INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Optimisation et parallelisation OpenMP d'addition et produit de deux matrices denses

Rapport de Burreau d'étude

Luc-Christelle Nguyen et Alicia Perrin

Institut National des Sciences Appliquées de Toulouse

8 novembre 2025

Résumé

Dans ce rapport, nous présenterons différentes méthodes d'optimisation et de parallélisation appliquées aux calculs sur des matrices denses. Nous commencerons par étudier l'impact de l'ordre d'accès à la mémoire sur les performances, en exploitant la proximité spatiale des données afin d'améliorer l'utilisation du cache.Nous explorerons ensuite l'utilisation des bibliothèques OpenMP et OpenBLAS dans le but d'accélérer les calculs. L'analyse des trois niveaux de routines BLAS (BLAS1, BLAS2, BLAS3) mettra en évidence les gains considérables que l'on pourra obtenir grâce à OpenBLAS. Nous testerons également différentes stratégies de parallélisation (options static et dynamic) et nous étudierons l'impact du nombre de threads utilisés sur les performances globales. Enfin, nous mettrons en œuvre une optimisation basée sur la division en blocs (cache blocking) afin de mieux exploiter la hiérarchie mémoire.

Table des matières

Chapitre1

Modifier l'accès à la mémoire pour additionner deux matrices

La première optimisation dont nous nous sommes servis utilise la proximité spatiale des informations. Nous avons fait en sorte que, pour l'addition de deux matrices, l'algorithme parcourt d'abord les colonnes et ensuite les lignes. En effet, une matrice est stockée en mémoire en column-major. Lorsque la fonction va aller chercher la première valeur de la matrice, elle va remplir le cache avec les valeurs suivantes. Ainsi, grâce à une bonne utilisation du cache, un grand nombre d'accès mémoire sont évités, ce qui permet d'augmenter la performance du programme.

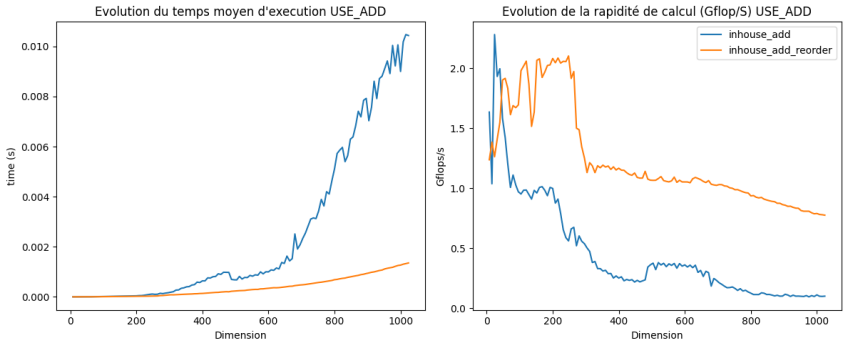

FIGURE 1.1 – Différences de performances en fonction de l'ordre d'accès à la mémoire

On peut remarquer sur ces simulations que la rapidité de calcul a doublé en moyenne (cf 1.1). C'est donc un critère important à prendre en compte lors de l'élaboration d'un programme.

Chapitre2

Compilation reliant les librairies OpenMP et BLAS

Il y a trois routines BLAS (Basic Linear Algebra Subprograms) différentes, qui sont des fonctions standards pour effectuer des calculs de base en algèbre linéaire :

- BLAS1 : ce sont des opérations vecteur-vecteur. Pour effectuer un produit matrice-matrice en utilisant uniquement des routines BLAS1, il faut appeler la routine pour chaque élément de la matrice résultat (car chaque élément est le produit scalaire d'une ligne et d'une colonne).
- BLAS2 : ce sont des opérations matrice-vecteur. Pour calculer un produit matrice-matrice avec des routines BLAS2, il faut appeler la routine pour chaque colonne de la matrice résultat.
- BLAS3 : ce sont des opérations matrice-matrice. Un seul appel de la routine permet de calculer tout le produit matriciel.

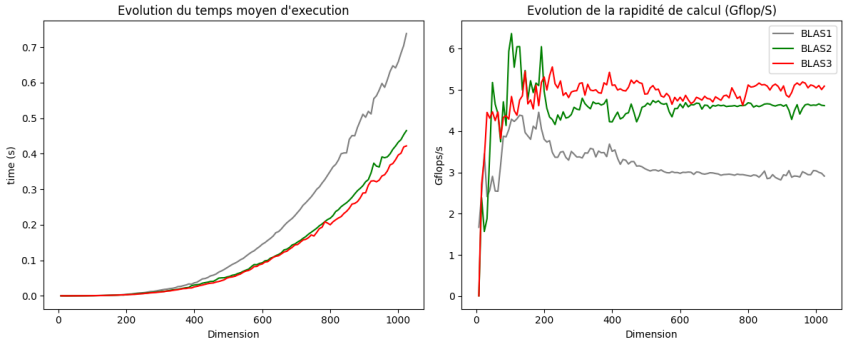

FIGURE 2.1 – Performances BLAS 1, 2, 3 sans optimisation Openblas

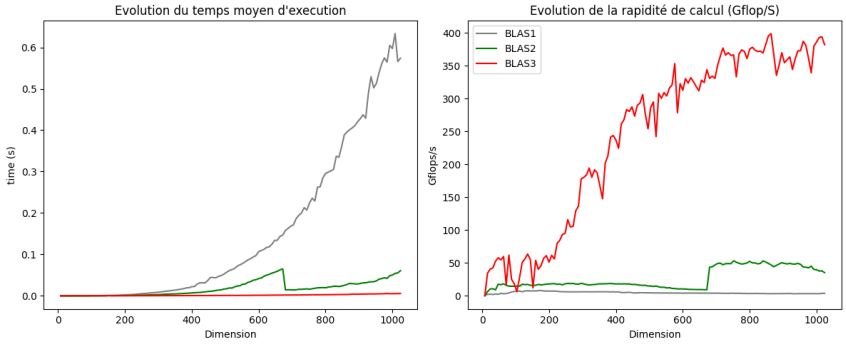

FIGURE 2.2 – Performances BLAS 1, 2, 3 avec optimisation Openblas

Lors de ces simulations, nous avons voulu premièrement montrer la différence d'efficacité entre les différents BLAS (cf. 2.1). On remarque que BLAS3 est légèrement meilleure que les autres, du fait qu'il ne fait appel qu'à une seule fonction. Puis nous avons voulu montrer la nette amélioration des performances lorsque nous utilisons la librairie OpenBLAS. C'est une version rapide et optimisée des routines BLAS. Elle utilise des optimisations spécifiques au processeur pour exploiter au mieux les instructions vectorielles et le calcul parallèle sur plusieurs cœurs. La différence de performance est énorme. Par exemple, pour BLAS3 la rapidité de calculs passe de 5Gflops à 400Gflops (cf 2.2).

Chapitre3

Options de parallelisation d'un produit matriciel

Dans cette partie, nous nous sommes concentrées sur l'optimisation par la parallélisation des tâches. Dans les simulations qui suivent, nous avons voulu tester les différentes options de parallélisation.

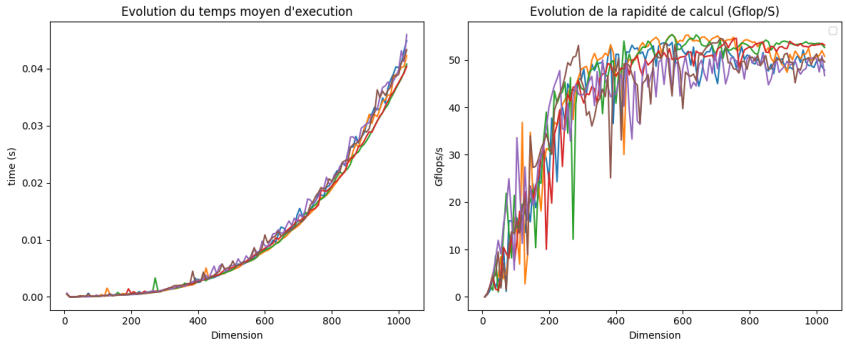

FIGURE 3.1 – Performances d'un calcul matriciel parallélisé avec l'option static et différents nombres de coeurs disponibles

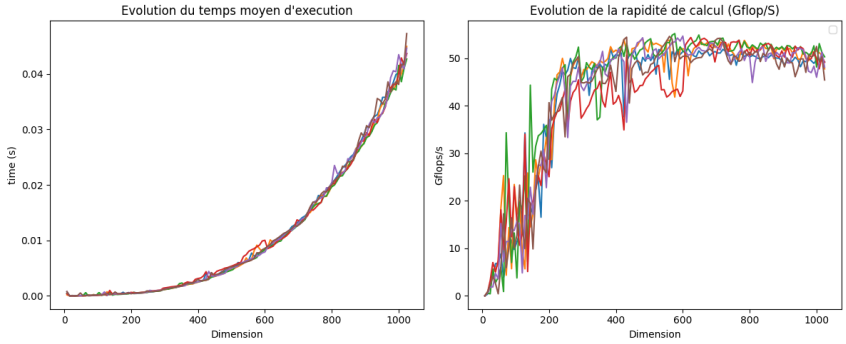

FIGURE 3.2 – Performances d'un calcul matriciel parallélisé avec l'option dynamic et différents nombres de coeurs disponibles

Nous pouvons remarquer que changer ces options n'a aucune influence réelle sur la performance du programme.

Chapitre4

Recherche du nombre de threads optimal pour BLAS3

Dans cette partie nous nous sommes intéressées à l'influence du nombre de threads utilisés sur la rapidiré de calcul et le temps d'éxecution de BLAS3.

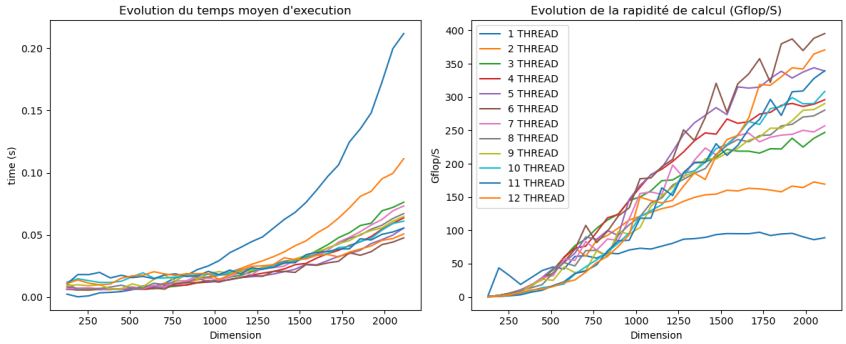

FIGURE 4.1 – Performance de BLAS3 en utilisant 1 à 12 threads lors de son éxecution

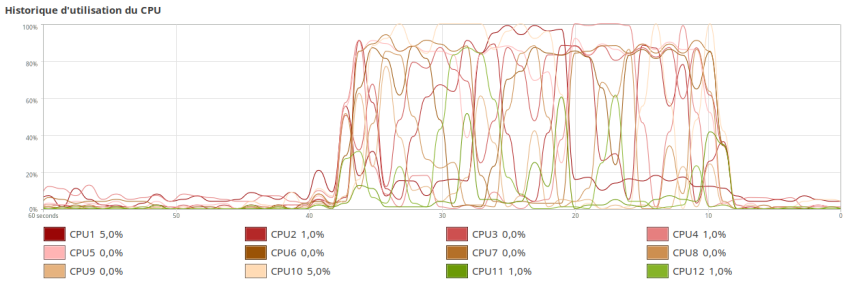

FIGURE 4.2 – Historique d'utilisation des threads pendant l'éxecution BLAS3

En faisant les simulations de BLAS3 avec 1 à 12 threads, nous observons que les performances de BLAS3 sont optimales lorsque le CPU utilise 6 threads sur les 12 disponibles (cf. 4.1). En effet, en introduisant plus de 6 threads, les ressources sont partagées entre davantage de threads, ce qui diminue les performances de BLAS3. De plus, en surveillant l'historique d'utilisation du CPU lors de l'exécution de BLAS3 avec 6 threads (cf. 4.2), nous remarquons que, bien que 6 threads soient utilisés en permanence, ce ne sont pas les mêmes qui sont actifs tout au long de l'exécution.

Chapitre5

La cache blocking pour optimiser la multiplication matrice-matrice

Dans le cadre de cette approche, nous cherchons à utiliser la mémoire cache de manière optimale en apportant des modifications à notre algorithme de multiplication matrice-matrice pour appliquer le principe de localité. Nous avons ainsi divisé les matrices en sous-blocs de taille adaptée au cache, permettant de maximiser la réutilisation des données dans les caches.

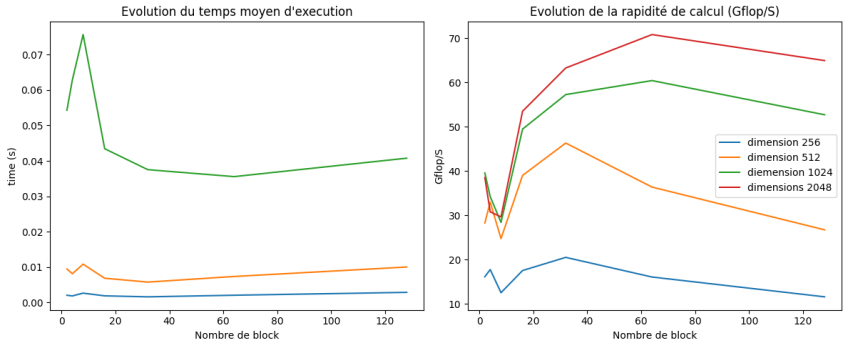

FIGURE 5.1 – Performance de calculs en fonction de la taille de block utilisée pour différentes tailles de matrices

FIGURE 5.2 – Performance de la méthode cache blocking, de BLAS3 et de la méthode de parallélisation

Nous nous sommes dans un premier temps penchées sur la taille de bloc optimale pour notre cache. En utilisant la méthode de blocking, nous cherchons à étudier les performances des calculs en fonction de la taille du bloc pour différentes tailles de matrices. Nous observons une taille de bloc optimal de 32 pour les petites dimensions (256 et 512) et de 64 pour les grandes dimensions. Nous avons choisi de continuer nos simulations avec un bloc de taille 32 qui nous paraît un meilleur compromis entre les petites et grandes dimensions. En effet, avec un bloc de taille 64, on observe une chute importante de la performance pour les matrices de taille 512. (cf. 5.1).

Enfin, nous avons cherché à comparer les performances de cette méthode à celles de la parallélisation et de BLAS3 de la librairie OpenBLAS. On observe un seuil de taille de matrice à partir duquel les performances de BLAS3 sont bien meilleures que celles de la méthode de parallélisation ou de cache-blocking. Ce seuil se situe entre 750 et 1000 (cf. 5.2). En dessous du seuil, c'est la méthode de cache

blocking qui est plus performante, suivie de la parallélisation. À l'inverse. Les simulations démontrent la supériorité nette des méthodes de la librairie OpenBLAS pour optimiser les performances des calculs sur de grandes dimensions.