Méthodes Numériques - TP02 : Multiplications de Matrices

Abonnenc Alicia - Gilles Bonhoure

Mars 2016

1 Présentation du Sujet

Le but de ce TP était de réaliser divers calculs matriciels afin de définir lequel était le plus optimisé.

Nous avons donc réalisé trois types de multiplications de matrices différents ainsi que des opérations complémentaires :

1. Une multiplication calculant ligne par ligne la matrice résultat
2. Une multiplication calculant colonne par colonne la matrice résultat
3. Une multiplication calculant "bloc" par "bloc" la matrice résultat (on découpe la matrice résultat puis on calcule ligne par ligne ces sous-matrices)
4. Une addition de matrices
5. Une multiplication de matrice par un vecteur
6. Gaxpy

2 Utilisation

Nous avons fait en sorte que toutes les fonctionnalités soient modifiables directement dans le code. Il faut donc modifier les valeurs des define pour éditer :

- CACHE : OBLIGATOIRE pour bien calculer si les matrices dépassent du cache
- N : pour changer la taille des matrices
- THREADS : pour choisir le nombre de threads
- ITER : pour choisir le nombre d'itérations (pour faire durer le programme assez longtemps)
- BLOC : pour choisir la taille des blocs de la fonction de calculs par blocs (optimisation etc.)

Par la suite, on peut compiler facilement depuis le *Makefile* avec la commande *make* dans le terminal.

3 Résultats des tests

Voici quelques traces des résultats obtenus à partir de notre programme :

```
Calculs sur 5 matrices
Dimension des matrices : 500
Nombre de threads : 1
Dépassement du cache : non
// MULTIPLICATIONS //
Multiplication | Lignes de la matrice de sortie
time = 596.010ms
MFLOPS : 2097.280128
Multiplication | Colonnes de la matrice de sortie
time = 665.613ms
MFLOPS : 1877.968128
Multiplication | Par blocs de 32 valeurs de la matrice de sortie
time = 553.988ms
MFLOPS : 2256.366592
// SOMME //
time = 0.995ms
MFLOPS : 1256.281344
// MULT MATxVECT //
time = 1.186ms
MFLOPS : 1053.962944
// GAXPY //
time = 1.219ms
MFLOPS : 1027.481600

====================================

Calculs sur 5 matrices
Dimension des matrices : 500
Nombre de threads : 2
Dépassement du cache : non
// MULTIPLICATIONS //
Multiplication | Lignes de la matrice de sortie
time = 315.348ms
MFLOPS : 3963.874816
Multiplication | Colonnes de la matrice de sortie
time = 406.755ms
MFLOPS : 3073.103104
Multiplication | Par blocs de 32 valeurs de la matrice de sortie
time = 326.862ms
MFLOPS : 3824.243712
// SOMME //
time = 0.905ms
MFLOPS : 1381.215360
// MULT MATxVECT //
time = 1.214ms
MFLOPS : 1029.654016
// GAXPY //
time = 1.182ms
MFLOPS : 1059.644672

====================================
```

```
Calculs sur 5 matrices
Dimension des matrices : 500
Nombre de threads : 4
Dépassement du cache : non
// MULTIPLICATIONS //
Multiplication | Lignes de la matrice de sortie
time = 284.034ms
MFLOPS : 4400.881152
Multiplication | Colonnes de la matrice de sortie
time = 384.111ms
MFLOPS : 3254.267904
Multiplication | Par blocs de 32 valeurs de la matrice de sortie
time = 352.714ms
MFLOPS : 3543.947776
// SOMME //
time = 0.996ms
MFLOPS : 1255.020032
// MULT MATxVECT //
time = 1.413ms
MFLOPS : 884.642560
// GAXPY //
time = 1.172ms
MFLOPS : 1068.686080

==================================

Calculs sur 5 matrices
Dimension des matrices : 1024
Nombre de threads : 1
Dépassement du cache : oui
// MULTIPLICATIONS //
Multiplication | Lignes de la matrice de sortie
time = 35120.485ms
MFLOPS : 305.730944
Multiplication | Colonnes de la matrice de sortie
time = 36086.209ms
MFLOPS : 297.549088
Multiplication | Par blocs de 32 valeurs de la matrice de sortie
time = 31372.399ms
MFLOPS : 342.256832
// SOMME //
time = 8.305ms
MFLOPS : 631.292032
// MULT MATxVECT //
time = 5.201ms
MFLOPS : 1008.052352
// GAXPY //
time = 5.257ms
MFLOPS : 998.287936
```

```
====================================

Calculs sur 5 matrices
Dimension des matrices : 1024
Nombre de threads : 2
Dépassement du cache : oui
// MULTIPLICATIONS //
Multiplication | Lignes de la matrice de sortie
time = 26305.517ms
MFLOPS : 408.181248
Multiplication | Colonnes de la matrice de sortie
time = 32123.846ms
MFLOPS : 334.250720
Multiplication | Par blocs de 32 valeurs de la matrice de sortie
time = 24620.369ms
MFLOPS : 436.119328
// SOMME //
time = 8.170ms
MFLOPS : 641.723392
// MULT MATxVECT //
time = 5.160ms
MFLOPS : 1016.062016
// GAXPY //
time = 5.143ms
MFLOPS : 1020.416128

====================================

Calculs sur 5 matrices
Dimension des matrices : 1024
Nombre de threads : 4
Dépassement du cache : oui
// MULTIPLICATIONS //
Multiplication | Lignes de la matrice de sortie
time = 24254.872ms
MFLOPS : 442.691200
Multiplication | Colonnes de la matrice de sortie
time = 25243.920ms
MFLOPS : 425.346720
Multiplication | Par blocs de 32 valeurs de la matrice de sortie
time = 23320.224ms
MFLOPS : 460.433728
// SOMME //
time = 8.494ms
MFLOPS : 617.245120
// MULT MATxVECT //
time = 5.186ms
MFLOPS : 1010.968000
// GAXPY //
time = 5.258ms
MFLOPS : 998.098176
```

4 Analyse des résultats

On remarque que le nombre de MFlops est impacté par le nombre de threads utilisés. De plus, quand la taille des matrices dépasse le cache, la vitesse de calcul est divisée par 10 ! On voit donc l'intérêt d'utiliser des fonctions réduisant les tailles d'informations à utiliser, telle que la fonction de calcul par blocs, ainsi que les threads.