Grenoble INP – ENSIMAG, UGA
École Nationale Supérieure d'Informatique et de Mathématiques Appliquées

Projet Génie Logiciel

Impact énergétique

Yanis BOUHJOURA, Hugo DABADIE, Garance DUPONT-CIABRINI, Marek ELMAYAN, Julien LALANNE

GL09 Groupe 2

27 janvier 2022

Table des matières

1 Moyens mis en oeuvre pour évaluer la consommation d'énergie

1.1 Moyens mis en oeuvre

En considérant le fait que la grande partie de l'énergie consommée serait dû à l'utilisation d'ordinateurs, nous sommes venu à la conclusion que la meilleure mesure de consommation d'énergie serait de faire un bilan de consommation en électricité ainsi qu'une mesure de stockage de nos fichiers.

1.2 Un bilan énergétique de ce projet

Durant ce projet, nous étions 5 à travailler, tous avec notre machine personnelle. Comptons que chaque ordinateur portable consomme en moyenne 50W et que nous avons travaillé environ 6h30 par jour, 6 jours par semaine pendant 3 semaines. Cela fait alors 5 x 3 x 6 x 6.5 x 50 = 30 000 Wh. On peut également ajouter à cela la consommation énergétique des serveurs GitLab. Admettons que trois ordinateurs fixes soient mobilisés pour faire fonctionner GitLab à plein temps. Un serveur de ce genre consomme environ 100W et tourne 24h/24. Cela fait donc 3 x 7 x 24 x 100 = 50 000 Wh qu'il faut répartir entre les environ 50 équipes réalisant le projet. Finalement, le serveur GitLab a consommé environ 1 000 Wh par équipe. De plus, nous utilisons des écrans externes pour un meilleur confort de travail. Ceux-ci consomment environ 20W soit une consommation en écran de 12 000 Wh.

Enfin, si l'on additionne ces consommations cela fait 43 000 Wh soit un équivalent de 6,5€ au prix de EDF. Cela ne paraît pas beaucoup mais si l'on compte sur l'ensemble des équipes cela fait 21 000 000Wh.

On peut ajouter à cela la consommation énergétique de transport pour aller travailler tous les jours qui est très faible étant donné que deux d'entre nous viennent en vélo et les trois autres utilisent le tramway.

1.3 Bilan énergétique du stockage des données

Un autre point important du bilan énergétique est le stockage des données. En effet, stocker inutilement beaucoup de données représente un gros gâchis énergétique. Notre projet complet pèse 55Mo donc le stocker sur chaque machine personnelle, chaque session des ordinateurs de l'ENSIMAG et sur Git utiliserait environ 600Mo. Cela peut être évité en ne stockant pas tous les fichiers qui n'ont pas besoin de l'être. Après un nettoyage, le projet ne pèse plus que 20Mo.

2 Impact de nos choix de compilation de Deca vers Ima

2.1 L'économie de l'espace de stockage - L'option -c

Le stockage des données étant une cause importante de consommation énergétique, nous avons pensé que c'était une bonne idée de réduire la place occupée par les fichiers assembleurs en sortie de compilateur. En effet, un fichier assembleur prend beaucoup plus de place qu'un fichier Deca. Nous avons en effet compté un rapport de 5 entre la taille de nos fichiers Deca et Assembleur. Nous avons donc décidé d'ajouter la commande -c à notre compilateur. Celle-ci permettra d'afficher ou non des commentaires dans le code assembleur. Si l'on ne l'utilise pas, le fichier sortant ne comporte pas de commentaires.

C'est l'option de base car pour la plupart des utilisations les commentaires assembleur sont inutiles. Avec l'option -c cependant, des commentaires apparaissent dans le code assembleur ce qui peut être utile dans des situations plus spécifiques. Enlever ces commentaires a permis de transformer ce rapport de 5 évoqué précédemment en un rapport de 4.

2.2 La performance de notre compilateur

Le code produit par notre compilateur est également plutôt performant, en comparaison aux scores des autres équipes sur le "palmarès projets GL 2022" nous sommes classés 9ème. Cela est très bien sur un plan énergétique car cela permet de consommer moins de ressources lors de l'exécution d'un programme .ass tout en réduisant la place que celui-ci prend sur le disque dur.

Cela pourrait cependant être mieux. Nous n'avons en effet pas réussi à implémenter l'instruction FMA qui aurait beaucoup réduit la consommation de l'exécution du programme avec IMA, réduisant énormément le nombre de tours de boucle que fait IMA dans des expressions de la forme a * x + b.

3 Processus de validation

Tout au long du développement de notre compilateur nous avons essayé de réduire l'impact environnemental du processus de validation. Ainsi, nous avons lancé des tests de manière individuelle le plus souvent possible. C'est à dire qu'au lieu de lancer la commande mvn verify (très gourmande en énergie car lance plus de 500 tests) ou encore le script all-test-context sur tous les sous répertoires de src/test/deca/context nous avons par exemple privilégié le lancement de test_context sur un fichier ou de all-test-context sur un sous-répertoire visant une fonctionnalié. Nous n'utilisons la commande mvn verify qu'un peu avant les suivis afin d'être sûr que tout fonctionnait correctement.

De plus, nous avons fait attention aux tests trop lourds (demandant beaucoup de ressources de stockage) et utilisant une grande partie des ressources énergétiques du pc tels que les tests de l'option -P de decac. Nous n'avons réalisé ces tests qu'une fois pour obtenir les mesures nécessaires et sans push les fichiers concernés sur git pour ne pas demander plus de ressources à git inutilement. Ainsi, les seules traces restantes de ce test sont les diagrammes d'utilisation des coeurs de la machine sur lequel il a été lancé ainsi que les mesures de temps en comparant decac (en série) et decac -P (en parallèle).

Dans une démarche de réduction de l'impact énergétique de nos tests, nous avons également décidé de réduire au maximum le nombre de fichiers assembleur stockés. Nous nous sommes pour cela interdit de push sur git nos fichiers assembleur car cela voudrait dire qu'ils seraient stockés sur chacune de nos machines ainsi que sur le serveur distant. Cette même méthode a été utilisée avec le dossier target dont nous avons pris soin de vider le contenu régulièrement à l'aide de la commande mvn clean.

4 Impact énergétique de l'extension

4.1 Le stockage des tableaux

Contrairement à une variable classique, un tableau requiert beaucoup plus de place pour être déclaré. Là où une variable prendrait une case mémoire, un tableau de taille n en prendra n + 1. Il faut donc mémoriser que chaque opération faite sur ces tableaux sera n fois plus gourmande que la même opération sur un type entier ou flottant.

D'un autre côté, si ils sont bien utilisés, les tableaux permettent au contraire de gagner en efficacité de calcul ainsi qu'en taille de fichier assembleur.

En effet, itérer sur des cases mémoire consécutive sera plus efficace et demandera moins d'énergie que d'aller chercher à des endroits différents des variables.

Pour ce qui est de l'assembleur, là aussi les tableaux peuvent se montrer efficaces : définir un tableau par l'adresse mémoire de son premier élément fera un fichier beaucoup moins lourd que de créer plusieurs variables séparées.

4.2 Les défauts des calculs matriciels

Comme dit précédemment, le calcul sur des tableaux et donc le calcul matriciel sont très énergivores. On peut vite se retrouver à avoir une complexité élevée et donc une consommation de ressources accrue. C'est pourquoi nous avons décidé dans notre extension d'optimiser cela. Nous avons par exemple utilisé l'algorithme de Strassen pour calculer des produits matriciels (qui est l'opération matricielle classique la plus demandante en ressources). Cela permet de réduire la forte demande énergétique de l'extension.