Analyse des impacts énergétiques du projet

Groupe gl27

1 Les moyens mis en œuvre

1.1 Pour le développement

Pour gérer la consommation énergétique du projet, on s'appuie principalement sur la commande `ima`, qui, grâce à l'option `-s`, permet d'afficher le nombre d'instructions qui se sont succédées avant la fin du programme, et le temps d'exécution de chaque programme de test.

Avec cela, on peut regarder si des programmes simples se terminent en un temps cohérent et si des programmes plus complexes finissent en un temps que l'on pourrait qualifier de raisonnable.

Aussi, on peut arriver à se demander si les tests à l'exécution ont un gros impact sur le temps de résolution des programmes.

Voici donc les axes de l'évaluation de la consommation, nous n'avons malheureusement pas eu le temps de travailler sur d'autres axes en profondeur, avec pour cause les petits imprévus rajoutés en cours de projet, qui ont grignoté petit à petit le temps alloué à ces ajouts.

Au niveau du stockage des programmes, nous avons utilisé le logiciel Git, ce qui permet de se retrouver avec une solution simple et qui marche en local, ce qui allège les transferts de fichiers.

1.2 Pour la communication/documentation

La communication au sein de l'équipe se séparait en 2 parties :

- Les messages directs pour se notifier des petites avancées sont stockés sur Messenger, ce n'est peut-être pas le plus optimal énergétiquement parlant mais ce moyen de communication étant très répandu dans l'école, on a directement pensé à celui-ci.
- Concernant nos réunions, celles que nous avons faites en distanciel se déroulaient sur Discord, dans un salon vocal, où on discutait des avancées et des éventuels problèmes que l'on peut rencontrer.

Par rapport à la gestion de notre documentation, puisqu'elle a été en partie écrite en Latex, on n'a qu'à enregistrer le fichier texte associé à chaque document pour les transférer, le reste de la documentation ayant été écrite sous la forme de Google Docs.

2 L'impact de nos choix

Étant une équipe qui a choisi le thème "Histoire", nous préférions faire les trois étapes pour ensuite se concentrer sur l'extension, sans vraiment chercher à optimiser le code que l'on avait écrit par la suite, puisqu'il nous paraissait satisfaisant au moment où il a été testé.

Tout d'abord, voici les performances du compilateur par rapport aux programmes apparaissant dans le palmarès des équipes, sur lequel nous sommes classés 16^e sur 37 :

Voici tout d'abord le programme appelé `syracuse42.deca` :

```
[cousinq@localhost Projet_GL]$ ima -s src/test/deca/codegen/perf/provided/syracuse42.ass
8
Nombre d'instructions :       49  Temps d'execution :  1306
```

Ensuite, voici le programme `ln2.deca` :

```
[cousinq@localhost Projet_GL]$ ima -s src/test/deca/codegen/perf/provided/ln2.ass
6.93148e-01 = 0x1.62e448p-1
Nombre d'instructions :       93  Temps d'execution :  14950
```

Enfin, voici le `ln2_fct.deca` :

```
[cousinq@localhost Projet_GL]$ ima -s src/test/deca/codegen/perf/provided/ln2_fct.ass
6.93148e-01 = 0x1.62e448p-1
Nombre d'instructions :      140  Temps d'execution :  18117
```

Suite à ces trois résultats, on peut en déduire que le nombre d'instructions n'a pas de véritable influence sur le temps d'exécution, tout du moins pour un petit nombre d'instructions (inférieur à 200), ce qui veut donc dire que les instructions n'ont pas toutes le même poids en temps d'exécution, et donc plus généralement ne sont pas égales en termes de consommation énergétique.

Nous avons donc essayé de rechercher des informations à ce sujet, mais malheureusement nous n'avons rien trouvé qui pourrait nous permettre de déterminer l'énergie nécessaire à la réalisation d'une opération élémentaire en langage assembleur, ornous n'avons rien trouvé de semblable, nous pouvons donc essayer de comprendre comment cela marche en faisant tourner des scripts de tests grâce à `ima -s`.

Aussi, on peut compiler les programmes `deca` directement avec `decac -n`, ce qui supprime les tests à l'exécution lors du lancement du fichier assembleur. Voici ce que les tests renvoient :

```
./src/test/deca/codegen/valid/test_method_call22.deca
Temps avec les tests à l'exécution
260
260
Nombre d'instructions :      718  Temps d'execution :  6458
Temps sans les tests à l'exécution
260
260
Nombre d'instructions :      582  Temps d'execution :  5672
./src/test/deca/codegen/valid/test_method_call23.deca
Temps avec les tests à l'exécution
ok
ok
Nombre d'instructions :      113  Temps d'execution :  561
Temps sans les tests à l'exécution
ok
ok
Nombre d'instructions :       80  Temps d'execution :  497
./src/test/deca/codegen/valid/test_method_call2.deca
Temps avec les tests à l'exécution
10
Nombre d'instructions :       84  Temps d'execution :  281
Temps sans les tests à l'exécution
10
Nombre d'instructions :       54  Temps d'execution :  243
./src/test/deca/codegen/valid/test_method_call3.deca
Temps avec les tests à l'exécution
1.00000e+01
Nombre d'instructions :       84  Temps d'execution :  283
Temps sans les tests à l'exécution
1.00000e+01
Nombre d'instructions :       54  Temps d'execution :  245
./src/test/deca/codegen/valid/test_method_call4.deca
Temps avec les tests à l'exécution
ok
Nombre d'instructions :       91  Temps d'execution :  309
Temps sans les tests à l'exécution
ok
Nombre d'instructions :       61  Temps d'execution :  271
./src/test/deca/codegen/valid/test_method_call5.deca
Temps avec les tests à l'exécution
10
Nombre d'instructions :       80  Temps d'execution :  263
Temps sans les tests à l'exécution
10
Nombre d'instructions :       52  Temps d'execution :  237
```

Suite à cette batterie de tests, qui s'est déroulée avec l'appel au script `codegen-1.sh`, on en déduit donc que les tests à l'exécution représentent une partie non négligeable du temps d'exécution, de l'ordre d'environ 10-15%. On peut donc comprendre qu'une partie du coût énergétique parte dans ces tests que l'on peut éviter, mais la majorité de ce coût vient bien des opérations. Enfin, grâce à ces tests, on pourrait déterminer quelle ligne de commande est la plus coûteuse parmi toutes, ce qui permettrait de prendre une décision de moins l'utiliser, par exemple.

La raison pour laquelle nous n'avons pas fait ceci a déjà été explicité, cela vient de notre envie de nous focaliser sur l'extension, quitte à ne plus améliorer nos temps, puisque nous les trouvions déjà suffisamment rapides.

3 Le processus de validation

Dans cette partie, nous allons essayer de voir si les choix que nous avons pris pour l'impact énergétique au niveau de notre processus de validation sont cohérents, c'est-à-dire si la qualité de notre compilateur n'est pas mise en cause, tout en essayant de garder intact l'effort de validation.

Étudions donc le prmeier point : La qualité du compilateur est représentée par sa rapidité et sa justesse. Or, notre compilateur marche correctement et est dans la moyenne des groupes de projet, nous pensons donc que sur ce point notre optimisation d'impact énergétique est correcte.

Pour le deuxime point, on doit étudier l'effort de validation, et la manière dont il a évolué avec cet effort.
Nous pensons que, de même que pour le point précédent, notre stratégie de réduction de l'impact énergétique n'a pas eu une si grande influence que ceci, on pourrait même dire que la partie de validation est un des outils pour jauger et évaluer la consommation,

Nous pourrions essayer d'aller chercher plus loin dans la réflexion, en effet nous pourrions essayer de comprendre en quoi les tests pourraient permettre d'atteindre un nouveau stade dans la recherche de l'optimisation. Je l'ai évoquée plus haut mais je n'y avais pas vraiment répondu, comment peut-on, à partir des lignes de commande les plus coûteuses, améliorer le compilateur afin de rendre l'impact énergétique plus faible ?

Pour répondre à ceci, il y a plusieurs possibilités :

- Soit on essaie d'optimiser cette ligne de code en particulier, c'est-à-dire que l'on essaie de réduire le nombre d'instructions associées à cette ligne, soit on réduit le nombre de calculs à faire, par les mathématiques ou autre manière.
- Si la première solution n'est pas possible, que cette ligne est donc la plus optimisée possible, mais qu'elle reste la plus coûteuse, on va donc chercher à remplacer cette ligne par des combinaisons d'autres lignes moins chères afin d'atteindre le même résultat, ou un résultat équivalent, pour un coût moins élevé qu'avant.

Ainsi, en appliquant une de ces méthodes, on pourrait, à partir des tests, trouver l'opération qui manque le plus d'optimisation et lui en apporter. Et cette méthode est applicable autant de fois que l'on veut, nous ne sommes limités qu'à ceci.

4 L'impact énergétique dans l'extension

Dans une extension de type "Histoire", la manière de programmer certaines choses change drastiquement, d'autant plus lorsque même le nombre de registres se met à varier.

Ainsi, l'impact énergétique se met à varier aussi, car le temps que l'on a accordé à cette partie n'est pas vraiment comparable à celui accordé aux parties précédentes. En effet, notre objectif principal par rapport à notre compilateur GBA était de faire fonctionner les aspects principaux du projet, donc pouvoir faire tourner des programmes compilé avec notre compilateur sur une console directement, puis implémenter les spécificités du projet petit à petit. Or, à cause du temps qui nous manquait, nous n'avons pas eu le temps d'implémenter la totalité des fonctionnalités, donc nous n'avons pas vraiment pu penser à l'impact énergétique, malheureusement.

Mais, étant donné que la compilation utilise beaucoup de choses qui sont déjà implémentées grâce aux étapes A, B et C du projet, la rapidité et la vision énergétique des fonctionnalités doit se ressembler, en effet il est plutôt difficile de pouvoir déterminer comment cela se passe dans la console, puisque nous n'avons pas de sorties équivalentes à celles du `ima -s` sur ce type d'appareil.