Université de Lille 1 SCIENCES ET TECHNOLOGIES

L'impact énergétique des algorithmes de tri

DIEULIN MAMBOUANA

ETIENNE WATTEBLED

Enseignant : Maxime COLMANT	**Module** : OPL
Établissement : Master IAGL - Université Lille 1	Lille, le 11 janvier 2016

TABLE DES MATIÈRES

INTRODUCTION

Ce document présente l'analyse des résultats tirés de l'utilisation d'un outil de calcul de l'impact énergétique des programmes informatiques sur la consommation globale d'un système d'information. Un outil d'analyse de la consommation énergétique est un software configurable capable d'estimer la consommation d'un processus en temps réel. Cela présente un grand intérêt dans la mesure où il force de constater qu'une consommation maîtrisée d'énergie d'un programme informatique est un facteur d'amélioration de ses performances et donc une garantie d'une meilleure expérience d'utilisation de l'application.

L'objectif principal de ces travaux est donc d'utiliser un outil de profilage de la consommation d'énergie afin de calculer l'impact des processus liés aux algorithmes de tri rédigés en différents langages sur la consommation globale d'un système d'information. Dans ce cadre, nous avons opté pour *PowerAPI* [1] comme software pour l'acquisition des métriques fournies par les différents composants d'un système tels que les capteurs ou encore les canaux. Il s'agit d'un outil expérimental d'analyse de la consommation énergétique fourni par l'*INRIA*[2].

L'analyse des résultats obtenus à partir des machines *DELL* de type *D07S* montre que des données mesurables ont été effectivement trouvées pour les différents processus liés aux algorithmes de tri que nous avons implémenté. En particulier on constate une forte différence de consommation entre les trois langages en fonction des algorithmes traités.

[1] *http://www.powerapi.org/*

[2] *http://www.inria.fr/equipes/spirals*

TRAVAIL TECHNIQUE

I. BUT

L'objectif principal de ce projet est de parvenir à déterminer de l'impact de la consommation d'énergie due à l'utilisation des algorithmes de tri sur nos ordinateurs. L'idée est d'utiliser *PowerAPI* comme outil pour récupérer les données brutes de consommation d'énergie associées aux processus exécutant trois algorithmes de tri écrits en autant de langages à savoir le *Java*, le *C* et le *Ruby*. A travers cette expérience, nous parvenons ainsi à l'évaluation de l'efficacité énergétique de ces algorithmes en fonction du langage utilisé.

II. OVERVIEW

La consommation d'énergie des systèmes d'information a fortement augmenté ces dernières années au vu du nombre de plus en plus croissant des ressources demandées par les logiciels. La connaissance de la consommation effective d'une application est donc un enjeu majeur dans la conception d'un logiciel informatique. En effet une forte consommation énergétique peut engendrer des fortes dépenses et favoriser l'usure précoce des composants d'une machine. Une consommation maîtrisée offre donc deux avantages principaux :

1. L'allongement de la durée de vie de l'environnement qui l'héberge en prévenant par exemple aux problèmes de surchauffe du CPU
2. Rationalisation des coûts au sein des centres informatiques

Afin de satisfaire ce besoin, plusieurs études et approches ont été proposées dans le but d'épargner l'énergie au niveau software et hardware. L'ensemble de ces efforts rentre dans le cadre du concept informatique qu'on appelle *green computing* [3], c'est-à-dire *l'informatique verte* ou *l'informatique durable.*

Comme le suggère le nom, *l'informatique durable* vise à réduire l'empreinte écologique, économique et sociale des systèmes d'information sur l'environnement qui l'entoure. En fonction de la stratégie visée, la réduction de l'impact négatif de ces systèmes peut se passer à différents niveaux :

- ✓ À la fabrication des différentes entités qui composent ces systèmes
- ✓ À l'utilisation de ces systèmes, on parle alors de la notion de la consommation d'énergie
- ✓ En fin de vie, avec la gestion des déchets, de la pollution et de l'épuisement des ressources

Le sujet de nos travaux porte sur la deuxième phase, à savoir la gestion de la consommation des systèmes d'information à l'utilisation.

[3] *http://www.cs.grinnell.edu/~davisjan/csc/105/2012S/articles/CACM-energy.pdf*

III. ALGORITHME ET IMPLEMENTATION

Cette section est axée sur la description des algorithmes de tris que nous avons mis en place. Pour cette expérience nous décidé d'implémenter trois algorithmes de tri en trois langages respectifs. Pour ce qui concerne les tris, nous avons choisi :

1. Le *tri comptage*[4] ou *counting sort* c'est-à-dire le tri casier
2. Le *tri fusion*[5]
3. Le *tri à bulles*[6] optimisé

TRI COMPTAGE

```
// Initialisation des variables
tabComptage[borneSuperieure + 1]
tailleTab = taille(tab) - 1
x = 0

// Init du tableau de comptage à 0
pour i = 0 à borneSuperieure:
   tabComptage[i] = 0
finPour

// Création du tableau de comptage
pour i = 0 à tailleTab:
   tabComptage[tab[i]]++
finPour

// Création du tableau trié
pour i = 0 à borneSuperieure:
   pour j = 0 à tabComptage[i]:
      tab[x++] = i
   finPour
finPour
retourne tab
```

TRI À BULLES OPTIMISE

```
// version optimisée
procédure tri_bulle(tableau t)
   i = longueur(t)
   échange = vrai
   tant que ((i>0) ET (échange)) faire
      échange = faux
      pour j allant de 0 à i-1 pas 1 faire
         si (t[j] > t[j + 1]) alors
               tmp = t[j]
               t[j] = t[j+1]
               t[j+1] = tmp
               échange = vrai
         fin si
      fin pour
      i = i - 1
   fin tant que
fin procédure
```

[4] *https://fr.wikipedia.org/wiki/Tri_comptage*

[5] *https://fr.wikipedia.org/wiki/Tri_fusion*

[6] *https://fr.wikipedia.org/wiki/Tri_%C3%A0_bulles*

TRI FUSION

```
'''procédure''' tri_fusion(tableau t)
   n = longueur(t)
   si n > 1 alors
      u = tri_fusion(t[1], ..., t[n / 2]) // 1ère moitié
      v = tri_fusion(t[n / 2 + 1], ..., t[n]) // 2ème moitié
      a = 1 // pointeur pour parcourir les éléments de u
      b = 1 // pointeur pour parcourir les éléments de v
      '''pour''' i '''allant de''' 1 '''à''' n '''faire'''
         '''si''' (a <= longueur(u)) // si tous les éléments de u n'ont pas été parcourus
            et (b > longueur(v) // et si tous les éléments de v ont été parcourus
            ou u[a] ≤ v[b]) '''alors''' // ou si l'élément courant de u est plus petit
            t[i] = u[a] // on choisit l'élément courant de u comme élément suivant de t
            a = a + 1 // on avance dans u
         '''sinon'''
            t[i] = v[b] // on choisit l'élément courant de v comme élément suivant de t
            b = b + 1 // on avance dans v
         '''fin si
      '''fin pour'''
   '''fin si'''
   renvoyer t
'''fin procédure'''
```

IV. ARCHITECTURE

Le mobile de mes travaux est d'intégrer un profileur d'énergie au sein d'une série de projets dans le but de quantifier leur consommation d'énergie. Cette expérience utilise *PowerAPI* comme outil pour mesurer automatiquement la consommation d'énergie et les algorithmes de tri écrits en trois langages comme base de test à savoir :

- ✓ *Java* dans sa version 1.7
- ✓ *Ruby*
- ✓ Langage *C*

Pour ce qui concerne la partie *Java* sur les implémentations des algorithmes, on retrouve trois *packages* [7] fonctionnels nommés *util*, *algorithmes* et *execution*. Le package *util* fournit les outils nécessaires à la manipulation d'un tableau. C'est par exemple le cas des fonctionnalités d'initialisation, d'affichage ou de recherche du nombre maximal dans un

[7] *https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-packages*

tableau d'enter. Le package *algorithmes* regroupe les implémentations des trois algorithmes de tri à savoir le *tri comptage*, le *tri fusion* et le *tri à bulles*. En fin le package *execution* contient la classe avec la méthode principale qui lance les trois algorithmes de tri sur un jeu de données saisi par l'utilisateur.

Pour ce qui concerne la partie écrite en *C*, on retrouve trois fichiers *.c* associés aux implémentations des trois algorithmes de tri. Dans chacun desquels, on retrouve les fonctions pour la manipulation des tableaux, la fonction *main* ainsi l'implémentation de l'algorithme associée.

Comme pour la partie précédente, la partie en *Ruby* est essentiellement composée de trois fichiers *.rb* implémentant respectivement les algorithmes de tri et fournissant des méthodes principales nécessaires pour lancer ces algorithmes.

V. UTILISATION

Pour procéder à l'analyse de la consommation d'énergie des différents processus liés aux algorithmes de tri, il suffit de générer dans un premier temps les exécutables associés aux différents algorithmes de tri pour chacun des langages d'édition et de les lancer de la manière suivante :

Langage	Tri fusion	Tri par comptage	Tri à bulles optimisé
Java	*jar cmf manifest fusion src/*	*jar cmf manifest comptage src/*	*jar cmf manifest bulles src/*
C	*gcc TriFusion.c*	*gcc TriParComptage.c*	*gcc TriABullesOptimise.c*
Ruby	*irb TriFusion.rb*	*irb TriParComptage.rb*	*irb TriABullesOptimise.rb*

L'étape suivante consiste en la récupération du *PID* du processus associé à l'algorithme lancé. Cela peut se faire sur Linux avec la commande suivante *ps ax | grep «nom processus»*. On peut par la suite lancer *PowerAPI* avec la commande suivante :

```
./bin/powerapi [TDP: Double (W)] [Frequency: Long (ms)] [PIDS: Long*]
```

Commande: *TDP : Thermal Design Power*
Frenquency : la frequence en ms
PIDS : les PID de processus

EVALUATION

I. COMPLEXITE

Pour cette expérience nous avons analysé la consommation des implémentions de trois algorithmes de tri sur trois langages de programmation à savoir le Java, le C et Ruby. Il s'agit des algorithmes de tri fusion, de tri par comptage et de tri à bulles optimisé. Ces algorithmes ont des complexités différentes et se résument comme suit :

Tri	Complexité
Fusion	*n*log n*
Par comptage	n^2
A bulles optimisé	*n + p (*)*

(*) : p : plus grand entier du tableau – plus petit entier du tableau

II. PERFORMANCE

Pour notre expérience, nous avons pris comme jeu de données un tableau à une dimension avec une taille maximale de 100000 éléments, la limite permise qui assure une exécution sans erreurs. Toutes nos expériences ont été réalisées sur des machines DELL de type D07S fournies par l'Université Lille 1. Ces machines ont une *enveloppe thermique (TDP)* de 84W.

Données de l'expérience	
Taille tableau	*100*
TDP	*84W*
Nb Algorithmes / langage	*3*
Langages	*Java, Ruby, C*
Algorithmes de tri	*Tri fusion, tri par comptage, tri à bulles optimisé*

Après le lancement des exécutables associés aux différents algorithmes de tri nous avons pu mesurer les temps d'exécutions suivants :

Tri fusion	
Langage	**Temps d'exécution**
Java	*80 ms*
C	*20 ms*
Ruby	*329 ms*

Tri par comptage	
Langage	**Temps d'exécution**
Java	*100 ms*
C	*5 ms*
Ruby	*121 ms*

Tri à bulles optimisé	
Langage	**Temps d'exécution**
Java	*12 539 ms*
C	*28 470 ms*
Ruby	*890 406 ms*

On peut donc constater que le langage *Ruby* a un temps de traitement des algorithmes beaucoup plus important que les deux autres langages. Cela peut se justifier par le fait qu'il soit un langage entièrement interprété. On observe des meilleurs résultats sur les exécutions faites sur *Java*, ce qui est cohérent vu que Java est un langage semi-compilé. Enfin, le langage *C* montre une excellente performance, à l'exception des exécutions effectuées sur le tri à bulles. Cela se justifie notamment par sa caractéristique d'être un langage entièrement compilé.

III. ANALYSE DE LA CONSOMMATION

Après l'évaluation des différents processus associés aux algorithmes de tri avec *PowerAPI*, nous obtenons les résultats de consommation suivants :

Tri fusion	
Langage	**Consommation moyenne**
Java	*29,63 joules*
C	*11,47 joules*
Ruby	*29,08 joules*

Tri par comptage	
Langage	**Temps d'exécution**
Java	*19,09 joules*
C	*0,08 joules*
Ruby	*27,59 joules*

Tri à bulles optimisé	
Langage	**Temps d'exécution**
Java	*15,4 joules*
C	*14,46 joules*
Ruby	*30,75 joules*

Les résultats montrent que le langage C a une consommation moyenne moins élevée et donc moins énergivore par rapport à Ruby et au Java. Cependant, les résultats ci-dessus ne tiennent pas compte du temps, vu qu'il s'agit d'une simple moyenne. Vu l'importance capitale

que représente le temps, nous avons refait les calculs en représentant la somme des moyennes des consommations sur des intervalles de temps de 5 ms :

Tri fusion	
Langage	**Consommation moyenne**
Java	*29,63 joules*
C	*11,47 joules*
Ruby	*29,08 joules*

Tri par comptage	
Langage	**Temps d'exécution**
Java	*19,09 joules*
C	*0,08 joules*
Ruby	*27,59 joules*

Tri à bulles optimisé	
Langage	**Temps d'exécution**
Java	*200,2 joules*
C	*448,45 joules*
Ruby	*18 6665 joules*

En analyse de ces valeurs met en évidence le caractère très énergivore de Ruby notamment dans le tri à bulles optimisé. En revanche le Java et notamment le langage C ont une consommation beaucoup plus faibles consomme.

La forte consommation des algorithmes en Ruby est accentuée par la consommation de l'interpréteur puisqu'il utilise beaucoup le processeur. Dans un cas similaire, les algorithmes en Java voient leur consommation incrémentée de celle de la *JVM* (*Java Virtual Machine*). Par conséquent, le langage C fournit des résultats beaucoup plus meilleurs, hormis le tri à bulles optimisé, car il ne requiert pas de machine.

CONCLUSION

Le langage Ruby n'est en aucun cas un langage intéressant. En effet, celui-ci prend beaucoup trop de temps et est vraiment énergivore. Le langage C, en enlevant le tri à bulles, est le langage le plus intéressant, il est très rapide et consomme très peu. Pour ce qui est de Java, il est moyen mais serait un très bon compromis dans le cas où les algorithmes seraient exécutés de façon équiprobable