Compte rendu sur le produit matriciel et routine BLAS

Pather Stevenson et Benedictus-Kent Rachmat

Enseignant : Fortin Pierre

L3 ICHP - Semestre VI - 2022

Table des matières

Produit matriciel

Récupérer le code du produit matriciel séquentiel $C+ = A \times B$ (les 3 matrices A, B et C sont toutes considérées comme étant carrées, d'ordre N). L'exécution pour différentes valeurs de N pourra se faire avec le script *run.sh*.

1.1 Naïf

Performance du produit matriciel naïf pour $N = 512$:

exec time : 0.7598 s, 0.353297 Gflop/s

1.1.1 Options d'optimisations

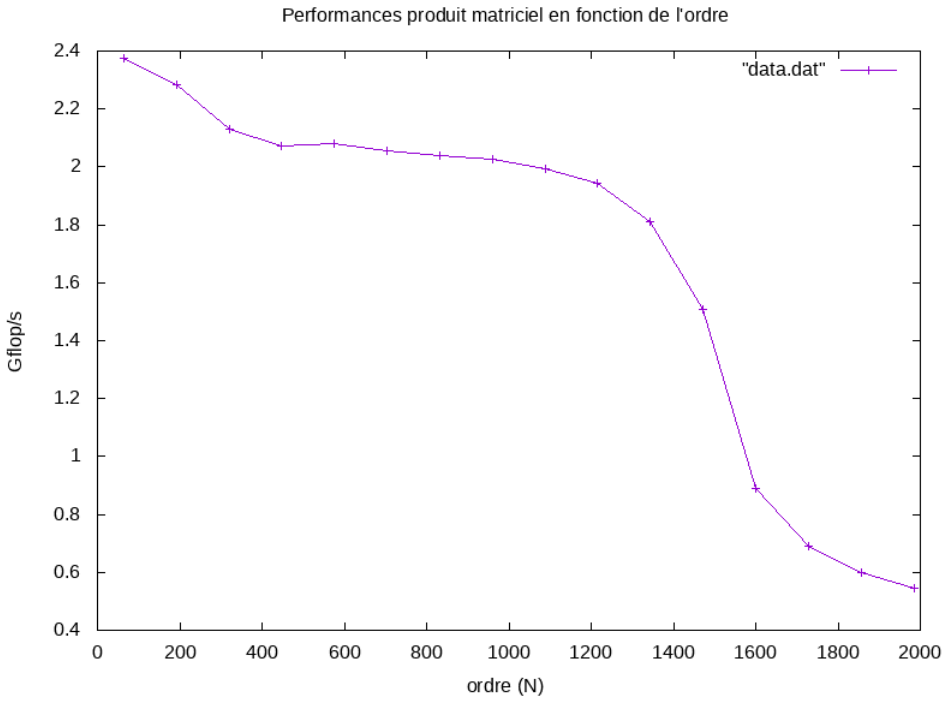
Voici les résultats des performances pour $N = 512$ en fonction des options de compilations en plus de **-march=native** et **-mtune=native** :

Ainsi on peut constater que l'option qui améliore le plus nos performances est l'option **-O3**. Nous allons donc utiliser ce choix d'options de compilation pour le lancement sur script **run.sh**.

	-O3	**-ffast-math**	**-funroll-loops**	**-fmodulo-sched**
Temps d'exécution (sec)	0.193224	0.759445	0.773548	0.760402
Gflop/s	1.38924	0.353463	0.347019	0.353018

1.1.2 Résultats du script

Voici les résultats que l'on obtient au lancement du script en utilisant notre programme matmul en ayant utilisé les options de compilation :

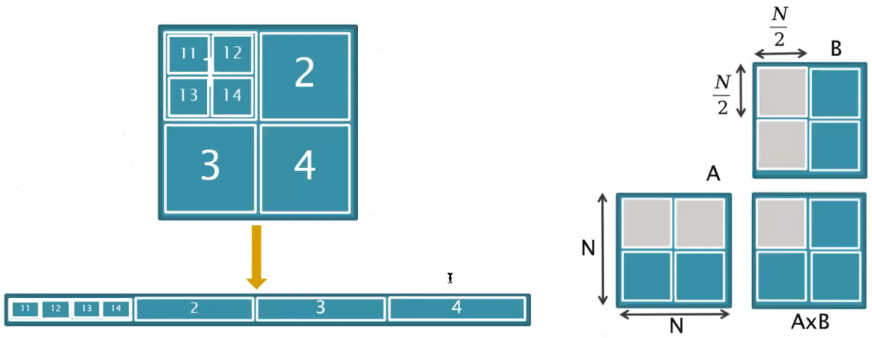


On peut constater une diminution d'environ 0.3 Gflop/s pour un ordre N compris entre 64 à 1216. Au délà de cet ordre on observe une chute critique du nombre de Gflop/s. Ainsi on observe globalement qu'au plus l'ordre augmente au plus les performances diminuent notamment pour $N > 1216$.

1.2 Récursif par bloc

Avec l'approche naïve faite avec trois boucles for, La localité pour les accès à la matrice A sont avantageuses mais ce n'est pas le cas pour la matrice B. Car chaque accès à un coefficient de la matrice B va charger une ligne de cache complète qui va contenir plusieurs coefficients alors qu'un seul sera utilisé.

Nous proposons donc l'approche récursive qui applique le principe diviser pour régner :



mm(int crow, int ccol, int arow, int acol, int brow, int bcol, int n, int stride, float *A, float *B, float *C) :

- Si $n < SEUIL$:

  - appel produit séquentiel pour les valeurs $n, crow, ccol, arow, acol, brow, bcol$

- Sinon :

  appel récursif avec :

- $crow, ccol, arow, acol, brow, bcol, n/2, stride, A, B, C$
- $crow, ccol, arow + n/2, acol, brow, bcol + n/2, n/2, stride, A, B, C$

- $crow+n/2, ccol, arow, acol, brow+n/2, bcol, n/2, stride+n/2, A, B, C$
- $crow+n/2, ccol, arow+n/2, acol, brow+n/2, bcol+n/2, n/2, stride+n/2, A, B, C$

- $crow, ccol + n/2, arow, acol + n/2, brow, bcol, n/2, stride, A, B, C$
- $crow, ccol + n/2, arow + n/2, acol + n/2, brow, bcol + n/2, n/2, stride$

- $crow+n/2, ccol+n/2, arow, acol+n/2, brow+n/2, bcol, n/2, stride+n/2, A, B, C$
- $crow + n/2, ccol + n/2, arow + n/2, acol + n/2, brow + n/2, bcol +$

$n/2, n/2, stride + n/2, A, B, C$.

1.2.1 Résultats

Nous obtenues les performances suivantes suivantes avec cette fonction récursive : **1.7496 Gflop/s**. Nous avions **1.25961 Gflop/s** pour la version naïve.

1.3 Parallélisation avec OpenMP

Il est possible de paralléliser ce produit matriciel à l'aide d'OpenMP et de ses directives : **#pragma omp task** et **#pragma omp taskwait**.

1.4 Calcul SIMD

BLAS