TP2

Objectifs

Tester les performances réelles (temps de calcul) d'un programme de multiplication de matrices, et mesurer les apports de différentes techniques d'optimisation.

1- Prendre une copie du programme « mult_mat.c », vérifier le code, le compiler et l'exécuter avec TM=800 et noter le temps d'exécution.
Ajouter dans la première ligne du « main » le mot clé « register » devant int i, j, k, TM, …
Exécuter de nouveau avec TM=800 et comparer les deux temps d'exécution.

2- Exécuter ce programme avec des valeurs de TM = 200, 256, 300, 400, 500, 512, 600, 700, 800, 1000, 1200 et 2000. Reporter les temps d'exécution dans la colonne C du premier tableau du fichier « TP2_ressultats.ods » (à prendre sur moodle).

- La colonne D de ce fichier calcule le rapport entre le temps d'exécution et le nombre total de passages dans la boucle interne K.
- La colonne I calcule le nombre total d'instructions assembleur exécutées dans les 3 boucles imbriquées (I, J, K), sur la base des informations obtenues en consultant le programme assembleur (obtenu avec gcc –S).
- La colonne J calcule le nombre de cycles par instruction (fréquence du processeur de 1,596 GHz, disponible dans /proc/cpuinfo)

Interpréter les résultats par rapport aux performances obtenues au TP1 avec le simulateur de cache, en vous aidant du schéma « Intel_Nehalem_arch », disponible sur moodle et présentant l'architecture des « cores » des ordinateurs de la salle de TP :

- Pourquoi n'observe-t-on une dégradation des résultats pour TM=256 comme dans le TP1 ?
- Pourquoi la dégradation des résultats est constatée pour des valeurs de TM plus grandes que celles observées en TP1 ?

Conserver une copie de ce code, qui servira de point de départ pour les étapes suivantes.

3- Modifier le code précédent comme dans l'étape 4 du TP1 (découpage de la boucle j pour profiter de la localité spatiale sur B. Exécuter avec TM=800, 1200, et 2000. Reporter le temps d'exécution dans les cases vertes des tableaux correspondants du fichier « TP2_Resultats.ods » (Pas =8 et 16).
Quel pas offre le meilleur gain ? pourquoi ?

4- Modifier le code de la multiplication (de l'étape 2) comme dans l'étape 5 du TP1 (découpage de la boucle k pour profiter de la localité temporelle sur B. Exécuter avec TM=800, 1200, et 2000. Reporter le temps d'exécution dans les cases vertes des tableaux correspondants du fichier « TP2_Resultats.ods » (Pas= 8, 16, 40 et 100).
Quel pas donne les meilleurs résultats ? pourquoi ?

5- Modifier le code de la multiplication (de l'étape 2) de manière à effectuer la multiplication sur la transposée de B (tout en ayant calculé cette transposée). Exécuter et noter les résultats pour TM=800, 1200 et 2000.
Quelle localité est exploitée dans ce cas ? Est-ce cohérent avec les résultats précédents ?

6- Prendre une copie du fichier add_vect.c, vérifier le code, compiler et exécuter.
Recompiler avec l'option –O3, exécuter et comparer les résultats.
En consultant le code assembleur généré par gcc –O3 –S add_vect.c, on peut remarquer que la boucle interne est exécutée seulement 25 fois (add $16, %rax, et rebouclage si %rax != 400), et que cette boucle utilise les instructions movaps et addps, instructions de l'ensemble sse qui travaillent sur 16 octets à la fois (4 éléments consécutifs du vecteur).

Les options de compilation, -O1 et -O2, introduisent quelques optimisations simples sur certaines opérations, boucles et sur l'utilisation des registres, tout en limitant la taille du code (voir http://wiki.gentoo.org/wiki/GCC_optimization/fr).
L'option –O3 pousse l'optimisation plus loin, en utilisant des caractéristiques ou des particularités du processeur.

7- Recompiler le programme de multiplication de l'étape 4 avec l'option -O3. Reporter les performances pour TM=800, 1200 et 2000.
Grande surprise !

Une explication est fournie dans le fichier « TP2_resultats.ods »