Bramas
Timothé

Compte-rendu TP1 OS202

Exercice 1 :

1) Tous les chiffres donnés sont obtenus en faisant la moyenne de 5 exécutions du produit matriciel .

Dimension	Durée d'exécution (s)	Opérations par seconde (Mflops)
1023	1,93	1109
1024	24,13	89
1025	2,16	1008

On voit que le temps de calcul est bien plus élevé pour une dimension de 1024.C'est le nombre d'opération diminue drastiquement pour 1024, cela indique donc que le problème se situe au niveau des accès mémoire et non du nombre d'instructions.
La cause de ce problème est l'associative memory cache : chaque portion de la mémoire est mappée sur une zone particulière du cache. Si la dimension de la matrice est un multiple de la taille des portions, chaque ligne sera mappée sur le même cache, cela reviendra à une utilisation virtuelle d'un cache beaucoup plus petit.
Pour résoudre ce problème on pourrait faire du padding: faire le calcul avec une matrice 25*25 complétée par une ligne et une colonne de zéros que l'on n'utilisera pas.

2) Le stockage de la matrice (ligne par ligne ou colonne par colonne) joue beaucoup
sur l'efficacité du parcours (pour pouvoir le plus possible parcourir la mémoire de manière continue) Si la matrice est définie ligne par ligne, il faudra la parcourir comme cela.
En regardant l'opérateur d'accès, on voit que l'élément A(i,j) se trouve en A.data[i+j*nRow]. Il n'y a pas de saut d'indice en incrémentant i : le stockage est fait colonne par colonne. Il faudra donc que la plus petite boucle parcoure des lignes pour réduire le nombre de sauts de mémoire effectués.

Il faut donc parcourir par i en dernier, comme c'est la variable qui parcours les lignes. On effectue donc par exemple les boucles dans cet ordre : k, j, puis i.

```
for (k = iColBlkA; k < std::min(A.nbCols, iColBlkA + szBlock); k++)
  for (j = iColBlkB; j < std::min(B.nbCols, iColBlkB + szBlock); j++)
    for (i = iRowBlkA; i < std::min(A.nbRows, iRowBlkA + szBlock); ++i)
      C(i, j) += A(i, k) * B(k, j);
```

Figure 1 : boucles parcourues dans l'ordre le plus rapide

On passe de 24 secondes pour 1024 à 1.5 secondes. Pour 1023 ou 1025 pas de différence.

3) On change le produit matriciel en ajoutant :
`#pragma omp parallel for private(i,j,k) shared(A,B,C) num_threads(n)`
avant les boucles. Cela va permettre d'effectuer les boucles en les répartissant sur n threads, en cherchant le plus petit n qui donne la performance maximale.
Je fais le test avec les boucles dans l'ordre le plus naif des boucles (i,j,k) pour observer une plus grande différence avec cette optimisation.

Nombre de threads	1	2	3	4	5	6	7	8
Temps d'exécution (s)	24	9	6	6	6	6	6	6

Note : en inversant l'ordre des boucles pour faire le parcours de manière efficace le résultat du calcul devient faux, je n'ai pas réussi à comprendre cet effet.

4) Ici les threads font plus ou moins la même chose, il semble plus intéressant de leur donner une mission chacun. Il semble aussi que l'on n'utilise que 3 threads pour la boucle sur les 8 disponibles sur ma machine.

5) En effectuant cette boucle :

```
Matrix operator*(const Matrix& A, const Matrix& B) {
  Matrix C(A.nbRows, B.nbCols, 0.0);
  int ii,jj,kk;
  int n=A.nbRows/szBlock;

  for (kk = 0; kk < n; kk+=szBlock)
    for (jj = 0; jj < n; jj+=szBlock)
      for (ii = 0; ii < n; ii+=szBlock)
        prodSubBlocks(ii,jj,kk,szBlock,A,B,C);
    //prodSubBlocks(0, 0, 0, std::max({A.nbRows, B.nbCols, A.nbCols}), A, B, C);
  return C;
}
```

Figure 2 : produit bloc par bloc

Ce code donne une erreur mathématique en l'exécutant, mais l'erreur est faible sur le coefficient qui pose problème. J'ai du coup enlevé la vérification pour avoir les temps de calcul.
Avec le bon ordre des boucles dans prodSubBlocks, en séquentiel :

Taille de bloc	2	4	8	16	64	128	256
Temps (s)	0,25	0,026	0,0055	0,0031	0,0033	0,0045	0,013

La taille optimale semble donc être 16.

6) Le calcul est bien plus efficace que celui scalaire (qui durait en séquentiel 1.5 secondes).

7) En parallélisant cela diminue légèrement le temps de calcul, et encore plus en parallélélisant la boucle des blocs. Cela ne semble pas cohérent, j'ai dû faire une erreur sur mon produit bloc par bloc.

8) Avec le test blas, on a un temps supérieur (0.045 secondes), la librairie utilisée par l'exemple dans test_product_matrice_blas.cpp ne doit pas être la librairie de calcul la plus performante, il y aurait par exemple openblas mais je n'ai pas réussi à l'utiliser dans le code.

EXERCICE 2 :

2.1 : Circulation d'un jeton dans un anneau

N'ayant pu utiliser que pyzo pour faire tourner mes scripts python, je n'ai pas réussi à exécuter plus d'un thread à la fois. J'ai donc écrit ce programme, sans toutefois pouvoir le tester :

```python
import mpi4py
import sys

globCom = MPI.COMM_WORLD.Dup()
nbp     = globCom.size
rank    = globCom.rank

if(rank==0):
    token=1

if(rank<nbp):
    globCom.send(rank,token,rank+1)
else :
    print(token)
```

Figure 3 : code de l'envoi de jeton

2.2 :

On peut séparer la création de x et celle de y dans deux threads différents, avec les premier qui envoie sont résultat au 2e qui fera les opérations suivantes, il n'y aurait donc besoin que de 2 threads, car si on en rajoute alors les threads suivants vont devoir attendre les résultats des précédents et resteront bloqués.

```python
# Calcul pi par une méthode stochastique (convergence très lente
import time
import numpy as np
import mpi4py



globCom = MPI.COMM_WORLD.Dup()
nbp     = globCom.size
rank    = globCom.rank


# Nombre d'échantillons :
nbSamples = 40000000

beg = time.time()
# Tirage des points (x,y) tirés dans un carré [-1;1] x [-1; 1]

if(rank==0) :
    x = 2.*np.random.random_sample((nbSamples,))-1.
    globCom.send(rank,x,1)

elif(rank==1):
    y = 2.*np.random.random_sample((nbSamples,))-1.

    # Création masque pour les points dans le cercle unité
    filtre = np.array(x*x+y*y<1.)

    # Compte le nombre de points dans le cercle unité
    sum = np.add.reduce(filtre, 0)

    approx_pi = 4.*sum/nbSamples
    end = time.time()
    print(f"Temps pour calculer pi : {end - beg} secondes")
    print(f"Pi vaut environ {approx_pi}")
```

Figure 4 : code du calcul de pi

Encore une fois, je n'ai pas pu exécuter mon code sur plusieurs threads donc je ne sais pas s'il est correct.