

TP4 – Exercice 4 (Q5)

Evaluation de performances (Cortex A15) en fonction de la taille des caches L1

Nom: _____ Groupe: _____

Année 2026

1 Objectif

Le but de la question Q5 est d'évaluer les performances d'un coeur *type Cortex A15* (modèle RISC-V dans gem5 en *out-of-order*) en faisant varier simultanément la taille des caches L1 (instructions et données) de 2 kB à 32 kB, avec un cache L2 fixé à 512 kB. On compare les résultats sur deux applications MiBench : **Dijkstra** et **Blowfish**.

2 Configuration et méthodologie

2.1 Paramètres processeur / caches

Pour le coeur "A15" (Tableau 12 du sujet), on utilise :

- CPU : Deriv03CPU (gem5 classic), mode `timing`.
- Prédiction : TournamentBP avec `BTBEntries=256`.
- Largeurs : `decode=4`, `issue=8`, `commit=4` ; `ROB=16` ; `LQ=16` ; `SQ=16` ; `fetchQueue=15`.
- Lignes de cache : 64 B (et `fetchBufferSize` mis à 64 B).
- L2 : 512 kB, `assoc=16`, `bloc=64 B` (fixe).
- L1I et L1D : `assoc=2`, `bloc=64 B`, taille variable : {2,4,8,16,32} kB.

2.2 Benchmarks et entrées

- **Dijkstra**: exécution de `dijkstra_small` avec `input.dat` (MiBench).
- **Blowfish**: on a mesuré `bf` en chiffrement (`e`) + déchiffrement (`d`) sur `input_small.asc`. Les courbes "Blowfish" correspondent à l'agrégation *enc+dec* (somme des cycles et des instructions).

2.3 Commande gem5 (paramètres d'exécution)

Les simulations ont été lancées via un script pour balayer les tailles de L1.

```
PYTHONUTF8=1 /opt/gem5/build/RISCV/gem5.opt -r -d <outdir> \
gem5/se_A15_q5.py --l1 <2kB|4kB|8kB|16kB|32kB> \
--maxinsts 0 \
--cmd <binary> --options <args>
```

Le script utilisé est `scripts/run_q5_a15.sh`. Les résultats bruts sont dans `m5out/q5/A15_nolimit_20260210/`.

Remarque (comparaison juste). Pour comparer les tailles de cache de manière équitable, les exécutions sont faites *jusqu'à la fin du programme* (`-maxinsts 0`).

3 Résultats

3.1 Synthèse (meilleure configuration)

La table suivante résume la configuration L1 donnant les meilleurs résultats (critère : cycles minimaux).

Aplicacao	L1 (KB)	Ciclos	IPC	MR(IL1)	MR(DL1)
Dijkstra	32	28456213	1.296	0.0002	0.0195
Blowfish (enc+dec)	32	61652380	1.672	0.0001	0.0004

3.2 Dijkstra

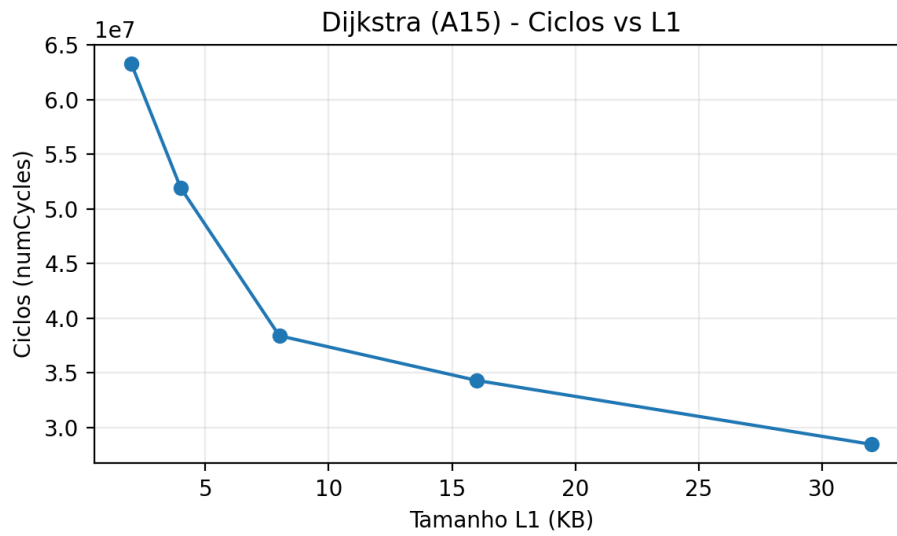


Figure 1: Dijkstra (A15) : cycles en fonction de la taille L1.

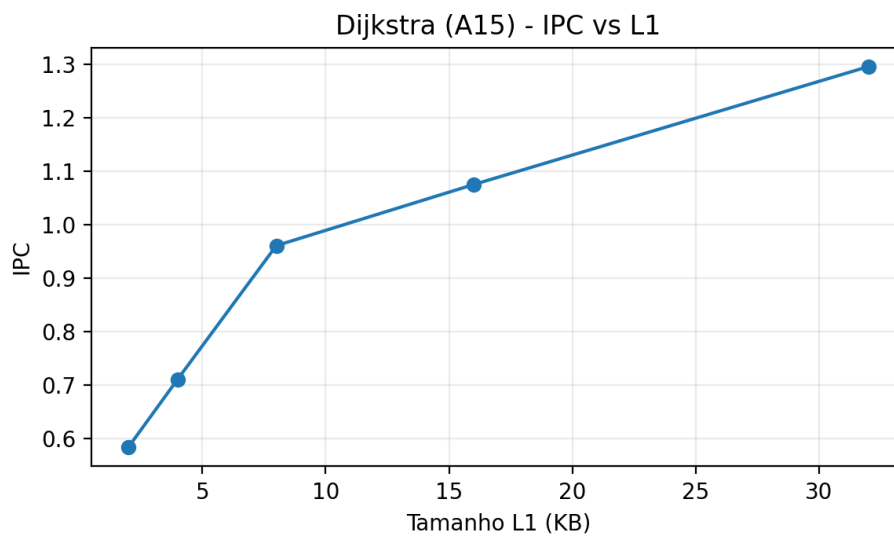


Figure 2: Dijkstra (A15) : IPC en fonction de la taille L1.

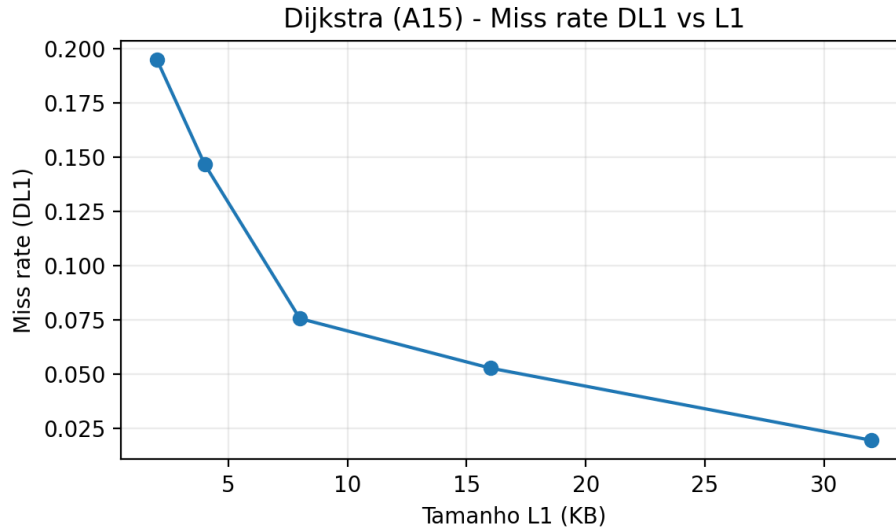


Figure 3: Dijkstra (A15) : miss rate DL1.

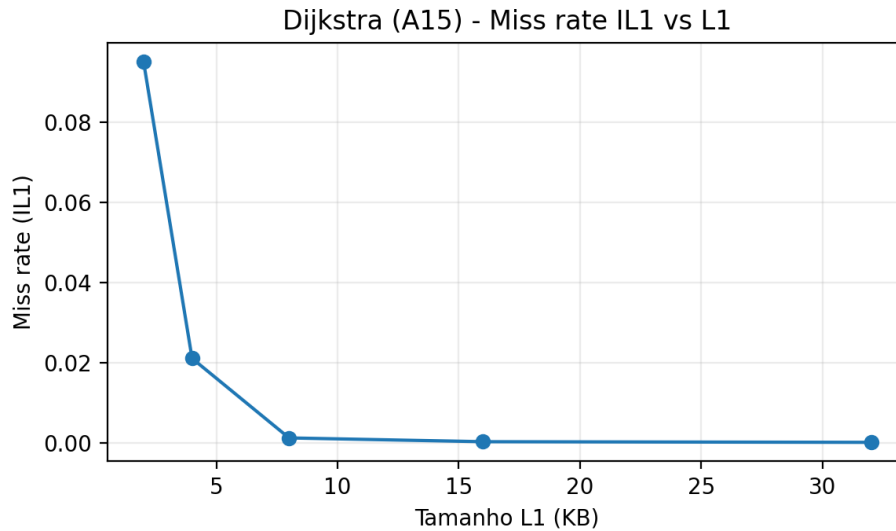


Figure 4: Dijkstra (A15) : miss rate IL1.

Analyse (Dijkstra). Quand on augmente L1, les miss rates (surtout DL1) diminuent et les cycles baissent, ce qui améliore l’IPC. Comme pour Q4, le taux de mauvaise prédiction varie peu avec la taille de L1 : les gains sont dominés par la hiérarchie mémoire plutôt que par la qualité de prédiction.

3.3 Blowfish (enc+dec)

Analyse (Blowfish). Blowfish est très *compute-bound* dans nos résultats (miss rates faibles), donc l’IPC varie moins fortement avec L1 que pour Dijkstra. Une taille L1 suffisante permet d’éviter des misses résiduelles, puis les gains saturent.

4 Conclusion

Pour le coeur A15, augmenter L1 améliore globalement les performances au début (moins de misses, plus d’IPC), puis les gains se stabilisent. La “meilleure” taille est celle qui minimise les cycles pour l’application considérée.

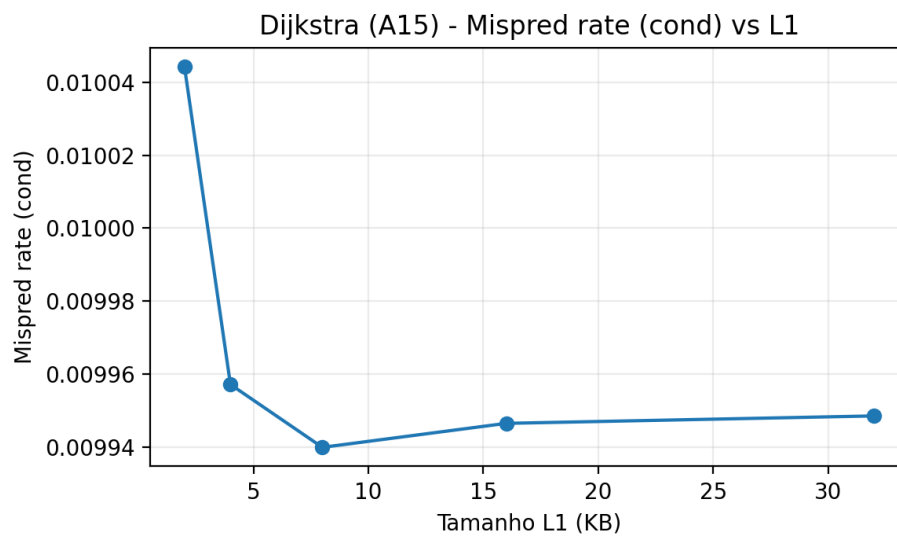


Figure 5: Dijkstra (A15) : taux de mauvaise prédiction (branches conditionnelles) vs taille L1.

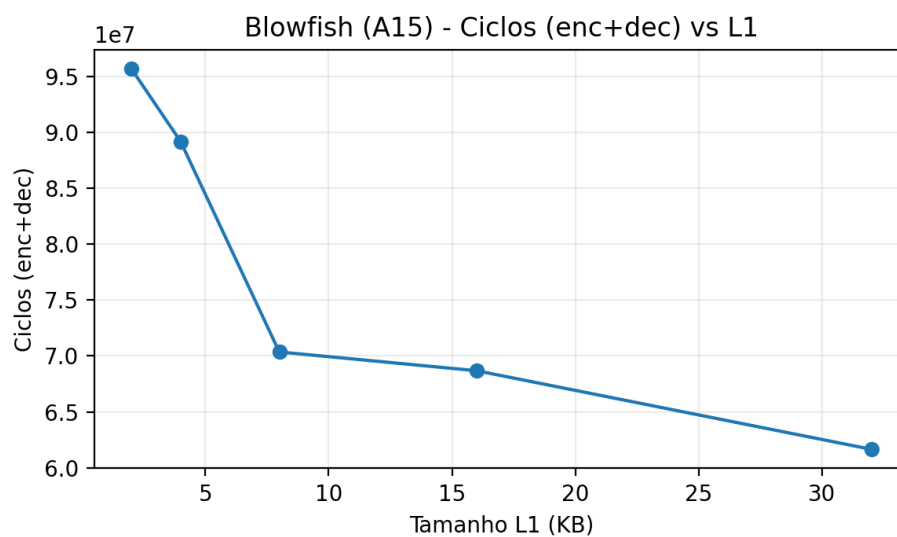


Figure 6: Blowfish (A15) : cycles (chiffrement+déchiffrement) vs L1.

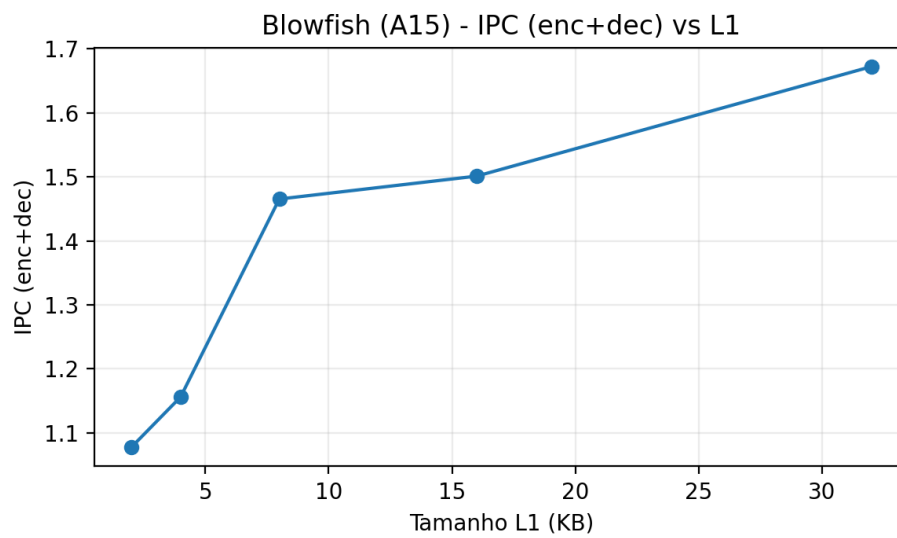


Figure 7: Blowfish (A15) : IPC (enc+dec) vs L1.

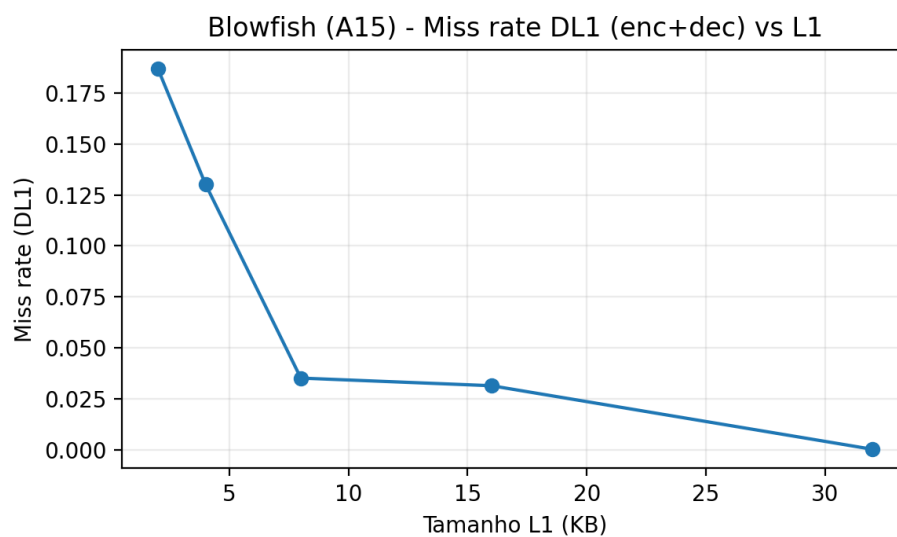


Figure 8: Blowfish (A15) : miss rate DL1 (enc+dec).

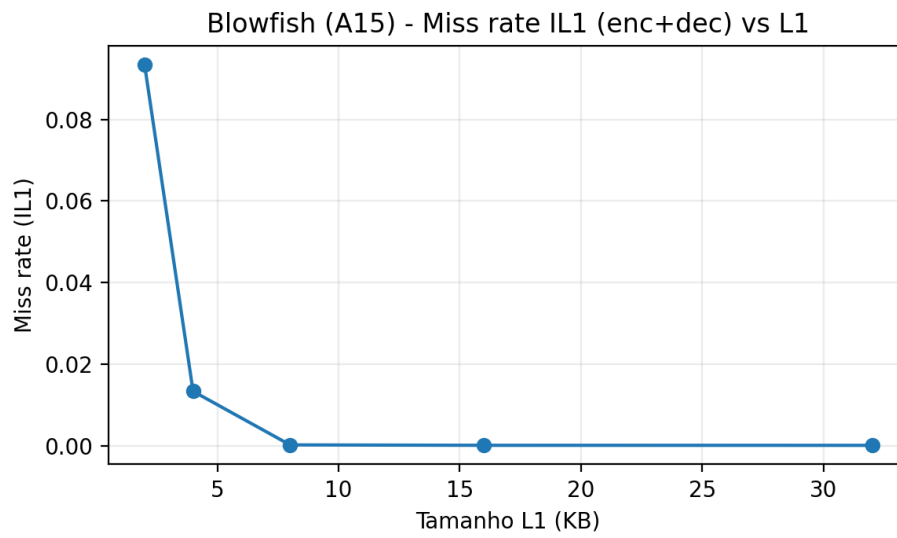


Figure 9: Blowfish (A15) : miss rate IL1 (enc+dec).

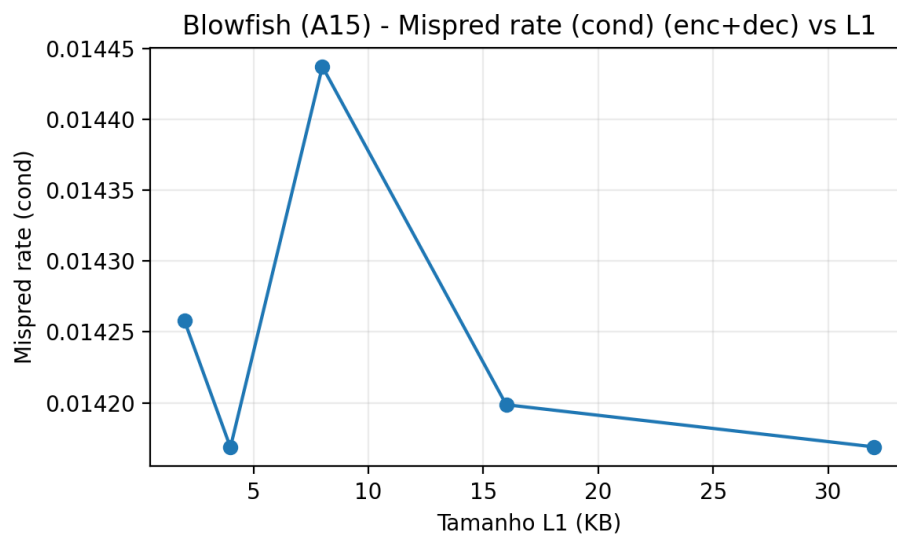


Figure 10: Blowfish (A15) : taux de mauvaise prédiction (branches conditionnelles) vs taille L1.