aient pas affectés les parties précédentes. Nous ne pouvions alors pas nous permettre dans lancer tous les tests que nous possédions, et nous avons alors développer des scripts permettant de faire tourner les tests propres à chaque partie de manière automatique. Ces scripts, pour les parties A et B, permettaient de choisir entre la partie sans objet, la partie avec objet, ou les deux. Pour la partie C, afin d'être sûr que notre travail n'avait pas impacté ce qui avait été fait précédemment, nous lancions tous les tests que nous avions fait.
En général, nous ne choisissions pas cet objet, préférant les tests unitaires. De plus, pour cette partie, il était nécessaire d'avoir des fichiers "réponses", dans lesquels nous mettions la réponse qui devait sortir. Il aurait été plus intelligent de mieux séparer les tests, notamment pour la partie génération de code, afin d'en effectuer seulement une partie. De cette manière, nous aurions pu réduire notre impact écologique et optimiser notre manière de tester.

# 5 Conclusion

Pour conclure, le développement durable, au même titre que l'optimisation (qui à notre avis vont de paire) est passé au seconde plan dans notre projet. Nos avons essayé de ne pas faire de tests inutiles et dès le début d'écrire un code effi-

cace ayant le moins d'impact énergétique possible, et malgré quelques réflexions, nous n'avons pas eu le temps de mettre en œuvre des solutions aux impact énergétiques moins conséquents.