TP3 : Mesures de Performances

1 Objectifs

- ➢ Concepts, méthodes et outils de mesure de performances
- ➢ Quelques pistes sur l'optimisation de code ...

2 Analyse du code

MPEG2 est une norme de codeur-décodeur vidéo utilisée dans la plupart des applications de diffusion de séquences d'images.

- ➢ Le code d'un décodeur MPEG 2 est fourni dans le répertoire (ou dans http://www.mpeg.org/MPEG/video/mssg-free-mpeg-software.html).
- ➢ L'exécution du décodage d'une séquence se fera par la commande `mpeg2dec -b test_015.m2v` sachant que `test_015.m2v` est une séquence test d'images que vous pouvez visualiser avec `winamp` (par exemple).

2.1 Profiling

Mettez vous à la place d'un constructeur d'une platine de lecture vidéo.

- ➢ Au niveau des perfomances, il souhaite d'atteindre le "temps réel" sur le décodage de cette séquence afin d'avoir un affichage fluide.

Il se doute qu'il ne pourra réaliser le décodage entièrement avec du logiciel, et qu'il va devoir réaliser un morceau de silicium (Il s'agit là de l'optimisation ultime !) pour accélerer le traitement.

- ➢ Lequel ? !
- ① Utiliser une "vue flat" pour trouver le ou les bootleneck.

3 Optimisation du code avec gcc

L'excellent compilateur `gcc` propose un grand nombre d'optimisations parmi lesquelles beaucoup sont indépendantes du processeur cible :

- ➢ `-ftree-loop_im` ,
- ➢ `-ftree-ch` ,
- ➢ ...

Remarque : toutes les optimisations dont le nom commence par `-f` sont "machine independent" !

Ces optimisations peuvent être activées lors de la compilation grâce à l'option `-On` où n est un entier :

- ➢ -O0 : supprime toute optimisation.

➢ -O1 ou -0 : réalise des optimisations de niveau 1
Dans ce niveau, `gcc` tente de minimiser à la fois la taille du code exécutable produit **et** son temps d'exécution sans pour autant accroître dramatiquement le temps de compilation.

➢ -O2 et -O3 : ajoute au niveau 1 des optimisations supplémentaires

➢ -Os : permet de signifier au compilateur l'intérét pour un code de taille minimale.

3.1 Mesures globales et Instrumentations manuelles

On souhaite étudier l'addition de deux matrices

Dans ce cadre, on va être amené à réaliser différents tests sous diverses conditions de tailles (512, 1024 et 2048) ou de compilation (avec ou sans optimisations) par exemple :

➢ Il serait souhaitable que ce travail s'accompagne de l'élaboration d'un makefile permettant de revenir **facilement** sur un test effectué dans le passé.

① Récupérer une version utilisant l'allocation dynamique à partir du TD de l'UE "Programmation Impérative" de Licence 2.
Dans un premier temps le programme sera compilé **sans optimisation**.

② Quelle commande shell permet de mesurer la performance globale d'une telle opération sur une matrice carrée de 512 lignes et colonnes.

③ On souhaite comparer le temps consacré à l'allocation des matrices et à l'addition proprement dite.
L'utilisation de cette commande permet-elle de le faire simplement ?

④ Quelle fonction C permet de déterminer le temps processeur ?
Instrumenter le code manuellement afin de répondre à la question précédente.

⑤ J'ai "entendu dire" que la fonction `alloca()` était plus "rapide" que la fonction `malloc()`.
En quoi diffèrent-elles ? et pouvez construire un programme permettant de comparer ces deux approches ?
Attention au problème de granularité du temps !

⑥ Je vous propose une autre version de l'addition de matrices essentiellement différente dans la forme d'allocation pratiquée : la matrice est allouée comme un tableau 1D !
Quelles sont ses performances ?

3.2 Utilisation des optimisations

Dans un second temps, mesurer les temps d'exécution du programme compilé avec le niveau d'optimisations numéro 1 (-O1).
Pour ce test, on mesure le temps moyen pris par 100 additions des deux matrices de taille 1024.

① Quel est le gain moyen apporté par l'activation des optimisations de compilation ?

② Quelle partie du code bénéficie de ces optimisations ?

③ L'option de compilation -O1 est une macro qui permet d'activer de nombreuses optimisations. Lesquelles ?

④ Décrire l'optimisation `-ftree-loop-optimize`
Pensez vous qu'elle ait un intérèt dans le cadre de cet exemple et **pourquoi ?**

⑤ Mesurer son effet !

⑥ Décrire l'optimisation `-fif-conversion` .
Quelle optimisation apporte le meilleur gain de performances ?

⑦ Etudiez la évolution de la taille du code et de ses perfomances en fonction du niveau d'optimisation choisi.

3.3 Profiling

Dans le cadre de l'exemple de la multiplication de matrices (fichier `matmult.c`),

① utiliser `gprof` pour constater ce qui se passe (entre les deux fonctions de multiplications et avec des tailles différentes).

② expliquer, en suivant le lien qui est fourni dans le code source, le phénomène mis en évidence.

3.4 Des pistes de réflexions . . .

Choisir et réaliser un travail parmi ceux qui suivent :

① Essayer de développer une des améliorations proposée par :

http://www.mostang.com/~davidm/papers/expo97/paper/doc005.html#s4.5

② Quel est l'impact sur les performances du type (statique/dynamique) de bibliothèque choisi ?

③ Existe t'il des bibliothèques statiques (`libc` par exemple) de taille minimale ?

④ Essayer ce qui se trouve au bout de ce lien !

http://www.advancedlinuxprogramming-fr.org/doku.php?id=livre:chap9:code_assembleur_en_ligne#fn__2