Impact environnemental et développement durable

- **Impact environnemental et développement durable**
  - Introduction
  - Reflexion en amont sur notre stratégie.
  - Application effective
    - Partie extension
  - Mesures et Classements
  - Impacts des procédures de tests
  - Bilan

Introduction

Depuis quelques années, il est demandé aux sociétés de toutes tailles et de tous secteurs de prendre en compte leur impact social et environnmentale, c'est ce qu'on appelle la RSE : Responsabilité sociétale des entreprises. Il s'agit en d'autre termes de la contribution des entreprises aux développments durables et à la transition écologique.

Pour cela, une politique dite 'RSE' doit être mise en place dans la structure. Celle-ci affecte tous les departements et secturs de la société, et notamment le secteur informatique. En effet, l'informatique joue un rôle plus qu'important dans la transition écologique.

Aujourd'hui, selon Greenpeace, le secteur informatique représente aujourd'hui 7 % de la consommation mondiale d'électricité, agissant comme un 7ème continent. Pour des organisations comme la WWF, il est urgent de réduire cette consommation, d'améliorer et d'optimiser les systèmes informatiques. Pour cela, de nombreuses stratégies ont été mises en place, et notamment le GreenCode.C'est cette aspect qui nous interessera pour l'impact environnementale de note compilateur.

Le GreenCode a pour objectif de reduire l'empreinte environnementale d'un service numérique en développant un code plus sobre et mieux fabriqué. En effet : l'optimisation du code a un impact direct sur la quantité d'énergie requise pour faire fonctionner un logiciel.

Un des aspects de notre projet etait donc d'optimiser un maximum notre code afin de rentrer dans cette politique de GreenCode.

Reflexion en amont sur notre stratégie.

Afin de minimiser notre impact impact environnemental, nous avons réalisé une réunion, brieve mais importante, en début de projet afin de determiner quelles pourraient être les différentes stréatégies et méthodes à mettre en oeuvre. Plusieurs idées sont ressorties.

La premier axe aborde l'efficacité du code assembleur. En effet, celui-ci est ensuite compilé par ima selon un nombre de cycle.

La première idée a donc été de prendre en compte les temps d'exécutions des instructions en assembleur donnés à la page 110 du poly. Ces temps, donné en faite en nombre de cycles interne, nous permettraient surement de réaliser des choix lors de la partie C du projet entre deux issues possible. Ce temps d'excection pourrait peser dans la balance, vis à vis de la complexité du code et de son efficacité.

Le deuxième axe a été d'améliorer le code java permettant de verifier le programme déca et de l'obtenir en langage assembleur.

Application effective

Lors de la realisation du projet, nous avons essayé un maximum de nous rappeler cette aspect lors des prises des décisions et de l'écriture des lignes de code. Pour ce qui est de l'étape A, nous avons conclu qu'il était difficile de modifier notre code étant donné que celui-ci est très restreint et plutôt isnignifiant vis à vis du reste du projet.

Lors du codage de l'étape B, l'objectif à été de stocker un maximum les valeurs pouvant être réutilisé par la suite afin de ne pas les redeterminer lors de nombreux appelles de méthodes. En effet, au vu des nombreuses classes et du large héritage des classes entre elle, il était possible, suite à une chaine d'appel de méthode, de récupérer certaine valeur, comme le type par exemple ou la définiton de la variable concernés.

Nous avons considéré plus interressant le choix de stocker ces valeurs afin qu'elle soit plus facilement et efficacement accéssibles. Par exemple, afin de gérer les registres de manière simple et efficaces, nous pouvons determiner les disponibilités de ceux-ci grâce à un simple appel à méthode. Afin de limiter ces accès, nous avons également réduit aux maximum le nombre de méthode et d'optimiser l'héritage, cela permettant de limiter les accès à d'autre de classe.

Nous avons eu aussi à choisir parfois entre la lisbilité du code, sa factorisation, et son optimisation en terme d'impact énergétique, 3 visions ayant leurs propres avantages et inconvénaint. Nous étions d'accord qu'il ne fallait pas non plus sacrifier les deux premiers points pour le troisième.

Lors de la réalisation de l'étape C, nous avons essayé de prendre en compte les cycles internes de chaque instruction. Par exemple, l'utilisation de "STORE" et "LOAD" plutôt que des "PUSH" et "POP" permet déja de réduire le temps d'instruction. Un choix a également du être fais pour ce qui est de la partie extension. Nous avons du régaliser l'extension "TAB" permettant d'utiliser des tableaux.

Partie extension

Comme décrit dans la documentation **Extension**, nous avons choisi pour stocker en mémoire les tableaux d'utiliser le tas. Hors, l'utilisation mémoire du tas au lieu de celui-ci de la pile tend à rendre le code moins efficace. En effet, pour allouer un espace dans le tas grâce à la commande `NEW`, cela nécessite 16 cycles à l'éxécution supplémentaire par rapport à un stockage dans la pile. Nous étions conscients des impacts sur l'efficacité mais semblait pour nous d'être un mal nécessaire pour la bonne réalisation de l'extension.

Mesures et Classements

Une première appréciation possible de l'efficacité énergétique de notre programme est de comparer les performances de notre algorithme à ceux des autres groupes. Grace à la commande `ima -s`, nous pouvons déterminer le nombre de cycles pour l'éxécution du programme. Les programmes des différents groupes ont été comparés sur 3 programmes différents qui exploitent des parties différentes du code.

Les scores attribués aux différents programmes sont : - Syracuse : 1708 cycles - Ln2 : 16728 cycles - Ln2_fct : 18669 cycles

Nous avons donc un score total de 52377 cycles pour l'ensemble du concours de performances. Cela nous place donc dans la moyenne (13ème sur 23) au niveau du nombre de cycles par rapport aux autres groupes ayant inscrit leur score. (Sur plus de 50 équipe).

Cela nous donne une bonne première indication sur l'efficacité énergétique de notre programme. On remarque alors que sur la partie sans-objet du projet, nous avons en moyenne un nombre de cycles supérieurs aux autres groupes. Toutefois, sur le test Ln2_fct qui utilise des classes, donc la partie avec objet, la tendance est inversée et notre code tend à effectuer moins de cycles que les autres groupes.

On peut donc penser que la réflexion que nous avons eu en amont sur la factorisation de notre code, des choix de conceptions et d'utilisations de fonctions assembleurs spécifiques a eu plus d'effets sur la partie objet que sur celle sans objet. Une explication possible est la meilleure compréhension et maîtrise des parties du code sur la fin du projet. Cela nous a permi de mieux comprendre et prédir les comportements de notre futur programme et ainsi pouvoir mieux l'optimiser.

Impacts des procédures de tests

Au cours du projet, l'une des actions la plus couteuse énergétiquement étaient celle de la procédure de tests. À la fois nécesaire pour s'assurer du bon fonctionnement du code et évitez les régressions, c'est une étape qui demande beaucoup de ressources au processeur. De nombreuses étapes sont nécessaires : vérifications d'erreurs ou d'absences d'erreurs, de bonne sorties du programme, de bonne

compilation et recompilation. Autant d'actions devant être effectuées de manière régulière. Un processus tellement demandant avec un nombre de tests élevés que l'éxecution de la commande de test `mvn test` prenait à la fin du projet en moyenne 5 minutes à finir.

Nous avons donc décidé au cours du projet de limiter le nombre de lancement de tests. Une personne était responsable du lancement de la batterie de tests de manière régulière pour vérifier la non régression du programme, mais sur une base journalière (hors moments de validation de fin d'étape pour vérifier le bon fonctionnement et fix les bugs), au lieu que plusieurs fois par jour. Les membres du groupe étaient encouragés à utiliser des tests spécialisés sur une fonctionnalité particulière, plutôt que des tests conséquant qui vérifient en même temps plusieurs propriétés, ces genres de tests étant réservés pour le script automatisé de `mvn test`. Par ce principe, la plupart des tests utilisés ne dépassent pas un nombre de cycles de 200.

On estime que nous avons lancé la commande `mvn test` au moins 150 fois au cours du projet. En prenant la denière version de nos scripts de tests, nous obtenons les informations suivantes avec la commande `/usr/bin/time` :

```
Command being timed: "mvn test"
    User time (seconds): 388.43
    System time (seconds): 37.01
    Percent of CPU this job got: 184%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 3:50.98
    Average shared text size (kbytes): 0
    Average unshared data size (kbytes): 0
    Average stack size (kbytes): 0
    Average total size (kbytes): 0
    Maximum resident set size (kbytes): 295780
    Average resident set size (kbytes): 0
    Voluntary context switches: 481322
    Involuntary context switches: 64598
    Swaps: 0
    File system inputs: 328
    File system outputs: 63456
    Socket messages sent: 0
    Socket messages received: 0
    Signals delivered: 0
    Page size (bytes): 4096
    Exit status: 0
```

On peut donc estimer un coût énergétique de la totalité des lancement de cette commande au cours du projet.

Bilan

Maintenant que le projet est fini, nous pouvons tirer certaine conclusion sur notre approche environnemental. Nous pouvons affirmer que c'est un aspect de notre code que nous n'avons pas ignoré. Ce facteur rentrait en compte dans les prises de décision. Nous avons, au cours du projet, fortement limité l'utilisation du script de test, très energivore, et preferé utiliser des tests individuels pour corriger ou tester des fonctionnalités précises. Cepandant, il est également certain que nous ne nous sommes pas focalisés sur cet aspect. L'éfficacité du code et la qualité technqiue du produit ont très souvent primé sur l'impact environemental. Cela peut s'expliquer par deux raisons : - Un manque certains de temps et de compréhension nous obligeant à selectioner entre éfficacité et impact environnemental, alors qu'il était très certainement possible d'allier les deux. - Une première partie du projet on nous nous étions vraiment focalisé sur la partie technique, voulant nous assurer d'un rendu fonctionnel.

En bref, nous avons eu une stratégie afin de réduire notre impact envrionnemental mais également quelques difficultés dans sa réalisation, ce qui donne un impact moyen comme peut le montrer le classsement.