TP Produit de Matrice, Cache Blocking

Estelle Oliveira

Avril 2025

0.0.1 Github

Lien github vers le code "matrix-product" : https://github.com/estelleoliveira/matrix-product

1 Introduction

L'objectif de ce TP est de mettre en évidence les effets du cache blocking ainsi que des layouts sur le produit de matrice. Nous réaliserons des mesures de profilages pour l'implémentation originale, naïve, du code ainsi que pour l'implémentation comprenant les optimisations Layout, cache blocking et taille de blocs/matrices.

2 Sans Optimisation

Dans un premier temps, nous nous intéresserons au temps d'exécution et à la performance du programme pour différentes tailles de matrices et différents nombres de threads. Nous pourrons également mettre en valeur les caches misses dans les caches L1 et L3.

On réalise plusieurs mesures de performance, afin de mesurer dans un premier temps, le temps d'exécution pour différentes tailles de matrices :

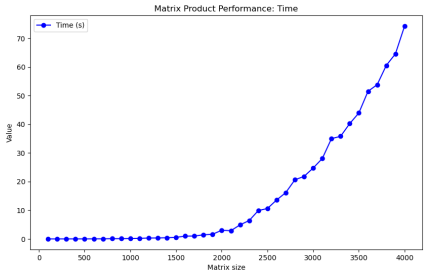

Figure 1: Temps d'exécution avec 6 threads pour des tailles différentes de matrices (Lower is better)

On remarque une perte importante de performance pour des tailles de matrices dépassant 2000×2000. Cela s'explique par le fait que les matrices ne tiennent plus dans les caches du processeur, entraînant une augmentation des accès à la mémoire principale (RAM) et une baisse des performances de calculs.

Ensuite, on fait une mesure de strong scaling qui nous permet d'obtenir les graphiques suivants :

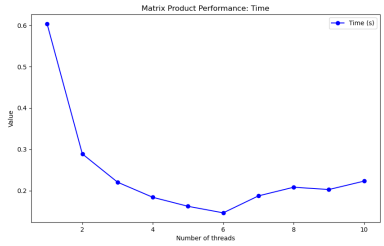

Figure 2: Temps d'exécution avec des matrices 1000x1000 pour différents nombre de threads (Lower is better)

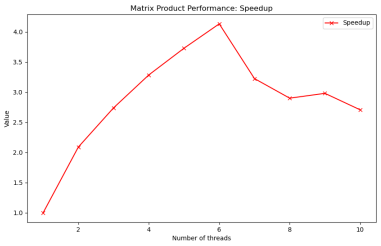

Figure 3: Speedup observé pour un différent nombre de threads, matrices 1000x1000 (Higher is better)

On remarque très rapidement que ce soit pour le Speedup ou pour le temps d'exécution, que les meilleurs performances sont pour un nombre de threads égal à 6, ce qui s'explique par l'architecture de l'ordinateur qui a fait les mesures. Il dispose de 10 coeurs physiques mais 6 coeurs sont des P-cores (Performance Cores qui permettent d'atteindre 4GHz) et 4 sont des E-cores (Efficiency cores qui permettent d'atteindre 3.5GHz).

On pourra donc en conclure que les meilleurs performances pour le programme sont pour 6 threads et pour des tailles de matrices allant jusqu'à 2000x2000.

C'est pour cela qu'on réalisera les mesures de cache misses de référence pour 6 threads ainsi que pour des tailles de matrices 2000x2000. Le tableau suivant nous indique le nombre de cache misses obtenu avec perf :

Table 1: Résultats de profilage avec `perf` pour une multiplication matricielle de taille 2000×2000

Événement	**Valeur**	**%**
Temps produit de matrice (s)	1.47 s	
Performance (GFLOP/s)	10.86	
cpu-core/L1-dcache-load-misses/	32 647 709	
cpu-atom/LLC-load-misses/	760 981	76.08%
cpu-core/LLC-load-misses/	329 780	99.97%
cpu-atom/cache-references/	400 285 548	76.08%
cpu-core/cache-references/	706 562 472	99.97%
cpu-atom/cache-misses/	154 767 318	38.66% of all cache refs
cpu-core/cache-misses/	86 349 266	22.32% of all cache refs

3 Layouts

Le layout permet de définir le mode de stockage de nos matrices, en col_major ou bien en row_major. Le col_major sera défini ici par LayoutLeft et le row_major par LayoutRight. Ces différents mode de stockage permettent de définir si les valeurs des colonnes (pour col_major) ou bien des lignes (pour row_major) seront contigus en mémoire. C'est donc une composante importante pour améliorer les défauts de caches et le temps d'exécution et donc le speedup du programme. On trace les défaults de cache en fonction du Layout pour chacune des matrices A, B, C. Ainsi, on fait un profilage pour toutes les combinaisons possibles de Layouts :

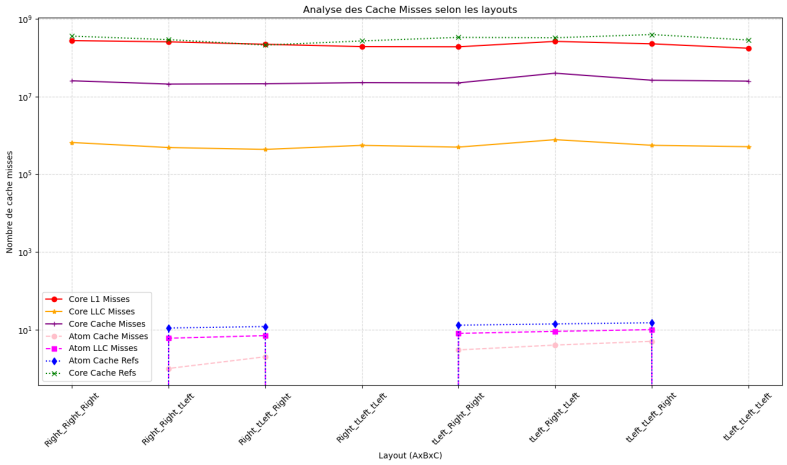

Figure 4: Caches misses observés pour différentes layout de matrices - Echelle logarithmique - (Lower is better)

On observe, d'une part au niveau du graphique mais aussi des données obtenues que la combinaison de Layouts qui permet d'obtenir le moins de défauts de cache est le Layout Right_Left_Right. Ce qui correspond à l'intuition théorique de la multiplication de matrices et de son parcours d'indices. En effet, lors de la multiplication de matrices, on va balayer les valeurs de A en lignes lorsqu'on balaye les valeurs de B en colonnes, ceci entraînant une meilleure gestion des caches pour un Layout Right, Layout Left et donc Layout Right, puisque les données sont généralement plus optimisées pour être écrites en lignes qu'en colonnes en C. Pour la suite des mesures de performances, nous resterons en Layout Right_Left_Right, afin d'obtenir la version la plus optimisée possible au niveau des défauts de cache.

On s'intéresse également au temps d'exécution du produit matriciel, et l'influence des Layouts sur celui-ci :

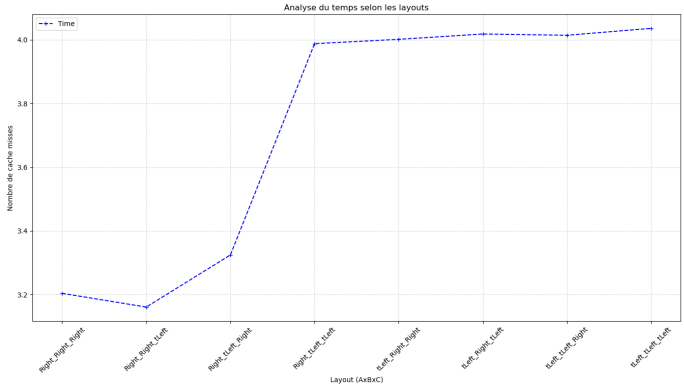

Figure 5: Temps observés pour différentes layout de matrices (Lower is better)

Les meilleures configurations en temps sont les configurations Layout Right_Right_Right, Right_Right_Left et Right_Left_Right, notamment dû à la méthode de stockage en C et C++, qui sont en row_major, donc en LayoutRight. Le Layout Right_Left_Right nous permet donc de diminuer drastiquement les défauts de cache tout en nous permettant de ne pas avoir trop de temps d'exécution.

4 Cache Blocking

Le cache Blocking est une technique consistant à exécuter les opérations matricielles par blocs pour éviter de rechercher des lignes de cache en mémoire. Au lieu de faire les opérations sur toute une ligne de matrice ou toute une colonne, on va découper la matrice en plusieurs blocs de matrices afin de bénéficier de la localité spatiale et temporelle. Ceci nous permet d'avoir moins de défauts de cache et de meilleurs performances.

On teste d'abord le cache blocking pour des tailles différentes de blocs, pour déterminer le temps d'exécution le plus rapide :

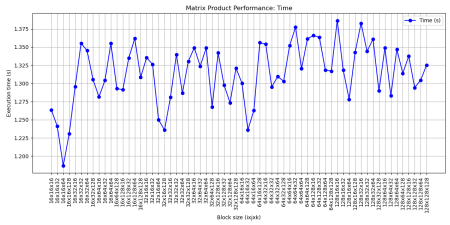

Figure 6: Temps d'exécution observé pour une taille de blocs variables, matrices de taille 2000x2000 et 6 threads (Lower is better)

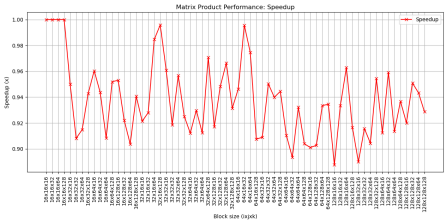

Figure 7: Speedup observé pour une taille de blocs variables, matrices de taille 2000x2000 et 6 threads (Higher is better)

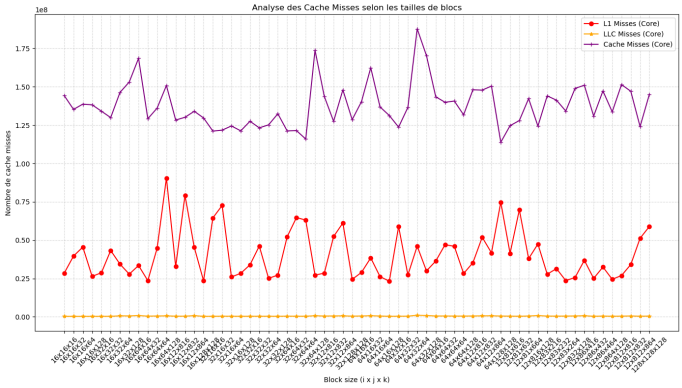

Figure 8: Caches misses observés pour différentes tailles de blocs (Lower is better)

On choisit un bloc de sorte à avoir un bon temps d'exécution avec le moins de défauts de cache possible. On sélectionne le bloc 64x16x32 qui permet d'avoir un très bon speedup et temps d'exécution, en éliminant beaucoup de défauts de cache.

Comparatif des résultats (bloc utilisé 64x16x32) :

Table 2: Comparaison des performances entre la version originale et la version cache-blocked, bloc de 64x32x64 - Layout Right Left Left

Événement	**Originale**	**Cache-blocked**	**Evolution**
Temps produit de matrice (s)	1.47	1.11	×1.32 plus rapide
Performance (GFLOP/s)	10.86	14.45	+33.1%
L1-dcache-load-misses (core)	32 647 709	34 927 249	+7.0%
LLC-load-misses (atom)	760 981	7 431	−99.0%
LLC-load-misses (core)	329 780	154 110	−53.3%
Cache references (atom)	400 285 548	30 333 778	−92.4%
Cache references (core)	706 562 472	57 148 708	−91.9%
Cache misses (atom)	154 767 318	3 948 946	−97.4%
Cache misses (core)	86 349 266	6 503 073	−92.5%

Les performances pour la version avec cache blocking et optimisations sont bien supérieures, avec un temps d'exécution plus faible (1.32x plus rapide). On remarque que presque tous les défauts de cache ont diminués et pour la plupart de 90% ou plus. Le nombre d'accès mémoire (cache references) a lui aussi énormément diminué, on observe sur la version optimisée une baisse de plus de 90% d'accès mémoire. C'est également parce qu'on fait beaucoup moins d'accès mémoire que les performances sont améliorées. La localité spatiale et temporelle a été améliorée, nous permettant de calculer plus vite et plus efficacement.

Enfin, une dernière étude de strong scaling nous permet de visualiser l'amélioration obtenue :

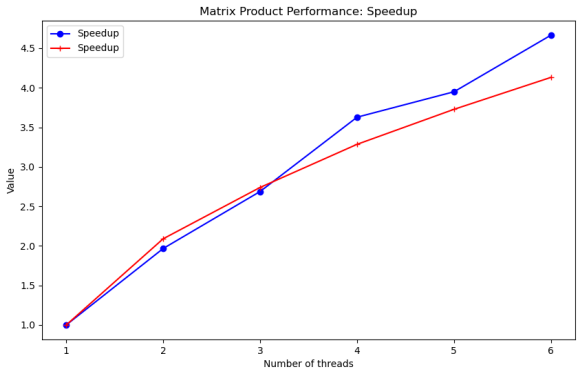

Figure 9: Etude de strong scaling version cache blocking et naïve (Higher is better)

5 Conclusion

Nous avons analysé les différentes optimisations que nous pouvons implémenter pour limiter les défauts de cache lors de l'exécution d'un programme tel que le produit de deux matrices.
Le Layout et l'architecture machine ont des effets importants sur les défauts de cache et doivent être pris en considération pour limiter les défauts de cache.
Le cache blocking permet de conserver les lignes de cache plus longtemps afin d'utiliser la localité spatiale et temporelle, et de bénéficier de celles-ci, au profit d'une exécution plus rapide, performante et optimisée. Enfin, le cache blocking a un effet notable sur les défauts de cache, les réduisant énormément, jusqu'à 90% et plus pour la plupart.