OS202 - Programming Parallel Computers

TP1

2 février 2026

NOUKOUA TATMFO Maëva Sandy

2025–2026

Table des matières

1 Produit matrice-matrice

Question 1

Les résultats suivants ont été obtenus :

n	Temps (s)	MFLOPS
1023	1.1473	1866.3
1024	2.8256	760.1
1025	1.22519	1757.91
Moyenne		

TABLE 1 – Temps d'exécution et performances du produit matrice–matrice (ordre *ijk*)

On observe une différence significative de temps d'exécution lorsque la taille des matrices est une puissance de 2 (1024). Ce phénomène s'explique par le fonctionnement du cache du processeur.

En effet, pour ces tailles particulières, les accès mémoire suivent des motifs réguliers qui provoquent des conflits de cache, entraînant un remplacement fréquent des données dans le cache et un recours accru à la mémoire principale.

À l'inverse, des tailles non puissances de 2 comme 1023 ou 1025 perturbent cet alignement et permettent une meilleure utilisation du cache, ce qui améliore les performances.

Question 2

Les programmes ont été compilés à l'aide du compilateur `g++` avec des options d'optimisation activées afin de se rapprocher de conditions d'exécution réalistes. La compilation a été réalisée via le `Makefile` fourni, qui utilise notamment les options `-O2` et `-march=native`. Ces options permettent au compilateur d'optimiser le code et d'exploiter les caractéristiques du processeur utilisé.

Ordre des boucles	1023	1024	1025
i, j, k	1.188	2.41204	1.13611
j, i, k	1.64774	2.99148	1.54089
i, k, j	4.35307	6.0035	4.75402
k, i, j	2.93009	5.816	3.13717
j, k, i	0.648505	0.58561	0.580549
k, j, i	0.660067	0.659519	0.624007

TABLE 2 – Temps d'exécution (en secondes) pour les différentes permutations des boucles

Les résultats montrent que le simple changement de l'ordre des boucles permet déjà d'obtenir un gain de performance très significatif, sans modifier ni le processeur ni le nombre de cœurs utilisés. En particulier, les ordres de boucles *jki* et *kji* apparaissent comme les plus efficaces. Ce gain s'explique par une meilleure exploitation de la hiérarchie mémoire, en réduisant les accès non contigus et les conflits de cache. Ainsi, avant même toute parallélisation, une optimisation de l'accès mémoire permet d'améliorer les performances d'un facteur proche de dix.

Question 3

Le produit matrice–matrice a été parallélisé à l'aide d'OpenMP en utilisant la permutation de boucles la plus performante identifiée précédemment. Les performances ont été mesurées en

faisant varier le nombre de threads via la variable d'environnement `OMP_NUM_THREADS`, pour différentes tailles de matrices.

OMP_NUM_THREADS	**Temps** ($n = 512$)	**Temps** ($n = 1024$)	**Temps** ($n = 2048$)
1	0.0656129	0.560631	5.16348
2	0.0613203	0.535099	5.15953
4	0.060877	0.528264	5.42387
8	0.0607436	0.552694	5.43888
12	0.0604554	0.546076	5.32373
16	0.0604496	0.547351	5.29456

Table 3 – Temps d'exécution OpenMP en fonction du nombre de threads

OMP_NUM_THREADS	**Speedup** ($n = 512$)	**Speedup** ($n = 1024$)	**Speedup** ($n = 2048$)
1	1.000	1.000	1.000
2	1.070	1.048	1.001
4	1.078	1.061	0.951
8	1.081	1.014	0.949
12	1.086	1.027	0.970
16	1.086	1.024	0.975

Table 4 – Accélération (speedup) OpenMP en fonction du nombre de threads

Les courbes de speedup montrent un gain limité avec OpenMP : l'accélération reste proche de 1, et peut même devenir inférieure à 1 pour $n = 2048$. Cela s'explique par le coût de parallélisation (création/synchronisation des threads) et surtout par une limitation mémoire : au-delà de quelques threads, la bande passante mémoire et les effets de cache deviennent le facteur dominant, ce qui empêche une accélération linéaire. Ces résultats confirment qu'optimiser les accès mémoire (ordre des boucles, blocage) est une étape essentielle avant d'espérer un gain significatif avec le parallélisme.

Question 4

Bien que la parallélisation OpenMP apporte un léger gain, les performances restent limitées par les accès mémoire et la saturation de la bande passante. Des optimisations supplémentaires, telles que le blocage des matrices ou l'utilisation de bibliothèques optimisées comme BLAS, permettraient d'améliorer significativement les résultats. Les performances obtenues reflètent donc les limites d'une implémentation naïve plutôt qu'une limite matérielle.

Question 5

Le produit matrice–matrice par blocs (tiling) vise à améliorer la localité mémoire en réutilisant des sous-blocs de A et B plusieurs fois alors qu'ils résident encore dans le cache. Contrairement à la version « scalaire », où l'on parcourt de grandes zones mémoire avec de nombreux accès non contigus, le blocage réduit fortement les défauts de cache (cache misses) et le trafic vers la mémoire principale. En conséquence, le temps d'exécution diminue et les performances augmentent, en particulier pour les grandes tailles de matrices.

La taille de bloc influence directement les performances : un bloc trop petit entraîne un surcoût de boucles et une réutilisation insuffisante des données, tandis qu'un bloc trop grand ne tient plus correctement dans le cache (L1/L2), ce qui annule le bénéfice. On observe donc généralement une taille de bloc optimale, correspondant à un bon compromis entre réutilisation et capacité de cache.

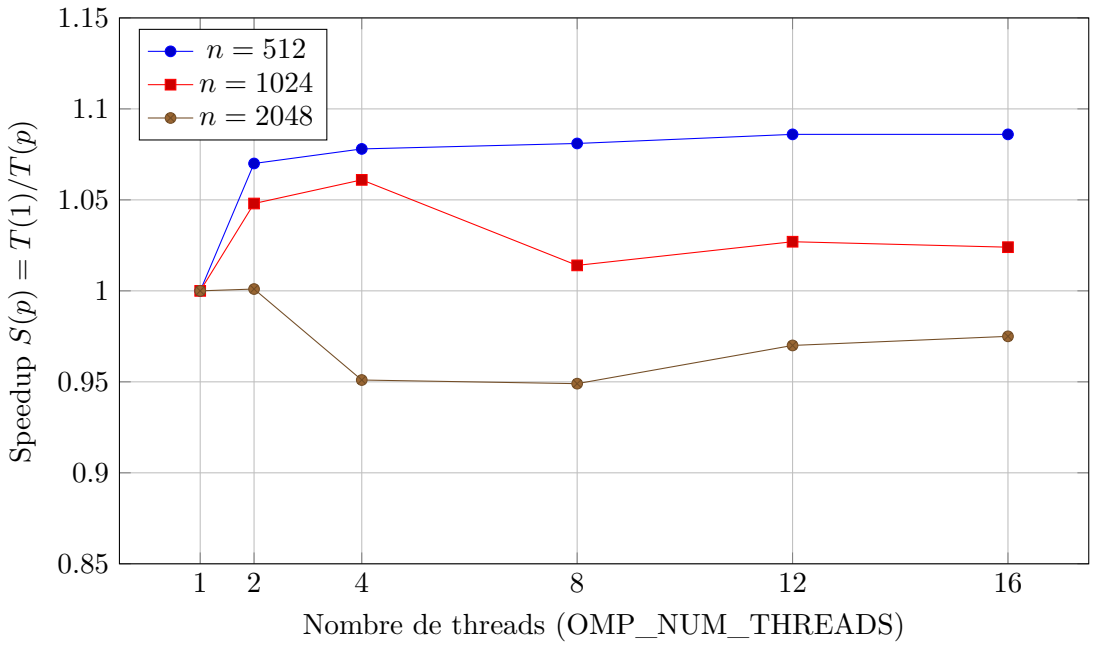

FIGURE 1 – Courbes de speedup OpenMP pour différentes tailles de matrices.

Configuration (ordre, taille bloc)	**Temps (n=1024) [s]**	**Temps (n=2048) [s]**
`mnjki` (bloc=16)	0.842497	7.283589
`mnjki` (bloc=64)	0.655394	5.342796
`mnjki` (bloc=256)	0.552740	4.578990
`nmjki` (bloc=16)	0.812334	7.745882
`nmjki` (bloc=64)	0.656772	5.463736
`nmjki` (bloc=256)	0.550865	4.661620

TABLE 5 – Produit matrice–matrice par blocs : temps d'exécution pour différentes configurations

Question 6

La comparaison avec le produit matrice–matrice « scalaire » montre que le blocage permet d'obtenir un temps d'exécution inférieur à celui de la meilleure version séquentielle non bloquée. Cela s'interprète par le fait que le calcul devient moins limité par la mémoire : en réutilisant efficacement les données dans le cache, on réduit les accès à la RAM, qui constituent souvent le principal goulot d'étranglement. Ainsi, le gain provient avant tout d'une meilleure exploitation de la hiérarchie mémoire, et pas d'une réduction du nombre d'opérations (qui reste $\mathcal{O}(n^3)$).

Dans cette implémentation, l'utilisation de blocs est plus simple lorsque n est multiple de la taille de bloc, afin d'éviter la gestion de blocs partiels aux bords. C'est pourquoi les tests se concentrent souvent sur des tailles telles que 1024 et 2048, qui sont facilement divisibles par de nombreuses tailles de blocs.

En conclusion, le blocage améliore significativement les performances du produit matrice–matrice en optimisant l'accès mémoire et l'utilisation du cache, et l'existence d'une taille de bloc optimale confirme l'importance de la hiérarchie mémoire dans ce type de calcul.

Question 7

L'accélération parallèle est calculée à partir du rapport $S(p) = T(1)/T(p)$, où $T(1)$ correspond au temps séquentiel par blocs et $T(p)$ au temps parallèle obtenu avec OpenMP. Pour la configuration la plus performante (*nmjki*, bloc=128), on obtient une accélération d'environ 6

pour $n = 1024$ et d'environ 11 pour $n = 2048$ avec 16 threads.

Ces accélérations sont nettement supérieures à celles observées pour la version scalaire parallélisée. Cela s'explique par le fait que le blocage améliore fortement la localité mémoire, ce qui réduit les accès à la mémoire principale et permet aux threads OpenMP d'exploiter plus efficacement le parallélisme. Sans blocage, le calcul reste limité par la mémoire, ce qui empêche toute accélération significative.

Question 8

Taille n	**Temps BLAS (s)**	**Temps bloc+OpenMP (s)**	**Rapport BLAS / bloc+OMP**
1024	0.514358	0.134669	3.82
2048	5.07078	0.674998	7.51

TABLE 6 – Comparaison des temps d'exécution avec BLAS

La comparaison avec l'exécutable utilisant BLAS montre que, pour les tailles testées, la version par blocs parallélisée avec OpenMP est plus rapide que BLAS. Le rapport de temps est d'environ 3.8 pour $n = 1024$ et 7.5 pour $n = 2048$.

Ce résultat s'explique par le fait que la bibliothèque BLAS utilisée n'exploite pas nécessairement tout le parallélisme disponible sur la machine, tandis que la version OpenMP est explicitement parallélisée avec un nombre de threads adapté. Néanmoins, dans un contexte industriel, une BLAS hautement optimisée (par exemple OpenBLAS ou MKL) reste généralement la solution la plus robuste et portable.

2 Parallélisation MPI