

ENSIMAG
PROJET GL
GL 59

Analyse des impacts énergétiques



WALID BAKIR

WALID ABIDI

ABDELOUAHAB MESTASSI

EL MEHDI SALAH-EDDINE

YASSIN EL KHALDI AHANNACH

1 Introduction

Pendant notre avancement en projet nous avons voulu toujours faire un produit fonctionnel qui utilise le minimum d'énergie possible c'est pour cela que nos programmes écrits étaient optimisés. Nous avons aussi réduit l'utilisation des grandes structures de données comme des Maps qui peuvent faciliter nos tâches car ils peuvent lier plusieurs informations entre elles et rendre l'accès à ces dernières plus simple à faire. Aussi nous avons privilégié d'ajouter le minimum de structure de donnée possible chose qui a rendu nos programmes difficiles à écrire mais moins consommateurs d'énergie.

2 Discussion de l'impact énergétique dans nos choix de conception

La conception du compilateur a été beaucoup influencée par l'impact énergétique. Du coup l'équipe était contrainte de minimiser la consommation énergétique du numérique mais il fallait faire un compromis entre la minimisation et l'automatisation. Pour cela on a opté pour des structures de données qui limitent la consommation d'énergie du programme tel que l'utilisation des listes chaînées pour la gestion des registres libres et occupés. Aussi pour le choix des algorithmes a été fait de manière à optimiser l'utilisation des ressources. Néanmoins on a été toujours contraint de faire un compromis entre vitesse d'exécution et stockage dans la mémoire, par exemple pour le nombre des variables et le choix de structure de données pour le stocker.

Cependant, à cause des contraintes temporelles, l'équipe a mis parfois de côté quelques optimisations pour factoriser le code, car ça jouait sur le fonctionnement et on préférait avoir un code fonctionnel vers la fin même s'il n'est pas idéalement optimisé.

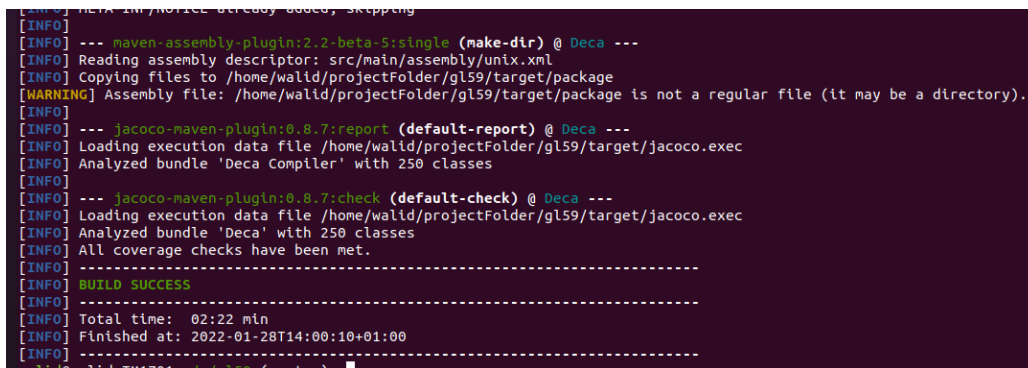
3 Évaluation de l'impact énergétique des programmes Ima

Comme il a été mentionné dans le poly, nous allons considérer que le nombre de cycle représente une estimation de la consommation de notre compilateur c'est pour cela qu'on a essayé d'utiliser le minimum de registre possible et d'éviter de faire des LOAD inutile dans notre programme et ceci afin d'avoir un nombre minimale de cycle, et le résultat que nous avons reçu était plus ou moins satisfaisant comparé

au compilateur des années précédentes par exemple le test `ln2.deca` qui est fourni demande 12000 cycles internes qui est satisfaisant, mais en ce qui concerne la partie objet ,et vu le temps qui nous restait nous n'avons pas bien gérer la génération du code et on a pas pris en considération de nombre de cycle interne chose qui a donné des résultats moins satisfaisant dans les test énergétique avec une consommation supérieur aux années précédentes notamment sur le test `ln2ft.deca`.

4 Évaluation de l'impact énergétique de la validation et tests

L'impact énergétique de notre compilateur se retrouve également dans l'exécution des tests et la validation. Lorsque nous tapons la commande `mvn -Djacoco.skip=false verify`, cela parcourt tous nos tests un par un. Le temps approximatif d'exécution de tous les tests est d'environ deux minutes.



```
[INFO] META-INF/NOTICE already added, skipping
[INFO] --- maven-assembly-plugin:2.2-beta-5:single (make-dir) @ Deca ---
[INFO] Reading assembly descriptor: src/main/assembly/unix.xml
[INFO] Copying files to /home/walid/projectFolder/gl59/target/package
[WARNING] Assembly file: /home/walid/projectFolder/gl59/target/package is not a regular file (it may be a directory).
[INFO] --- jacoco-maven-plugin:0.8.7:report (default-report) @ Deca ---
[INFO] Loading execution data file /home/walid/projectFolder/gl59/target/jacoco.exec
[INFO] Analyzed bundle 'Deca Compiler' with 250 classes
[INFO] --- jacoco-maven-plugin:0.8.7:check (default-check) @ Deca ---
[INFO] Loading execution data file /home/walid/projectFolder/gl59/target/jacoco.exec
[INFO] Analyzed bundle 'Deca' with 250 classes
[INFO] All coverage checks have been met.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:22 min
[INFO] Finished at: 2022-01-28T14:00:10+01:00
[INFO] -----
[INFO] Walid@walid-TM1761: ~/gl59 (master) %
```

FIGURE 1 –

Nous nous sommes assurés d'utiliser des tests ciblés sur chaque fonctionnalité de manière à ne pas répéter un test et si plusieurs opérateurs ou fonctions peuvent être testés dans un seul fichier, nous les avons combinés.

5 Évaluation de l'impact énergétique des autres outils et du développement

Sur notre dépôt git nous avons essayé d'utiliser le minimum de push possible dans le but de réduire la consommation énergétique de notre travail sur le projet, chacun de nous travailler sur sa machine et à chaque grand avancement on dépose notre travail et nous évitons le maximum de faire des push tous les 5-10 minutes car ceci consomme une partie de l'énergie non négligeable.

6 Impact énergétique de l'extension

La prise en considération de l'impact énergétique lors de l'implémentation de notre extension était un élément indispensable , notamment à cause de son importance

pour la planète . Un bon ingénieur se doit de respecter les contraintes écologiques et énergétiques lors de la réalisation d'un travail.

Pour implémenter nos fonctions , nous avons le choix entre plusieurs algorithmes : Les approximations polynômiales , CORDIC etc ... Nous avons finalement opté pour des approximations polynômiales utilisant des séries de Fourier - Chebyshev ou des séries de Taylor car elles sont moins coûteuses et gardent tout de même une bonne précision.

Les coefficients des polynômes utilisés ont été calculés au préalable , notamment pour le coût que leurs calculs pourraient induire , notamment avec toutes les opérations qu'elles comportent , des divisions , des calculs d'intégrales ...

Nous vérifions également le temps d'exécution de nos méthodes , en les réécrivant en Java en respectant la syntaxe Deca , et on compare ensuite les résultats obtenus avec ceux des méthodes de la Bibliothèque java.lang.Math , on utilise la méthode suivante :

```
public static void main(String[] args) {  
    MathTrigo m = new MathTrigo();  
  
    long startTime = System.nanoTime();  
    m.my_cos(0);  
    long endTime = System.nanoTime();  
    long duration = (endTime - startTime);  
    System.out.println(duration);  
}
```

On remarque qu'après une série de calculs , les méthodes java sont plus rapides , cela est dû au fait qu'elles soient beaucoup plus optimisées .