Projet Génie Logiciel

Documentation : Impact Énergétique

Gr2 - Gl10 - 2023 - 2024

Membres du groupe:

CHRIF M'HAMED Mohamedou
DIAB Dana
KONE Madou
MOHAMED AHMED Mohamed Lemine

1. Comparaison de complexité temporelle : decac vs javac

Pour obtenir une vision générale de la complexité de notre compilateur, nous avons élaboré un script en **Python** qui génère des fichiers Java et Deca. Utilisant une boucle **while** avec la condition i < pas, où le pas varie de 0 à 1000 avec un intervalle de 500, à chaque itération, nous exécutons les commandes **decac** et **javac** puis mesurons le temps d'exécution pour chaque fichier généré.

Enfin, nous représentons ces données sous forme d'un graphique qui affiche les temps d'exécution de `decac` et `javac` en fonction des différentes itérations de la boucle. Cette démarche nous permet d'obtenir des informations sur la performance de notre compilateur à différentes échelles d'entrée, offrant ainsi un aperçu de sa complexité.

```python
def genere_fich_java() :

    data_java = []
    for pas in range(0 , 10000 , 500) :
        fich = open("test.java", 'w')

        prog = f"""
        class test {{
            public static void main(String[] args) {{
                int i = 0 ;

                while(i<{pas}) {{
                    System.out.println(i);
                    i=i+ 1 ;
                }}
            }}
        }}
        """

        print(prog , file=fich)

        commande = "javac test.java"
        try  :
            t1 = time.process_time()
            subprocess.run(commande , check = True ,  shell=True , stdout=subprocess.PIPE , text=True)
            t2 = time.process_time()
            data_java.append(t2 - t1)
        except subprocess.CalledProcessError as e  :
            print(f"could not run javac : {e}")
```

Explication du script :

Ce processus d'évaluation de la complexité de notre compilateur implique la génération systématique de fichiers Java et Deca en fonction de paramètres spécifiques. Le script Python utilise une boucle `while` avec une condition déterminée par la variable `i` et un pas variant de 0 à 10000 avec un incrément de 500. Pour chaque itération, les commandes `decac` et `javac` sont exécutées, et le temps d'exécution de chaque fichier généré est mesuré.

L'objectif est d'observer comment le temps d'exécution évolue en fonction de la taille des fichiers sources, représentant ainsi la complexité du compilateur. En analysant les données recueillies, nous pouvons visualiser ces variations et comprendre comment les performances du compilateur sont impactées par des entrées de différentes tailles.

Le résultat final est un graphique illustrant de manière graphique les temps d'exécution respectifs de `decac` et `javac` en fonction de différentes tailles d'entrée, ce qui fournit des informations essentielles sur la performance du compilateur.