GRENOBLE INP Ensimag UGA

Documentation énergétique

*Projet Génie Logiciel - **Equipe GL 23***

Boukhriss Nada / Dakhil Yahya / Elhjouji Salah /Et-tarraf Zineb / Tamouh Alae

JANVIER 2022

Introduction :

La consommation d'énergie devient rapidement l'un des contraintes de conception les plus importantes lors de l'édition des logiciels .

En programmant un compilateur on est censé investir un temps important pour voir les retours de consommation d'énergie issues de l'exécution des programmes tests via notre compilateur . Vu que ce dernier sera utilisé pour produire des logiciels, qui consommeront de l'énergie lors de leur exécution. Il faut donc qu'on évalue le coût énergétique des assemblages d'instructions que réalise notre compilateur. Et choisir la conception de code qui permettra l'assemblage le plus optimal .

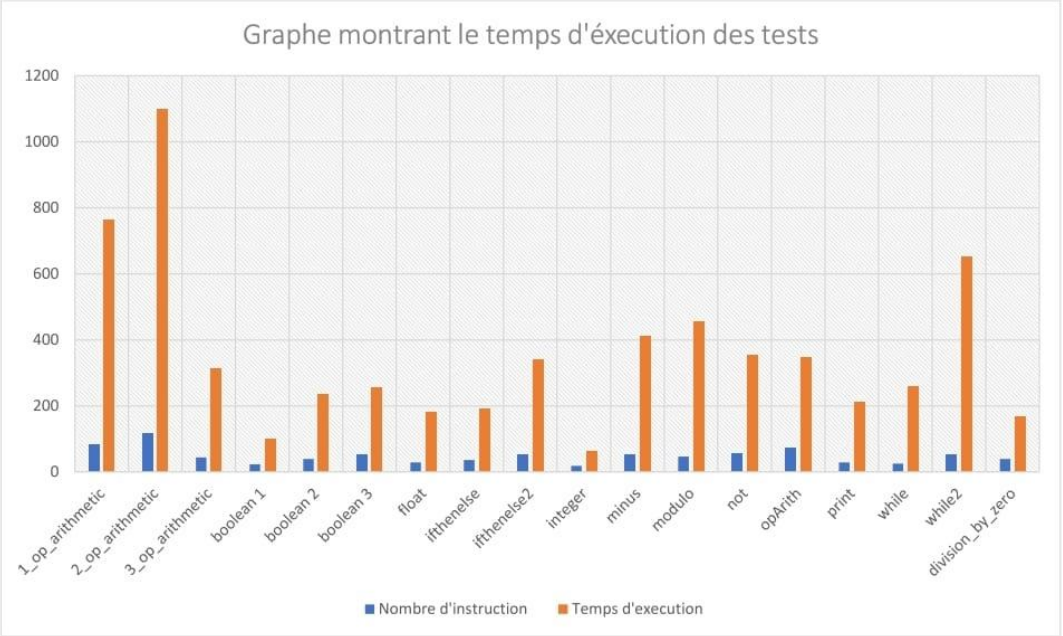
1) L'impact des choix de conception compilation

Les choix de conception sont l'élément principal qui affecte le code assembleur à la sortie de la compilation et donc affecte directement la consommation énergétique du programme ima, mais aussi il affecte directement la consommation énergétique durant le processus même de la compilation.

Un point principal qui fait la différence dans la consommation d'énergie et le choix des classes de collections inadaptées qui est susceptible d'augmenter la consommation énergétique de presque 38%. On a accordé beaucoup d'attention sur ce point en choisissant judicieusement les types de collection adaptés dans chacune des différentes situations qu'on a pu rencontrer durant l'implémentation de notre projet. Cependant, on s'est retrouvé parfois contraint d'utiliser des technique coûteuses en temps et en énergie tel que l'insertion d'un élément dans une liste chaînée. En effet, lors de la définition des méthodes on a été contraint d'insérer les instruction TSTO et PUSH en ajoutant une méthode d'insertion dans la liste chaînée des instructions car celle-ci demandait d'avoir le nombre exacte de registre utilisée qui ne pouvait se faire qu'après la détermination des instruction qui suivent. Mais notant que l'insertion en début de liste chainée et la moins coûteuse de toutes les collections et donc à été utilisé en priorité quand c'était pertinent.

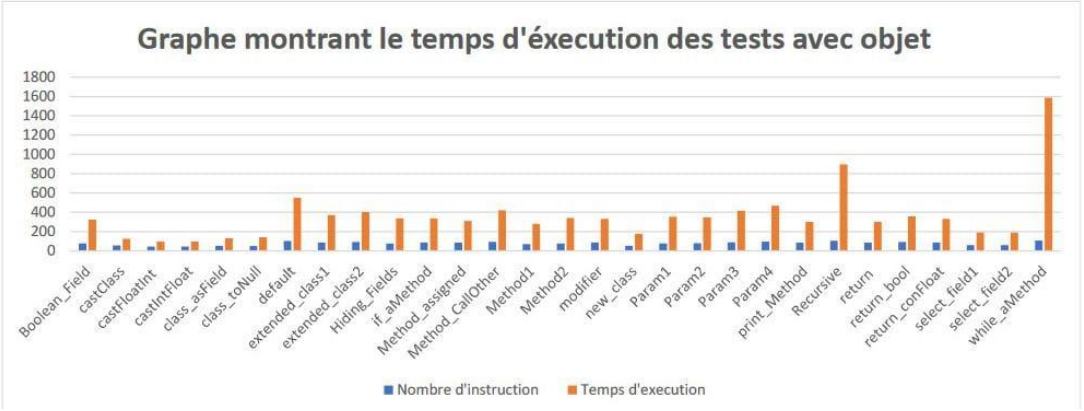
On a aussi pensé à limiter la consommation de l'énergie de l'exécution des programmes IMA en optimisant au maximum le code assembleur généré. On a donc essayé d'éliminer le maximum d'instructions inutiles dans le temps qui nous a été imparti. On pensait alors toujours à libérer les registres pour éviter une utilisation extensive de la pile. On peut citer à titre d'exemple les LOAD inutiles tel qu'un LOAD d'un registre r1 vers un registre r2 puisque on peut utiliser le registre r1 directement pour la suite.

2) La consommation d'énergie des tests valides avec objets



TOTAL : Nombre d'instructions : 871
Temps d'exécution : 6424

3) La consommation d'énergie des tests valides avec objets



TOTAL : Nombre d'instructions : 2261
Temps d'exécution : 10509

REMARQUE :

Dans ce graphe, montrant la consommation énergétique des tests avec objet, on voit que les tests de la récursivité et les boucles while consomment plus d'énergie par rapport aux autres tests.

On note que l'évaluation de l'aspect énergétique des tests valides de l'étape C de la génération du code prend un temps d'exécution (avec objet et sans objet combinés) égal à peu près au temps d'exécution que prend le test du ln2.deca.

Ainsi

Equipe	Extension	S=Syracuse42	L0=ln2	L1=ln2_fct	Score=L0+L1+10*S
VieuxCompDeProf	TRIGO	1340	15194	18102	46696
gl36	TRIGO	1528	19056	22588	56854
gl18	TAB	1708	16728	50000	83808
gl00		5000	50000	50000	150000

```
afaw@afaw-VirtualBox:~/Projet_GL$ ima -s /home/afaw/Projet_GL/src/test/deca/codegen/perf/provided/syracuse42.ass
8
Nombre d'instructions :       39  Temps d'execution :  1408
afaw@afaw-VirtualBox:~/Projet_GL$ ima -s /home/afaw/Projet_GL/src/test/deca/codegen/perf/provided/ln2.ass
6.93148e-01 = 0x1.62e448p-1
Nombre d'instructions :      115  Temps d'execution :  18644
```

4) Choix d’optimisation énergétique au niveau du Testing

La partie de la validation de notre compilateur représente également un défi énergétique vu qu’on a cherché dès le début à réduire la consommation énergétique au niveau de cette étape. Pour comprendre nos choix pour réaliser cette fin, on doit d’abord vous expliquer comment la validation du compilateur est faite.
Pour valider notre compilateur, on a écrit un code .deca et à chaque code, on associe un fichier .des contenant la sortie désirée. Ainsi, durant l’étape de validation, le programme du test génère un fichier contenant la sortie qui va le comparer avec le fichier .des.

Cette comparaison consommait déjà un temps considérable puisque au début on a utilisé la commande **diff** qui parcourt le contenu des deux fichiers et après elle renvoie si les deux fichiers se ressemblent ou pas. Après, on a opté pour une optimisation qui nous a fait gagner 10 secondes pour les 50 tests rédigés. Cette optimisation consiste à échanger la commande **diff** par **cmp** qui renvoie **false** dès le premier bit qui diffère du fichier .des.