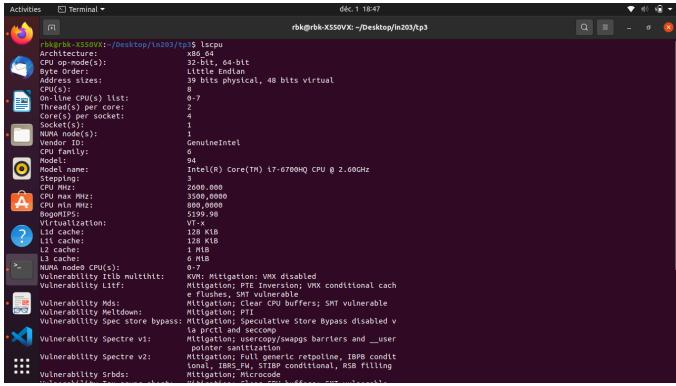
Commande Icspu:



En utilisant la commande Iscpu, mon pc a 8 coeur physiques et virtuelles, en fait, il s'agit de deux threads par coeur physique et la somme total des coeurs physiques est 4.

Le nombre de socket est le nombre d'emplacement de processeur du pc.

Exercice1:

Q1)

Code sur github

Q2)

Pour le temps de calcul du produit scalaire seulement on a:

*pour le code séquentiel:

T=0.78

En exécutant le code parallèle (OpenMP) avec différents nombre de thread on a le tableau suivant:

Nombre de threads	Speed up =ts/tp
2	0.78/0.38=2.05
4	0.78/0.2=3.9
6	0.78/0.184=4.23
8	0.78/0.180=4.33

Commentaire:

On remarque que si on augmente le nombre de thread le temps de calcul augmentre legerement de même pour le speed up et que le temps de calcul parallele est legrement plus petit que le temps de calcul du code séquentiel.

Q3) j'ai essayé d'éxécuter le fichier "dotproduct_thread.cpp" en modifiant le nombre de threads dans le code et j'ai obtenu le tableau suivant:

Nombre de threads	Speed up
4	0.78/1.49=0.52
8	0.78/1.29=0.4

Q4)

En fait, le temps de calcul en parallele avec OpenMP est très petit devant celui utilisé avec les threads.

Q5)

Tout en utilisant deux méthodes de prallélisation , le temps de calcul reste proche de celui du code séquentiel ce qui n'est pas souhaitable.

Donc ,il s'agit de 'memory bounding" c'est à dire le temps de calcul et d'éxécution est limité par le temps d'accès aux données de la mémoire. Puisque le nombre d'opérations est égale au nombre d'accés a la mémoire.

Exercice 2:

Q1) Ce qu'on peut remarquer dans cet exercice est que le nombre d'accés a la mémoire(3*blocksize^2) qui est inférieure au nombre d'operations(2*blocksize^3). Il s'agit de CPU bound.

dimension	temps	Mflops
1023	1.561	1371.27
1024	4.08	526.058
1025	1.562	1390

Commentaire:

On remarque le temps pour dim=1024 est plus log par rapport aux deux autres dimensions, on peut explique ça par le fait que le nombre de ligne=1024 est multiple de nombre de lignes caches c ad que les données seront stockés dans la même partie d'une ligne de cache. Ce qui n'est pas le cas pour dim=1023 et 1025.

D'ou ,l'execution dure plus pour dim=1024.

Q2)

Pour n=1024

Permutation des boucles	temps	Mflops
l,j,k	3.83	560.653
J,ik	6.30	340.5
I,k,j	17.73	121.079
K,I,j	18.32	117.211
J,k,i	0.95	2250.15
K,j,i	1.08	1981.55

COMMENTAIRE:

On remarque que le temps de calcul le optimale est réalisé lorsque i est au bas niveau ce qui est approuvé par le cours .

On peut expliquer cela par le fait que travailler en ligne en bas niveau les odnnes de chaque ligne de cache seront accessibles

Q3)

En utilisant la boucle j,k,l(dont le temps de calcul est le plus court) en parallélisant le code avec Open MP on obtient le tableau suivant :

Pour n=1024

Nombre de threads	Temps	Mflops 2250
Temps sequentiel	0.95	
2	0.477	4494.29
4	0.24	8927.25
8	0.20	10651.3

Q4)on peut améliorer notre code par passage au calcul par bloock de matrice qui nous permet une meilleure exploitation des caches .

Q5) code sur github

En testant le code pour differents valeurs de szblock(taille du block), on remarque que pour szblock=510, on atteint un maximum de MFlops qui est égale a peu près à 3000.91.

EN fait d'apres la fentre de la commande Iscpu, on remarque que la capacite de memoire cache L3 est 6MB==(a peu près)6100k, d'autre part pour les 3 matrices A, B et C on a, 510^2*3*8(sizeof double)=6200K qui peuevent alors etres stockes dans L3.

Q6)

Pour la boucle jki

dimension	temps	MFlops
1023	0.73	2923.5
1024	0.72	2948.44
1025	0.74	2845.64

Malgré l'augmentation de la taille (dimension) de la matrice il n'ya pas de grand changement concernant le temps et MFlops ce qui se differe de la premiere question.

Q7) code sur github

Pour la dim 1024

Pour le code sequentiel:

1024	0.72	2948.44

j'ai essayer differents nombre de threads sur le code par block

Nombre de thread (open MP)	temps	MFlops
2	0.68	3138
4	0.367	5847
8	0.368	5832

On remarque que le speed up est fixe à 2 pour nombre de thread >=4 car pour un szblock de 500 on a que 2 block a traiter pour la boucle parallelisée.