ES201 - Rapport de projet de TP5

par

Bastien Hubert - Liam Kelley - Sylvain Largent - Thomas Raynaud

Professeur encadrant :
Nicolas Gac

Mars 2022

Sommaire

1 Introduction

2 Partie 1 : Analyse théorique de cohérence de cache

2.1 Question 1

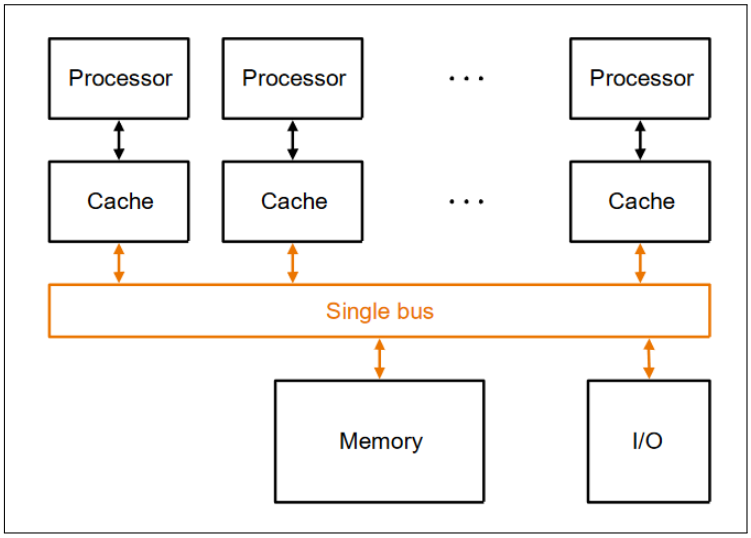


FIGURE 1 – Architecture multiprocesseur à base de bus.

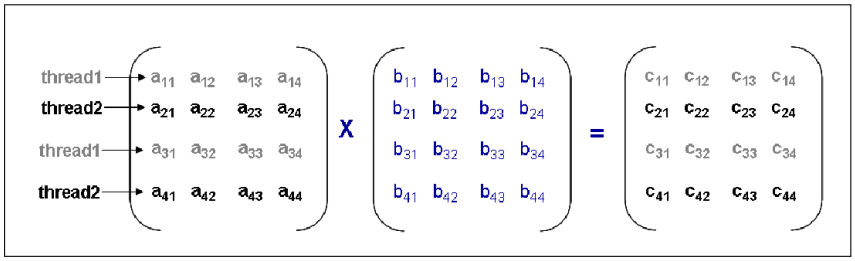


FIGURE 2 – Parallélisation du Produit matriciel.

Ici, on considère l'architecture ci-dessus, de type multicoeurs à base de bus et à un niveau de cache. Dans notre cas, les matrices A et B de la multiplication de matrices, sont stockées dans le bloc "**Memory**". Chaque mémoire "**Cache**" récupérera la ligne de la matrice A sur laquelle il opère, et l'ensemble de la matrice B pour que le processeur puisse réaliser la produit matriciel.

Il n'y a pas de problématique au niveau de **la cohérence des caches**, car les variables utilisées par chaque thread sont en lecture seule. De plus, il n'y a pas de concurrence d'écriture car chaque thread s'occupe d'une ligne différente.

3 Partie 2 : Paramètre de l'architecture multicoeurs

3.1 Question 2

Valeurs par défauts pour 6 différents paramètres. Les paramètres ont été sélectionnés car ils sont similaires à ceux intéressants dans le TD4. Il semble possible que LQEntries + SQEntries soient équivalents au lsq :size du TD4.

Nom du paramètre	Description	Valeur par défaut
fetchQueueSize	"Fetch queue size in micro-ops per-thread"	32
decodeWidth	"Decode width"	8
issueWidth	"Issue width"	8
commitWidth	"Commit width"	8
LQEntries	"Number of load queue entries"	32
SQEntries	"Number of store queue entries"	32

3.2 Question 3

Valeurs par défauts pour les 3 types de cache.

Nom du cache	Associativité	Taille du Cache	Taille de la ligne
Cache de données de niveau 1	2	64Kb	64
Cache d'instruction de niveau 1	2	32Kb	64
Cache unifié de niveau 2	8	2Mb	64

4 Partie 3 : Architecture multicoeurs avec des processeurs superscalaires in-order (Cortex A7)

4.1 Question 4

Le processus exécutant toujours le plus grand nombre de cycle est le processus 1. En effet, la parallélisation du produit matriciel se fait selon une procédure maître-esclave : le processus 1 crée m-1 processus esclaves qui effectuent chacun une partie du calcul. Puis ce processus attend la fin de l'exécution de tous les autres threads pour considérer la réussite du calcul. On peut donc en conclure que l'analyse du processus 1 en terme de nombre de cycle constitue une analyse du nombre total de cycle : le processus 1 est le premier à effectuer des opérations, et attend que tout le monde ait terminé pour clôturer sa propre tâche.

Pour la suite du problème, nous considérons une matrice carré de taille 16x16.

4.2 Question 5

Avec le graphe suivant, nous représentons le nombre de cycle effectué par le processus principal en fonction du nombre de threads.

Remarque : Nous nous sommes confrontés à une erreur de type *"Segmentation Fault (core dumped)"* pour 16 processus.

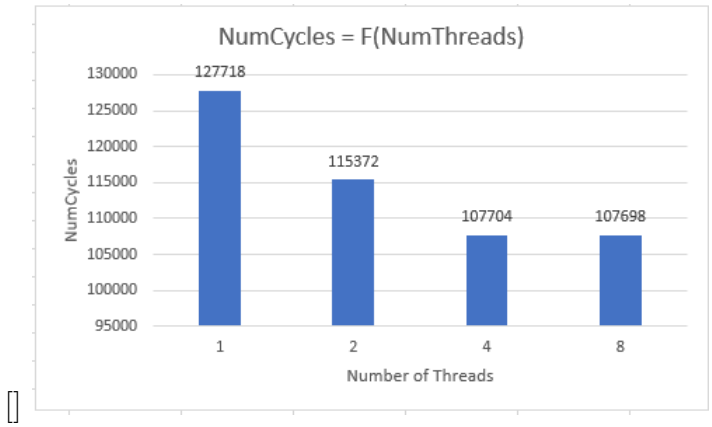


FIGURE 3 – Nombre de Cycle en fonction du nombre de threads

4.3 Question 6

On peut donc en déduire le Speed Up par rapport à 1 thread.

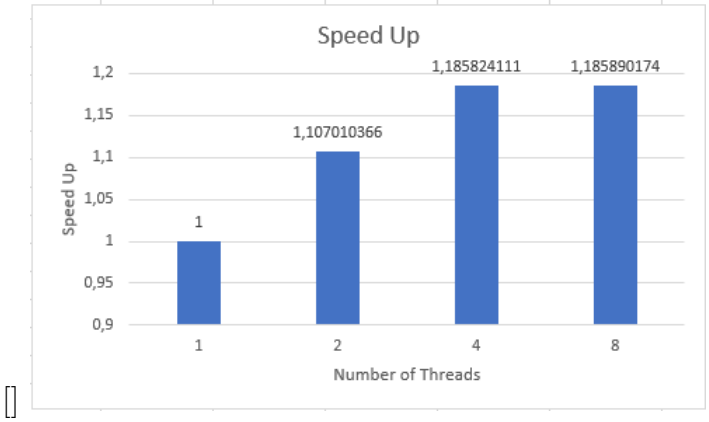


FIGURE 4 – Calcul du Speed Up

4.4 Question 7

Enfin, nous pouvons regarder le CPI ou l'IPC pour chaque configuration, facteurs qui constituent des critères de performance.

Pour chaque configuration, on commence par relever le nombre d'instruction totale simulées :

Nombre de Threads	Nombre d'instructions totales simulées
1	86730
2	100012
4	124108
8	187864

On a de plus les formules suivantes :

- $CPI = \frac{NumCycles}{NumInstructions}$
- $IPC = \frac{1}{CPI}$

On peut donc tracer l'évolution de ces critères de performance en fonction du nombre de threads :

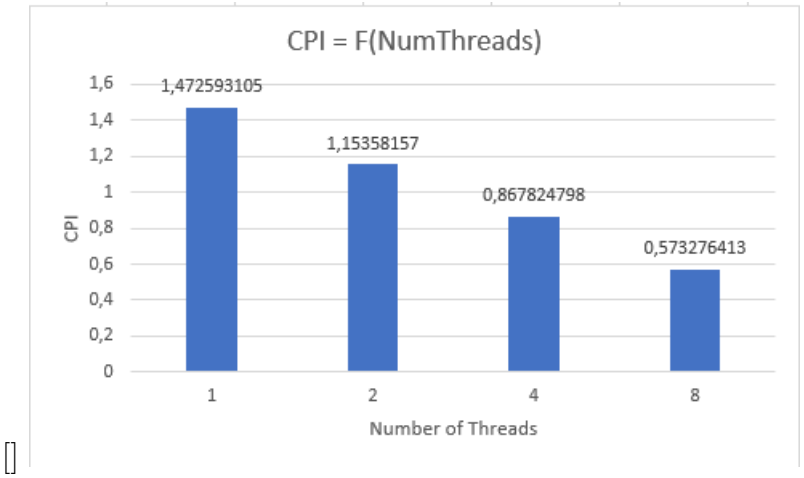


FIGURE 5 – CPI en fonction du nombre de threads

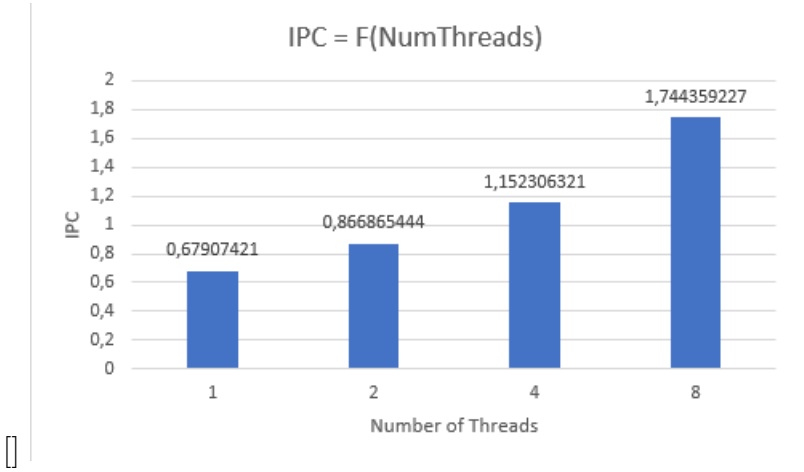


FIGURE 6 – IPC en fonction du nombre de threads

4.5 Question 8

Avec les résultats sur le Speed Up et sur le calcul de l'IPC (ou CPI), deux choses sont particulièrement remarquables :

Tout d'abord, augmenter le nombre de processus semble améliorer les performances de notre processus, et ce, avec une tendance quasi exponentielle comme le suggère le graphe de l'IPC. En effet, une façon d'augmenter les performances d'un processeur est de réussir à augmenter l'IPC, qui est le nombre moyen d'instructions exécutées par cycle d'horloge.

Cependant, le graphe sur l'évolution du Speed Up suggère que l'IPC ne constitue pas un critère de performance à lui seul. En effet, alors que l'IPC ne cesse de croître avec le nombre de threads, le Speed Up atteint rapidement un palier car le nombre de cycle effectué par le processus principal converge vers une limite. Au delà de 4 threads, on semble donc entrer dans un cas de **Memory Bound**, c'est-à-dire que la vitesse de calcul n'est plus limitée par le temps de calcul CPU, mais par des temps d'accès mémoire et de communication entre threads.

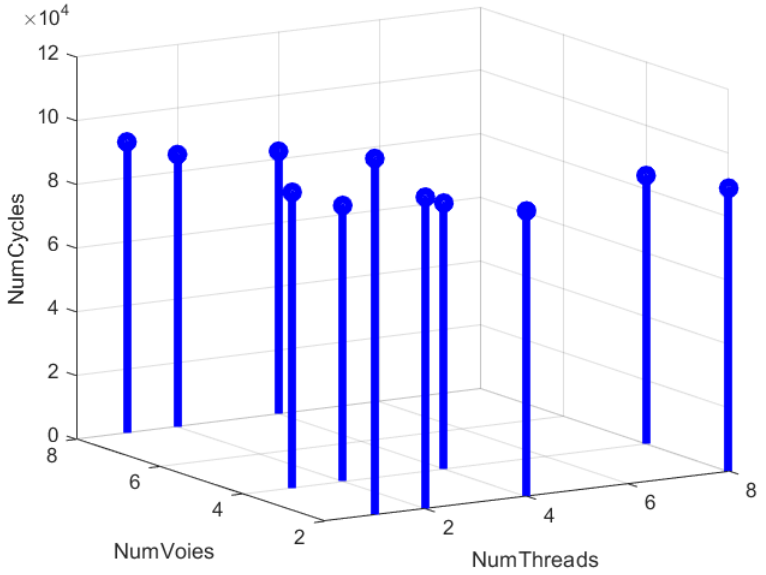
Nous tenons en dernier lieu soulever un point important. Les tendances obtenues semblent cohérentes : augmenter le nombre de coeurs et de threads par coeur impliquent naturellement une amélioration des performances en terme de temps d'exécution et d'instruction par cycle jusqu'à l'apparition d'un plateau qui témoigne de la multi-dimensionalité du problème d'optimisation considéré. Cependant, il est important de souligner que les évolutions ne sont pas flagrantes. Nous supposons alors que cela est dû à notre choix (ou notre erreur) d'avoir considérer des petites matrices (16x16). En effet, la parallélisation multi-processus offre généralement des améliorations de performances importantes pour des problèmes de grandes dimensions.

5 Partie 4 : Architecture multicoeurs avec des processeurs superscalaires out-of-order (Cortex A15)

5.1 Question 9

L'étude est similaire à celle réalisée sur le processeur A7, cependant nous allons ici considérer, en plus, la largeur du processeur A15 qui est superscalaire. Avec le graphe, ci-dessous, nous observons le nombre de cycles effectué en fonction du nombre de threads et de la largeur évoquée. (On choisit toujours le nombre de cycles maximum, donc celui du processus 1).

Les valeurs admissibles pour NumThreads sont les mêmes que prédédemment, et les valeurs admissibles pour NumVoies sont 1, 2, 4 et 8.
On constate que globalement le nombre de cycles diminue avec le nombre de threads et la largeur du superscalaire. Cependant nous avons une erreur du type "Segmentation Fault (core dumped)", pour la configuration $NumThreads = 8$ et $NumVoies = 8$.
Finalement la meilleur configuration est $NumThreads = 4$ et $NumVoies = 4$, pour laquelle on a 82546 cycles.



[]

FIGURE 7 – Nombre de Cycle en fonction du nombre de threads et de la largeur du processeur

5.2 Question 10

On en déduit alors le Speed Up par rapport à 1 thread d'une largeur de 1.
Il nous faut considérer un paramètre supplémentaire, la largeur du processeur, on va donc considérer la meilleure configuration pour chaque nombre de thread. (En général, la plus grande largeur possible).

Nombre de Threads	Nombre d'instructions totales simulées	Largeur du processeur
1	86730	8
2	116470	8
4	174986	8
8	235986	4

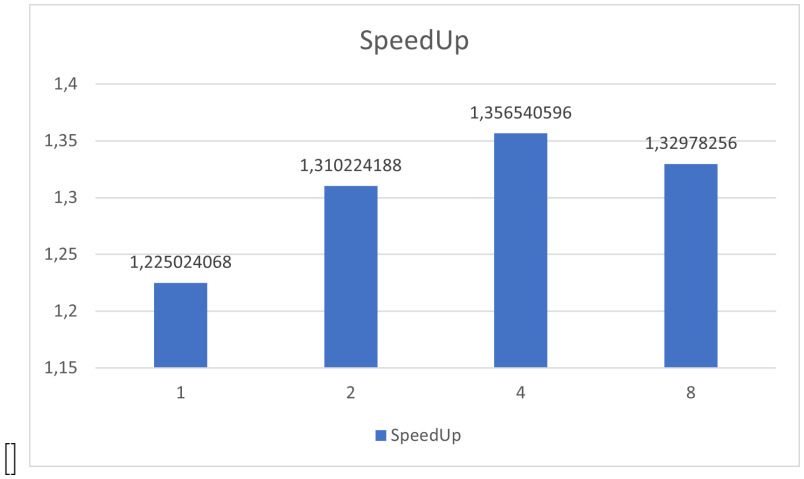


FIGURE 8 – SpeedUp en fonction de la meilleure configuration par nombre de threads

5.3 Question 11

Encore une fois, on ne va s'intéresser qu'aux meilleures configurations et déterminer l'IPC et le CPI, grâce aux formules précédentes :

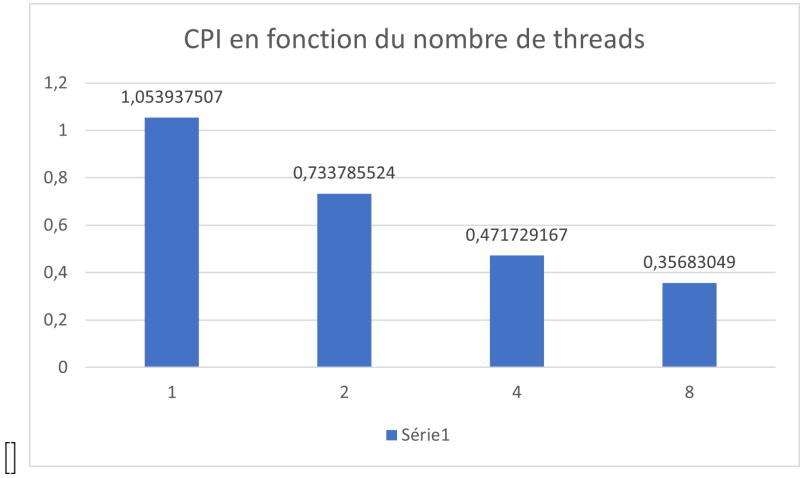


FIGURE 9 – CPI en fonction du nombre de threads

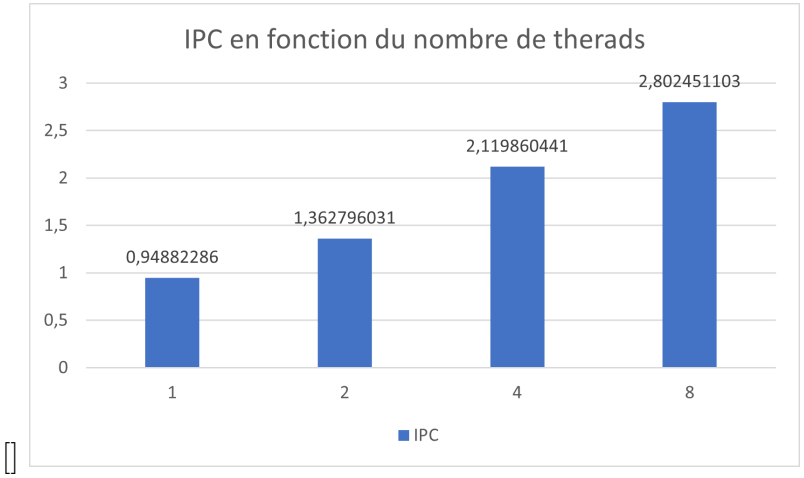


FIGURE 10 – IPC en fonction du nombre de threads

On pourra noter que le nombre d'instructions total augmente avec la largeur du processeur superscalaire. En effet, il se peut que les prédictions ne soient pas toujours correctes.
La graphique suivant illustre cette augmentation, pour un processus avec $NumThreads = 4$:

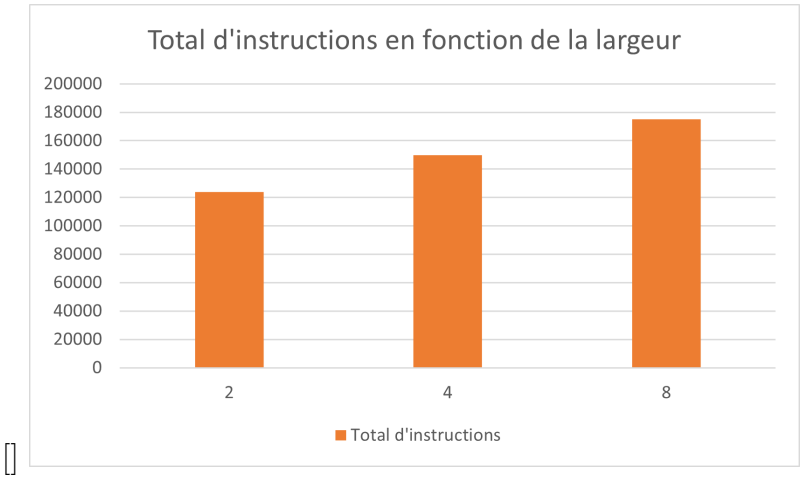


FIGURE 11 – Nombre total d'instructions en fonction de la largeur du processeur

5.4 Question 12

Certains aspects de la question 8 ne varient pas, en effet augmenter le nombre de processus continue d'améliorer les performances de l'exécution du programme. Il faut surtout souligner que l'augmentation de la largeur des processus permet une forte augmentation du nombre d'IPC (Presqu'une multiplication par 3).

Malheureusement, nous ne pouvons pas conclure sur le fait que nous atteignons un plateau au niveau du SpeedUp, comme à la question 8, car l'une des configurations produisait l'erreur *"Segmentation Fault (core dumped)"*. Cependant, l'augmentation de la largeur du processeur est directement liée à l'amélioration du SpeedUp. En effet, pour une largeur de 8 avec un seul thread, on obtient déjà un SpeedUp de 1.225. Et dans la meilleure configuration énoncée à la question 9, on atteint un speed up de1.357.

Ainsi, tout comme augmenter le nombre de threads et de coeurs, augmenter la largeur du processeur superscalaire permet naturellement l'amélioration des performances en terme de temps d'exécution et d'instructions par cycle. Comme remarqué à la question 8, les améliorations auraient sûrement été plus flagrantes, si nous avions considéré des matrices de tailles plus grandes (au-delà de matrices de taille 16x16).

6 Partie 5

6.1 Question 13

On recherche à maximiser l'efficacité surfacique.

Selon le TD4, le cortex A7 possède de loin l'efficacité surfacique la meilleure, cela sera alors notre choix de type de processeur pour le projet du concepteur. Pour l'optimisation de l'efficacité surfacique du cortex A7 on avait une taille de cache de 8KB pour Dijkstra et 4KB pour BlowFish.

Selon nos résultats dans la partie 3, la configuration la plus appropriée pour le cortex A7 semble être 4 coeurs et 4 threads. Ceci est valable pour les matrices de petites tailles sur lesquelles nous avons fait nos évaluations de Speed Up, où nous avions rencontré un palier d'accélération autour de ce stade, mais il est probable qu'avec des matrices de taille plus grande une quantité plus importante de coeurs et de threads pourrait nettement bénéficier la vitesse de calcul. Il faut faire des compromis. Augmenter la quantité de coeurs augmente directement la surface. Le concepteur priorisant l'efficacité énergétique, nous resterons sur 4 coeurs et 4 threads, car l'efficacité énergétique rediminue à partir d'une certaine surface.