Analyse des impacts énergétiques

Version 1.0 (release)

Sommaire

Introduction

En tant que futur ingénieur, nous nous devons d’avoir une certaine réflexion sur l'efficacité énergétique de notre méthode de développement ainsi que celle de notre code, ou du code généré par notre compilateur.

Le numérique est responsable d’environ 2% des émissions de gaz à effet de serre en France. Ce chiffre pourrait s'accroître dans les années qui viennent pour atteindre 7% des émissions si rien n'est fait.

L’étude de l’impact énergétique de notre code semble alors pertinente. Elle nous pousse à le réduire et cela ne peut être que bénéfique pour notre futur.

Causes sur l’efficacité énergétique

Il existe plusieurs façons de mesurer l'efficacité énergétique d'un programme. Tout d’abord, le nombre de cycles de processeur nécessaires pour exécuter le programme est à prendre en considération. Plus le programme nécessite de cycles pour être exécuté, plus il consommera d'énergie.

De plus, il est utile d’étudier la quantité de mémoire vive utilisée par le programme, car plus elle est grande, plus le programme consomme.

Enfin, la quantité de données qui doivent être lues ou écrites sur le disque dur pendant l'exécution du programme compte également pour beaucoup. Plus le programme doit accéder au disque dur, plus il consommera.

Ainsi, pour réduire sa consommation, il faut privilégier des assemblages d'instructions moins gourmands en énergie, minimiser le nombre de cycles de processeur nécessaires pour exécuter le programme, mais aussi minimiser la quantité de mémoire et d'accès au disque dur utilisé.

Réduire sa consommation

Il est possible d'optimiser le code source du programme en utilisant des algorithmes plus efficaces et en utilisant des techniques de compilation avancées.

Il existe plusieurs outils qui permettent de mesurer la consommation énergétique d'un programme informatique. Le profilage de performance permet de mesurer le temps d'exécution et le nombre de cycles de processeur utilisés par chaque partie du programme. Les moniteurs de consommation énergétique permettent de mesurer la consommation énergétique du système informatique pendant l'exécution du programme. Les analyseurs de code permettent de mesurer l'efficacité énergétique du code en fonction de son organisation et de son utilisation des ressources du système.

Efficience du code produit

Concernant le code produit, nous avons fait le choix d'une traduction systématique du code deca vers le langage machine associé à ima. Ainsi, le code répond tout d'abord à la correction et à la justesse de la traduction. Cependant, même si le code produit n'est pas optimisé à son maximum, les considérations du coût des opérations machine ont été prises en compte lors de la phase de développement, dans le but de minimiser les instructions superflues.

Nous pouvons par exemple regarder le nombre de cycles du test ln2.deca situé dans src/test/deca/codegen/perf, On obtient avec l'option -s de ima un nombre de cycle égale à 15 554.

```
6.93148e-01 = 0x1.62e448p-1
Nombre d'instructions :      109  Temps d'execution :  15554
```

Efficience du procédé de fabrication

Concernant le procédé de fabrication et la partie test, nous avons eu des idées assez tôt afin de limiter l'impact des tests qui représentent une partie importante du temps de processeur que l'on utilise.

Premièrement nous avons décidé de ne pas utiliser une pipeline gitlab qui permet d'exécuter des tests automatiquement sur un serveur dédié. Cependant, l'un des principaux désavantages est qu'à chaque commit le test est lancé. Nous avons préféré faire des tests de non-régression chaque soir afin de vérifier que l'on avait pas endommagé des fonctionnalités pendant la journée, et si l'on souhaitait seulement lancer une partie des tests on utilisait un script python.

Nous avons également eu des démarches d'optimisation des tests en eux même, par exemple pour le lexer sur le test des caractères UTF8 ou nous avons voulu limiter l'accès mémoire en plaçant tous les caractères dans un même fichier. Nous avons aussi essayé d'être le plus méthodique afin de limiter les redondances dans les tests comme dans le test où nous avons parcouru la grammaire pour trouver les erreurs possibles puis où nous avons réalisé les tests liés. Pour l'extension nous avons décidé de seulement lancer les tests pour l'extension et ne pas doubler les tests sur des parties déjà tester.

À présent sur la partie analytique de la consommation du projet, nous avons pensé que prendre en compte la consommation générale des ordinateurs utilisés pourraient être utiles mais n'ayant pas d'ampèremètre sous la main nous n'avons pas pu réaliser cette démarche.

Cependant, nous avons pu évaluer la consommation du processus de test qui est de 1.17W pour le maven test grâce à PowerTop pour la consommation.

Nous avons utilisé le programme usr/bin/time pour le temps processus (704 s) et l'utilisation moyenne du processeur (173%). Cela fait une consommation de 823J soit environ 74s d'utilisation d'une ampoule de 11W, ou encore, cela correspond à moins qu'une recherche google (~1000 J), On constate que la consommation d'un test est assez faible.

```
704.50user 151.24system 8:13.60elapsed 173%CPU (0avgtext+0avgdata 353136maxresident)k
63816inputs+98472outputs (7049major+7851918minor)pagefaults 0swaps
```

Conclusion

Durant tout le projet, nous avons réfléchi à la dimension énergétique du développement, que cela soit dans le coût machine du code généré ou dans la consommation des tests. Malgré tout, nous aurions pu optimiser un peu plus notre code généré afin de réduire le nombre de cycles machines qui reste au dessus de celui du "Vieux Compilateur Prof". Pour cela nous pouvons par exemple essayer de transformer les multiplications par deux en shift à gauche (20 contre 2 cycles machine).