TP2 : Analyse de performances

Auteur : Alain Mérigot (Université Paris Saclay)

Compte rendu de TP par équipe de 4 à déposer sur ecampus au plus tard le 5 décembre sous forme de fichier pdf nommé tp2–nom1-nom2-nom3-nom4.pdf)

1 Objectifs du TP

Ce TP a pour objectifs de permettre :

— une familiarisation avec les méthodes de mesure de performances de programmes.
— l'étude de l'impact des options de compilation sur les performances
— la mise en évidence de l'influence des caches sur les temps d'exécution

Il repose sur une instruction du pentium `rdtsc` permettant de lire un compteur interne du processeur systématiquement incrémenté à chaque cycle (*TimeStamp Counter*) [1] et de le copier comme un `uint64_t` dans les registres `eax` et `edx`.

Des fonctions définies dans le fichier `tsc.h` permettent d'accéder en C à l'état de ce compteur avant et après l'exécution de la procédure que l'on veut évaluer et de convertir le résultat en double pour une manipulation plus facile.

```
#include "tsc.h"
...
long long debut, fin; // les valeurs du TSC sur 64 bits (long long)
double dt ;      // le nombre de cycles écoulés en double
...
debut = start_timer();// renvoie la valeur du TSC au début
 //Partie du programme dont on mesure le temps d'exécution.
fin = stop_timer(); //renvoie la valeur du TSC après la fonction
dt=dtime(debut, fin); // Nombre de cycles d'horloge processeur en double
```

Il faut exécuter la fonction à analyser plusieurs fois. Les traitements se feront sur des vecteurs ou des matrices et nous afficherons le nombre de cycles par point.

On retirera les mesures aberrantes (trop élevées [2]) et on prendra la moyenne (ou si on préfère la médiane ou le minimum des valeurs).

Le fichier `tp2.c` comprend la plupart des fonctions du TP déjà programmées. Il faudra par contre faire varier des paramètres du programme (taille des vecteurs, type des données, etc) et/ou des options de compilation, et analyser et interpréter les résultats obtenus.

Il sera parfois nécessaire d'avoir des informations sur le processeur, par exemple pour convertir un nombre de cycles en secondes, ou avoir la taille des caches, etc. Toutes ces informations peuvent être obtenues par diverses commandes :

— `lscpu` (donne des information détaillées notamment sur la tailled es caches)

1. Il existe de nombreux autres compteurs matériels dans le pentium (pour mesurer les défauts des caches ou des TLB, les mauvaises prédictions, les accès mémoire, etc). Mais seul le compteur de cycles TSC peut être accédé sans être en mode privilégié (`root`) (et donc être utilisé en TP).

2. Pour quelle(s) raison(s) voit-on de petites dispersions dans les mesures ? Et pourquoi a-t-on parfois des dispersions très importantes ?

— `cat /proc/cpuinfo` (donne notamment les fréquences de chaque processeur. Attention, ces fréquences peuvent varier de manière importante, notamment sur un portable.)

On peut forcer l'exécution sur un processeur spécifique. Par exemple pour forcer l'exécution de `tp2` sur le processeur 2, faire :
`taskset -c 2 ./tp2`

2 Optimisations du compilateur

Nous utiliserons le compilateur `gcc`. Il comprend de nombreuses options, notamment pour l'optimisation.

— `gcc tp2.c` ou `gcc -O0 tp2.c` aucune optimisation
— `gcc -O1 tp2.c` ou `gcc -O tp2.c` quelques optimisations
— `gcc -O2 tp2.c` niveau d'optimisation à utiliser par défaut pour le TP, sauf indication contraire.
— `gcc -O3 tp2.c` optimisations agressives avec notamment vectorisation SIMD (qui rendent les résultats plus difficiles à interpréter)

On peut voir la liste des optimisations réalisées par le compilateur au niveau `-Oi` (avec `i` variant de 0 à 3) par
`gcc -c -Q -Oi --help=optimizers`
et la significations des différentes optimisations par
`gcc --help=optimizers`

Pour quelques fonctions du fichier `tp2.c`, étudier l'évolution des performances avec le niveau d'optimisation.

3 Etude de diverses fonctions

Les différents programmes dont on veut analyser les performances seront exécutés pour différentes tailles des objets, en faisant varier le paramètre `N` qui contrôle la taille des objects : les vecteurs ont (N^2) éléments et les matrices sont ($N \times N$). On peut également redéfinir la macro `TYPE` qui donne le type des données.

Le programme permet de modifier la plupart des paramètres en ligne de commande. Par exemple
`gcc -O2 -DN=500 -DTYPE=int tp2.c -o tp2`
compile en redéfinissant le paramètre `N` à 500 et le type des données en `int`. Ceci permet, avec des boucles réalisées par le shell, de faire simplement un ensemble de mesures :
`for n in 100 500 1000; do gcc -O1 -DN=$n tp2.c -o tp2; ./tp2; done`

Les programmes seront compilés avec l'option `-O2`. On pourra étudier les résultats en l'absence de toute optimisation (option `-O0`), ou avec des optimisations avancées (option `-O3`). On reportera les résultats en nombres de cycles par itération.

3.1 Mise à zéro d'un vecteur

La fonction `zero()` met un vecteur de taille **N*N** à zéro. C'est un exemple de programme limité par les accès mémoire (*memory bound*). Regarder l'évolution du temps d'exécution pour différentes taille de vecteurs.

On peut également estimer ainsi le débit d'écriture en mémoire. Pour cela, il faut compiler en `-O3` et faire varier le nombre d'octets des données : `char`, `short`, `int`, `float`, `double`. Quelle valeur approximative trouve-t-on pour ce débit ?

3.2 Copie de matrices

On considère maintenant la copie d'une matrice `[N][N]` dans une autre. Il existe dans le fichier `tp2.c` deux versions de cette fonction : en balayant lignes puis colonnes (`ij`) ou colonnes puis lignes (`ji`).

Comparer les performances des deux versions en fonction de `N` et expliquer l'origine des différences. Quel est l'impact de la taille des données `N` ?

3.3 Addition de matrices*

Mêmes questions pour l'addition de deux matrices qui existe en deux versions suivant l'ordre du balayage.

3.4 Produit scalaire de deux vecteurs

On considère maintenant le produit scalaire de deux vecteurs de taille `N*N`. On prendra des vecteurs flottants, et on fera la mesure pour différentes valeurs de `N`.

3.5 Produit de matrices

3.5.1 Produit de matrices ijk de flottants 32 bits

Donner le temps d'exécution pour chaque itération d'un produit de matrices `N×N` en faisant des itérations `ijk`. On prendra différentes valeurs de `N`. Comment pourrait-on en déduire un ordre de grandeur de la taille des caches ?

3.5.2 Produit de matrices ikj

Même question, en procédant avec des itérations dans l'ordre ikj. Comment expliquer les améliorations obtenues ?

3.5.3 Produit de matrices ijk par blocs

On découpe maintenant la matrice en blocs de taille `B`, et on effectue la multiplication dans l'ordre `ijk` bloc par bloc. Pour `N=1000`, à partir de quelle taille de blocs a-t-on une amélioration significative ? Expliquer le résultat obtenu.

3.5.4 Produit de matrices avec transposition*

Une autre solution est de commencer par transposer (en copiant dans une autre matrice) une des matrices avant de faire la multiplication en `ijk`. Programmer la fonction correspondante et comparer les résultats avec les méthodes de multiplication de matrices précédentes. On considèrera les cas entiers et flottants.

4 Questions additionnelles [3]

4.1 Comparaison du produit scalaire et de la multiplication de matrices en flottant

Le produit scalaire flottant est sensiblement plus lent par itération que la multiplication de matrices (même en *ijk*), alors que les opérations effectuées dans la boucle interne sont comparables (deux accès mémoire, une multiplication et une addition). Pour quelle(s) raison(s) ceci arrive-t-il à votre avis ?

Proposer une méthode permettant d'améliorer le temps d'exécution du produit scalaire. La programmer et comparer avec la version initiale.

4.2 Multiplication de matrices en entier et en flottant

Comparer les temps d'exécution de la multiplication de matrices (`ikj`) en entiers et en flottants pour des données de même taille (`float` et `int` ou `double` et `uint64_t`). On se placera en optimisation `-O1`.

Comment peut-on expliquer les résultats obtenus ?

4.3 Produit scalaire en entier et en flottant

Comparer de même le produit scalaire (non optimisé) en entier et en flottant.

Commenter et expliquer les résultats obtenus.

4.4 Mesure de latences d'opérations

Comment pourrait-on étendre le TP pour estimer les latences de certaines opérations (additions, multiplications ou divisions) ? Proposer une méthode de mesure. Ne pas oublier que le pentium est superscalaire et que toute instruction s'exécutera s'il n'y a pas de dépendances. Evaluer ainsi les latences des additions, multiplications et divisions pour des types entiers et flottants.

3. et plus difficiles (mais optionnelles).