TP5 : Architecture des microprocesseurs

Júlia Ellen Dias Leite
julia-ellen.dias@ensta-paris.fr

Charbel Chahla
charbel.chahla@ensta.fr

Martin Brun
martin.brun@ensta.fr

Paul-henri Djoko
paul-henri.djoko@ensta.fr

23 février 2026

Analyse théorique de cohérence de cache

Q1 : En considérant que chaque thread s'exécute sur un processeur dans une architecture de type multicoeurs à base de bus et 1 niveau de cache (comme décrit Figure 21), décrivez le comportement de la hiérarchie mémoire et de la cohérence des caches pour l'algorithme de multiplication de matrices. On supposera que le thread principal se trouve sur le processeur d'indice 1.

- **Matrice A :** Chaque ligne est stockée dans le cache d'un processeur différent chargée depuis la mémoire principale.
- Ainsi, chaque processeur a une ligne différente donc ne communique pas avec les autres, on a donc peu d'incohérence.
- Plus on a de threads moins on a d'incohérence.
- **Matrice B :** Elle est lue et stockée dans chaque thread pour calculer C mais elle n'est pas modifiée donc nous n'avons pas non plus d'incohérence.
- **Matrice C :** Par définition du produit de matrice chaque ligne de C correspond au produit de la ligne de A correspondante avec B, ainsi, chaque processus écrit des lignes différentes de C donc on n'a pas de problème de cohérence.

Paramètres de l'architecture multicoeurs

Q2 : Examinez le fichier de déclaration d'un élément de type « processeur superscalaire out- of-order », et présentez sous forme de tableau cinq paramètres configurables de ce type de processeur avec leur valeur par défaut. Choisissez de préférence des paramètres étudiés lors des séances TD/TP précédentes. Le fichier à consulter est le suivant :

```
$GEM5/src/cpu/o3/O3CPU.py
```

TABLE 1 – Paramètres de configuration du processeur

Paramètre	Signification	Valeur par défaut
fetchWidth	Nombre d'instructions récupérées par cycle	8
fetchBufferSize	Taille du buffer	64
decodeWidth	Nombre d'instructions décodées par cycle	8
cacheStorePorts	Nombre de ports disponibles pour accéder au cache (écriture)	200
issueWidth	Nombre d'instructions émises par cycle	8

Q3 : Examinez le fichier d'options de la plateforme se.py, puis déterminez et présentez sous forme de tableau les valeurs par défaut des paramètres suivants :

— Cache de données de niveau 1 : associativité, taille du cache, taille de la ligne
— Cache d'instructions de niveau 1 : associativité, taille du cache, taille de la ligne
— Cache unifié de niveau 2 : associativité, taille du cache, taille de la ligne

Le fichier d'options à consulter est le suivant :

```
$GEM5/configs/common/Options.py
```

TABLE 2 – Spécifications des caches

	L1 (l1d)	**L1 (l1i)**	**L2 (l2)**
Taille du cache	64kB	32kB	2MB
Taille de la ligne	64B	64B	64B
Associativité	2	2	8

Architecture multicoeurs avec des processeurs superscalaires in-order (Cortex A7)

Q4 : Déterminez quel est le processeur exécutant toujours le plus grand nombre de cycles. Expliquez pourquoi. Expliquez également pourquoi l'analyse du nombre de cycles sur ce processeur revient à analyser le nombre total de cycles d'exécution de l'application.

Q5 : Pour chaque configuration, quel est le nombre de cycles d'exécution de l'application ? Vous pourrez présenter vos résultats sous forme de graphe 2 axes

Q6 : Déduire le speedup par rapport à la configuration à 1 thread.

Q7 : En utilisant le nombre total d'instructions simulées, déterminez quelle est la valeur maximale de l'IPC pour chaque configuration ?

Q8 : Discussion et interprétation (max. 10 lignes).

Architecture multicoeurs avec des processeurs superscalaires out-of-order (Cortex A15)

La campagne de mesures a été exécutée sur le serveur distant ENSTA (hôte `salle`) avec le modèle `DerivO3CPU` (Cortex-A15) en mode SE, avec caches L1+L2 activés. Pour garantir une exécution reproductible, nous avons utilisé `gem5.fast` ARM pré-compilé, le script de configuration `se_a15.py`, ainsi que le binaire `test_omp` synchronisé dans `tp5_work`. Les paramètres explorés couvrent les largeurs `width` 2, 4 et 8, les nombres de threads OpenMP 1, 2, 4, 8 et 16, avec une taille de matrice fixée à 64.

Nous avons rencontré des `segfault` intermittents sur plusieurs configurations, ce qui a nécessité des relances ciblées. Pour améliorer la robustesse, nous avons ajouté les variables d'environnement `OMP_WAIT_POLICY=ACTIVE` et `GOMP_SPINCOUNT=1000000000`, afin de réduire les risques de blocage. De cette manière, nous avons pu collecter un ensemble de résultats plus complet, en executant la commande suivante pour chaque combinaison de paramètres :

```
OMP_NUM_THREADS=${T} OMP_WAIT_POLICY=ACTIVE GOMP_SPINCOUNT=1000000000 \
$GEM5/build/ARM/gem5.fast \
    --outdir=results/s64_w${W}_t${T} \
    se_a15.py --cpu-type=detailed --o3-width=${W} --num-cpus=${T} \
    --caches --l2cache \
    -c ./test_omp -o "${T} 64"
```

Le flux expérimental a été automatisé par un script de lancement global `run_all.sh`, qui itère sur les différentes configurations, puis par un script de post-traitement `extract_results.py` pour extraire les métriques pertinentes et générer les figures du rapport. Les indicateurs analysés incluent le temps d'exécution simulé (`sim_seconds`), le nombre maximal de cycles (`cycles_max_cpu`), l'IPC maximal (`ipc_max_cpu`), le nombre total de ticks simulés (`sim_ticks`) et le nombre total d'instructions exécutées (`insts_max_cpu`).

Q9 : Pour chaque configuration, quel est le nombre de cycles d'exécution de l'application ? Vous pourrez présenter vos résultats sous forme de graphe 3 axes.

Temps d'exécution simulé (sim_seconds) en fonction du nombre de threads et de la largeur du processeur :

TABLE 3 – Temps d'exécution (sim_seconds) par largeur de processeur et nombre de threads.

Width	1 thread	2 threads	4 threads	8 threads	16 threads
2 voies	0.001151	0.000638	0.000382	0.000254	0.000196
4 voies	0.000680	0.000398	0.000258	0.000188	0.000157
8 voies	0.000665	0.000390	0.000253	0.000188	0.000158

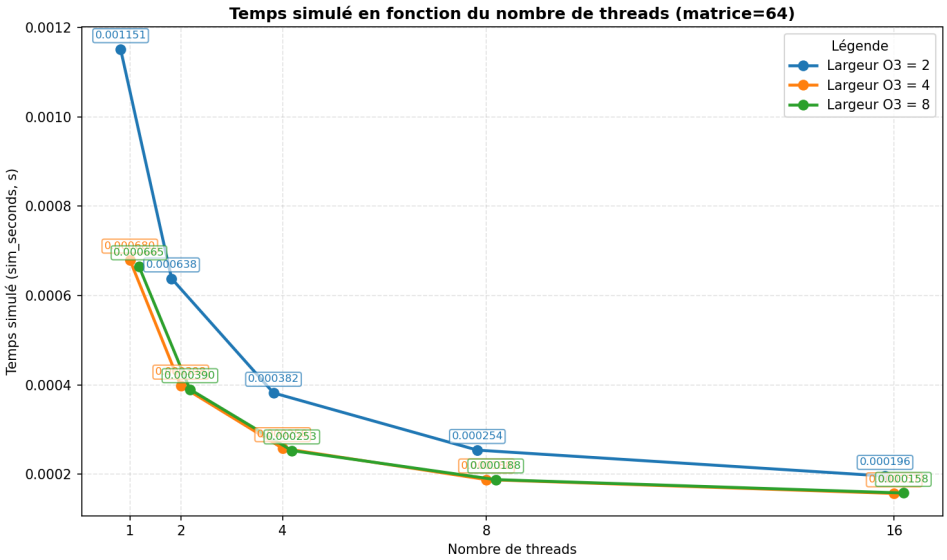

FIGURE 1 – Temps simulé en 2D (threads vs largeur O3).

Les résultats montrent que le gain principal vient du passage de 1 à 4 threads ; au-delà, la pente se réduit nettement, ce qui indique une saturation progressive. À charge identique, la configuration **2 voies** reste systématiquement derrière **4/8 voies**, ce qui confirme l'apport de l'élargissement du coeur pour exploiter l'ILP. L'écart entre **4 voies** et **8 voies** devient toutefois faible à fort parallélisme (8–16 threads), ce qui suggère que la limite se déplace du front-end vers la hiérarchie mémoire et la synchronisation. Le point le plus lent observé est **2 voies / 1 thread** (0.001151 s), tandis que le plus rapide est **4 voies / 16 threads** (0.000157 s), très proche de **8 voies / 16 threads** (0.000158 s).

Remarque : Le fichier de résultats inclut aussi `sim_ticks`, `cycles_max` et `insts_max`, ce qui permet d'analyser les tendances en cycles en plus du temps simulé.

Q10 : Déduire le speedup par rapport à la configuration à 1 thread.

Le **speedup** est défini comme le rapport du temps de base (1 thread) au temps pour n threads :

$$\text{Speedup}(n) = \frac{T_1}{T_n}$$

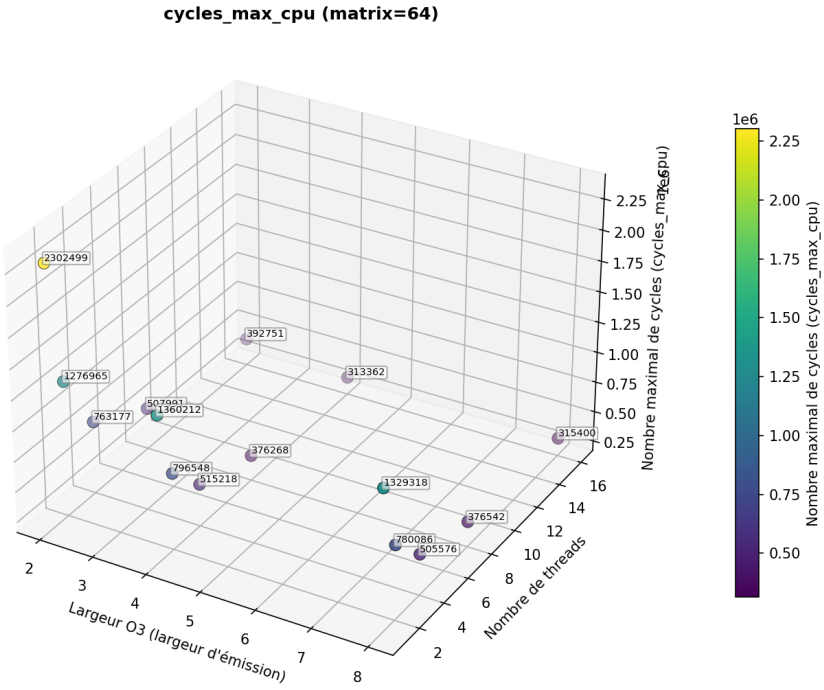

FIGURE 2 – Cycles maximaux en représentation 3D.

TABLE 4 – Speedup par rapport à 1 thread.

Width	2	4	8	16
2 voies	1.80	3.01	4.53	5.87
4 voies	1.71	2.64	3.62	4.33
8 voies	1.71	2.63	3.54	4.21

Le speedup reste **sub-linéaire** sur l'ensemble des configurations : il augmente avec le nombre de threads, mais pas de manière proportionnelle. En **2 voies**, le gain atteint **5.87×** à 16 threads, contre **4.33×** en **4 voies** et **4.21×** en **8 voies**. Ce résultat s'explique par un effet de base : la configuration 2 voies part d'un cas 1-thread plus lent, ce qui amplifie le speedup relatif, alors qu'en valeur absolue les configurations 4 et 8 voies restent les plus rapides.

Q11 : En utilisant le nombre total d'instructions simulées, déterminez quelle est la valeur maximale de l'IPC pour chaque configuration ?

L'**IPC (Instructions Per Cycle)** mesure le parallélisme au niveau des instructions exécutées par cycle d'horloge.

TABLE 5 – IPC par largeur du processeur et nombre de threads.

Width	1 thread	2 threads	4 threads	8 threads	16 threads
2 voies	1.784232	1.927467	1.923079	1.910628	1.880036
4 voies	3.020259	3.537798	3.514918	3.457122	3.282794
8 voies	3.090452	3.617589	3.598638	3.677927	3.808231

L'IPC augmente nettement avec la largeur du coeur : dès 1 thread, les configurations 4 et 8 voies sont largement au-dessus de 2 voies. Le meilleur IPC observé est **3.808** (8 voies, 16 threads), ce qui confirme un meilleur remplissage du pipeline pour les coeurs les plus larges. À l'inverse, en 2 voies l'IPC plafonne autour de 1.9, tandis qu'en 4/8 voies il reste dans la plage 3.2–3.8. L'écart entre 4 et 8 voies devient néanmoins modéré sur plusieurs points, ce qui suggère qu'une partie du potentiel supplémentaire est absorbée par la contention mémoire et la synchronisation OpenMP.

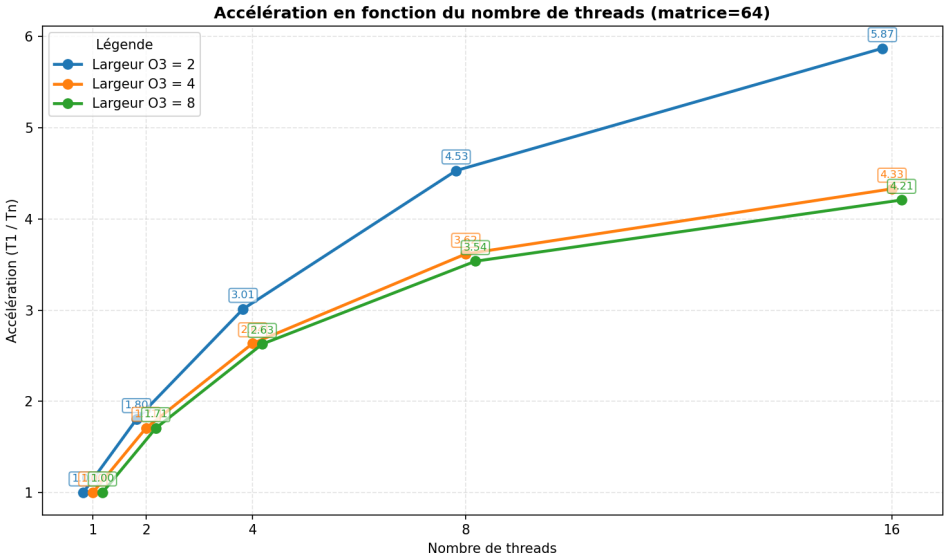

FIGURE 3 – Accélération en fonction du nombre de threads.

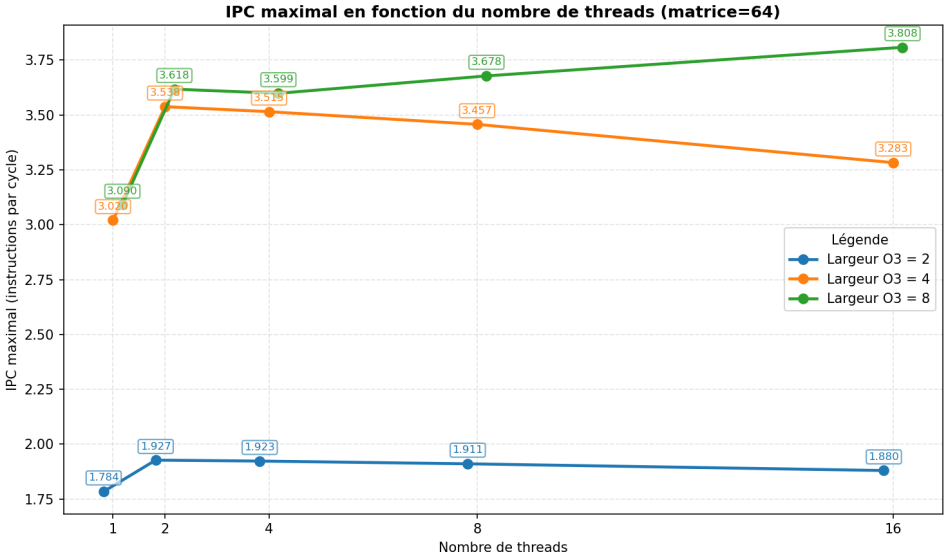

FIGURE 4 – IPC maximal en fonction du nombre de threads.

Q12 : Discussion et interprétation (max. 10 lignes)

Les résultats montrent que l'augmentation de `threads` et de `width` réduit globalement les cycles, mais avec un gain sous-linéaire à cause des surcoûts de synchronisation et de la mémoire partagée. L'écart entre `width=4` et `width=8` se réduit à forte concurrence, signe d'une saturation progressive. Nous avons aussi observé une limite de stabilité en mode SE (`segfault` à forte charge), partiellement atténuée par `OMP_WAIT_POLICY=ACTIVE` et un `GOMP_SPINCOUNT` élevé. En résumé, les meilleures performances absolues sont obtenues avec des coeurs plus larges, tandis que le speedup relatif maximal dépend fortement du point de référence mono-thread.

Configuration CMP la plus efficace

Q13 : Proposez une configuration ou une gamme de configuration de l'architecture CMP (nombre de threads de l'application test omp, nombre et type de coeurs) qui vous semble la plus appropriée si la contrainte recherchée par le concepteur du système est l'efficacité surfacique ? Discussion et interprétation (max. 10 lignes).

Q14 : Au regard de l'évolution théorique du speedup et son évolution constatée lors des questions précédentes, proposez une tentative d'explication (max. 10 lignes).