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

6 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

1 Modifier l'accès à la mémoire pour additionner deux matrices 2

2 Compilation reliant les librairies OpenMP et BLAS 3

3 Options de parallelisation d'un produit matriciel 4

4 Nombre de threads utilisé pour BLAS3 5

5 Utiliser les blocs du cache 6

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 de 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.

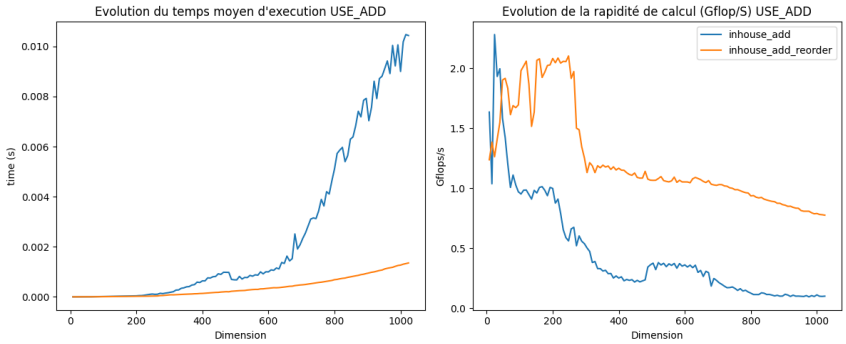

FIGURE 1.1 – Différences de performances en fonction de l'ordre d'accès à la mémoire

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.

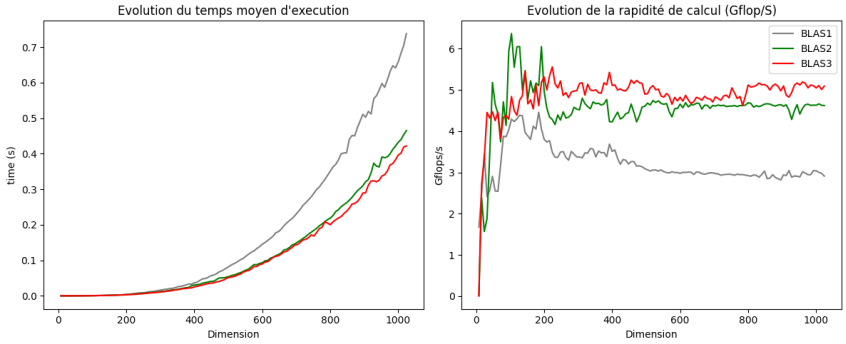

FIGURE 2.1 – Performances BLAS 1, 2, 3 sans optimisation Openblas

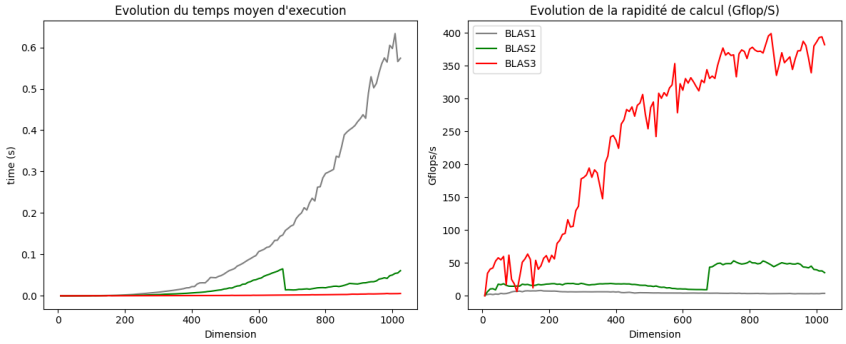

FIGURE 2.2 – Performances BLAS 1, 2, 3 avec optimisation Openblas

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.

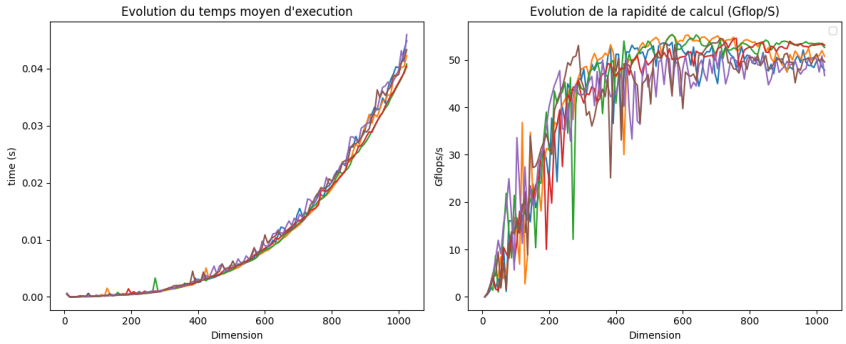

FIGURE 3.1 – Performances d'un calcul matriciel parallélisé avec l'option static et différents nombres de coeurs disponibles

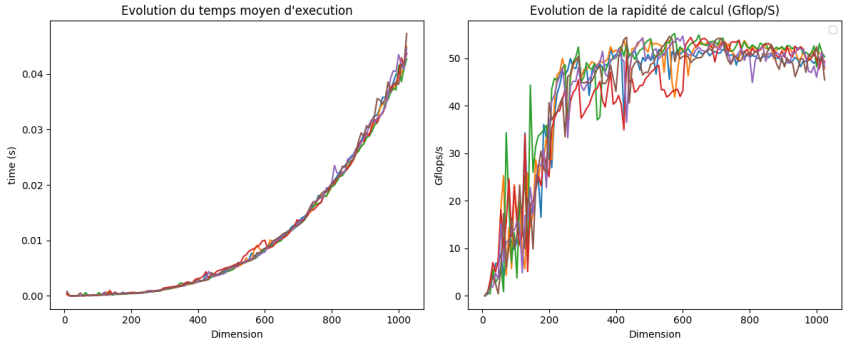

FIGURE 3.2 – Performances d'un calcul matriciel parallélisé avec l'option dynamic et différents nombres de coeurs disponibles

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

Chapitre4

Nombre de threads utilisé pour BLAS3

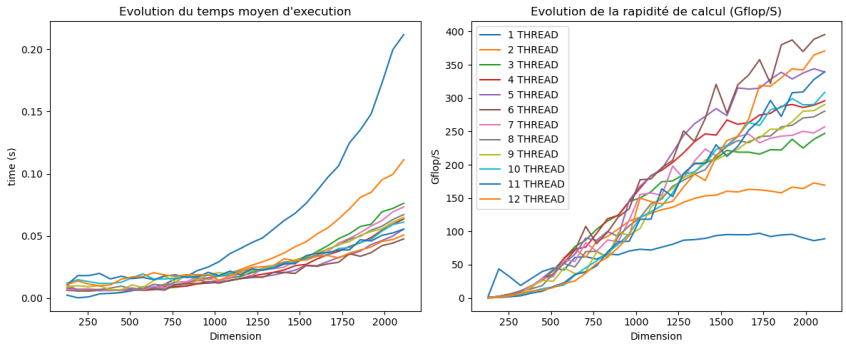

FIGURE 4.1 – Trouver titre

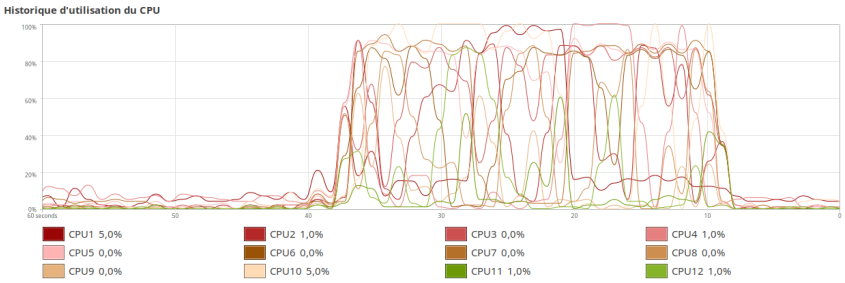

FIGURE 4.2 – Trouver titre

Fait des tests de 1 à 12. Le nombre optimal c'est 6 parce que plus on parrallelise, plus on doit partager des données entre les différents threads.

Expliquer que 6 threads qui ravail en permanence mais ce ne sont pas tout le temps les même.

Chapitre5

Utiliser les blocs du cache

Expliquer la technique du block.

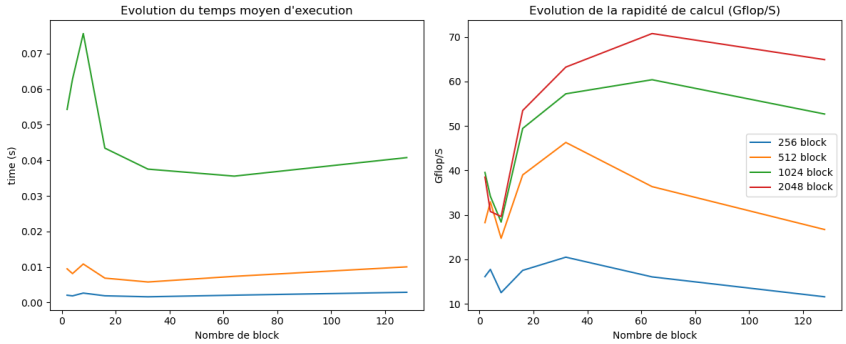

FIGURE 5.1 – Trouver titre

FIGURE 5.2 – Trouver titre

On a commencé par une rechercher de la taille de block optimale en fonction de différentes tailles de matrices. On a choisi 32 (mieux que 64).

Puis on a comparé cette technique avec openblas et la parralelisation.

On peut observer que la librairie openblas est la meilleure solution pour optimiser les performances de calcul en partageant efficassemet les thread.

Bibliographie