TP4 : Architecture des microprocesseurs

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

Charbel Chahla
charbel.chahla@ensta.fr

Martin Brun
martin.brun@ensta.fr

16 février 2026

Exercice 3 : Analyse des Performances de la Hiérarchie Mémoire

Q1.Configurer le simulateur gem5 avec deux hiérarchies de caches différentes (C1 et C2) et extraire les taux de défauts (Miss Rates) pour l'Instruction Cache (L1I), le Data Cache (L1D) et le cache unifié de niveau 2 (L2).

TABLE 1 – Paramètres de cache pour chaque configuration

Configuration	IL1	DL1	UL2
C1	4kB, DM (assoc=1)	4kB, DM (assoc=1)	32kB, DM (assoc=1)
C2	4kB, DM (assoc=1)	4kB, 2-way (assoc=2)	32kB, 4-way (assoc=4)

Q2. Remplir les tableaux de mesures pour les différents programmes fournis (P1 à P4).

TABLE 2 – Instruction Cache (il1) Miss Rate

Programmes	Configurations de caches	
	C1	C2
P1 (normale)	0.000119	0.000119
P2 (pointeur)	0.000089	0.000089
P3 (tempo)	0.000123	0.000123
P4 (unrol)	0.000141	0.000141

Q3. Analysez les résultats obtenus. Pourquoi observe-t-on des variations de performance entre C1 et C2 ? Quel est l'impact de l'associativité sur les taux de défauts observés ?

D'après les résultats obtenus dans les Tableaux 9, 10 et 11, nous pouvons tirer les conclusions suivantes :

1. **Analyse de l'IL1 :** Le taux de défauts reste identique entre C1 et C2. Cela s'explique par le fait que la configuration de l'IL1 n'a pas été modifiée (4kB, Direct Mapped dans les deux cas).

TABLE 3 – Data Cache (dl1) Miss Rate

Programmes	Configurations de caches	
	C1	C2
P1 (normale)	0.298316	0.306917
P2 (pointeur)	0.299729	0.308452
P3 (tempo)	0.299724	0.308445
P4 (unrol)	0.300090	0.305467

TABLE 4 – Unified Cache (ul2) Miss Rate

Programmes	Configurations de caches	
	C1	C2
P1 (normale)	0.437203	0.423355
P2 (pointeur)	0.437227	0.423262
P3 (tempo)	0.437231	0.423260
P4 (unrol)	0.434496	0.425146

2. **Analyse de la DL1 :** On observe une légère augmentation du Miss Rate en C2 (2-way) par rapport à C1 (DM) pour certains programmes. Bien que l'associativité réduise normalement les défauts de conflit, dans des caches de très petite taille (4kB), l'algorithme de remplacement (LRU) peut parfois évincer des données utiles prématurément par rapport à un mapping direct, ou la structure de l'accès aux données des boucles favorise un mapping fixe.
3. **Analyse de l'UL2 :** L'augmentation de l'associativité (de DM à 4-way) dans le cache L2 de 32kB montre une amélioration systématique (baisse du Miss Rate) pour C2. Cela démontre que pour un cache de second niveau recevant des flux de données et d'instructions, une associativité plus élevée est cruciale pour réduire les défauts de conflit entre les blocs provenant de la L1I et de la L1D.

Exercice 4 : Mémoires caches - Evaluation des performances de différentes configurations de mémoires caches (instructions et données)

L'objectif principal est la caractérisation détaillée de deux cœurs distincts : le Cortex-A7 (cœur à haut rendement et à exécution séquentielle) et le Cortex-A15 (cœur à hautes performances et à exécution déséquilibrée), à l'aide du simulateur d'architecture gem5 et de l'outil de modélisation physique CACTI.

Profiling de l'application

Q1. Générez le pourcentage de chaque classe d'instructions de ces applications et remplissez les valeurs dans un tableau.

Pour générer le pourcentage de chaque classe d'instructions, nous avons utilisé les statistiques de performance extraites du simulateur gem5 pour les applications Blowfish et Dijkstra. Les classes d'instructions sont regroupées en catégories telles que IntAlu, IntMult, IntDiv, MemRead, MemWrite, FloatMemWrite, et No_OpClass.

TABLE 5 – Pourcentage par classe d'instructions (blowfish, dijkstra)

Classe	Blowfish	Dijkstra
IntAlu	65.48%	63.95%
IntMult	0.00%	3.30%
IntDiv	0.00%	0.00%
MemRead	22.51%	22.38%
MemWrite	12.01%	10.36%
FloatMemWrite	0.00%	0.00%
No_OpClass	0.00%	0.00%

Q2. Quelle catégorie d'instructions nécessiterait une amélioration de performances ? Expliquez en quelques lignes (max 5 lignes).

La catégorie la plus critique est MemRead/MemWrite (mémoire), car elle représente une part importante des instructions et subit les latences mémoire. Améliorer cette catégorie (meilleure hiérarchie de caches, prélecture, réduction des accès) aura l'impact le plus direct sur le temps d'exécution. Les IntAlu dominent en volume mais sont déjà rapides, donc moins sensibles.

Q3. Au regard des résultats obtenus lors du TP2, pouvez-vous justifier d'éventuelles similitudes/divergences comportementales entre dijkstra, BlowFish, SSCA2-BCS, SHA-1 et le produit de polynômes ?

Dijkstra et Blowfish ont des profils d'instructions similaires (dominance d'IntAlu et MemRead/Write), ce qui explique des comportements de cache comparables.

Evaluation des performances

Nous allons nous baser sur les métriques suivantes pour analyser les questions 4 et 5 :

- **IPC (Instructions Par Cycle)** : nombre moyen d'instructions exécutées par cycle. Plus l'IPC est élevé, meilleure est la performance.
- **CPI (Cycles Par Instruction)** : nombre moyen de cycles nécessaires pour exécuter une instruction. Plus le CPI est faible, meilleure est la performance.
- **I-miss rate (L1I)** : taux de défauts du cache d'instructions. Un taux élevé indique des fetchs fréquents en mémoire plus lente.
- **D-miss rate (L1D)** : taux de défauts du cache de données. Un taux élevé implique des latences mémoire plus importantes.
- **L2 miss rate** : taux de défauts du cache de niveau 2, impactant la pression sur la mémoire principale.
- **Branch misprediction rate** : proportion de branches mal prédites. Un taux élevé pénalise le pipeline et réduit l'IPC.

Q4 : Générez les figures de performances détaillées (performance générale, IPC, hiérarchie mémoire, prédiction de branchement, etc.) en fonction de la taille du cache L1 pour les configurations testées. Analysez les résultats. Quelle configuration de L1 donne les meilleures performances pour le Cortex A7 pour les applications sélectionnées ?

Analyse des résultats :

Les figures ci-dessous présentent l'évolution des métriques de performance en fonction de la taille du cache L1 pour les applications Blowfish et Dijkstra sur le Cortex A7.

TABLE 6 – Synthèse des métriques pour différentes tailles de cache L1 (Cortex A7)

Prog.	Taille L1	IPC	CPI	I-Miss (%)	D-Miss (%)	L2-Miss (%)	Br. Misp (%)
Blowfish	1kB	0.20	5.04	54.98	49.23	4.32	3.83
	2kB	0.21	4.87	2.73	41.00	8.42	3.83
	4kB	0.22	4.56	2.40	27.46	12.33	3.82
	8kB	0.25	3.97	2.04	5.79	58.98	3.82
	16kB	0.25	3.97	1.88	5.73	60.48	3.82
Dijkstra	1kB	0.23	4.32	14.17	23.61	0.10	1.99
	2kB	0.24	4.14	12.11	17.50	0.13	1.99
	4kB	0.25	3.95	5.59	13.10	0.19	1.98
	8kB	0.28	3.63	0.52	6.14	0.49	1.99
	16kB	0.28	3.54	0.03	4.12	0.77	1.98

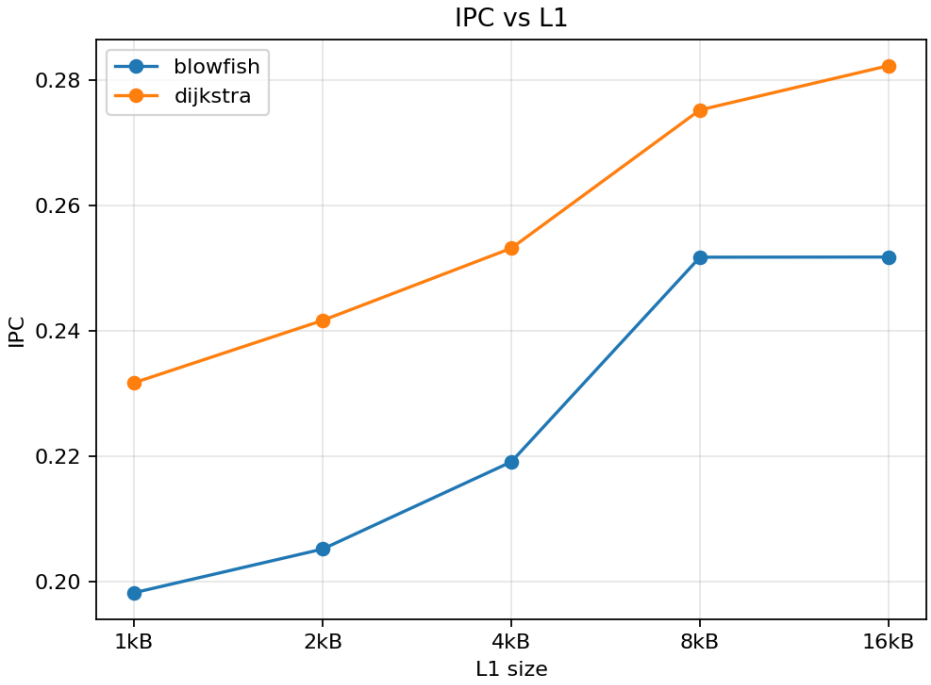
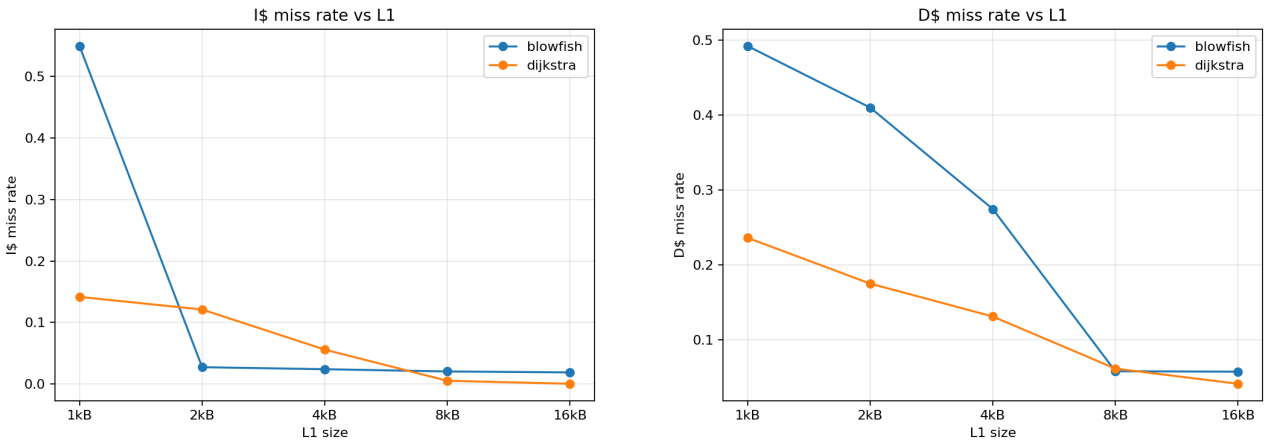


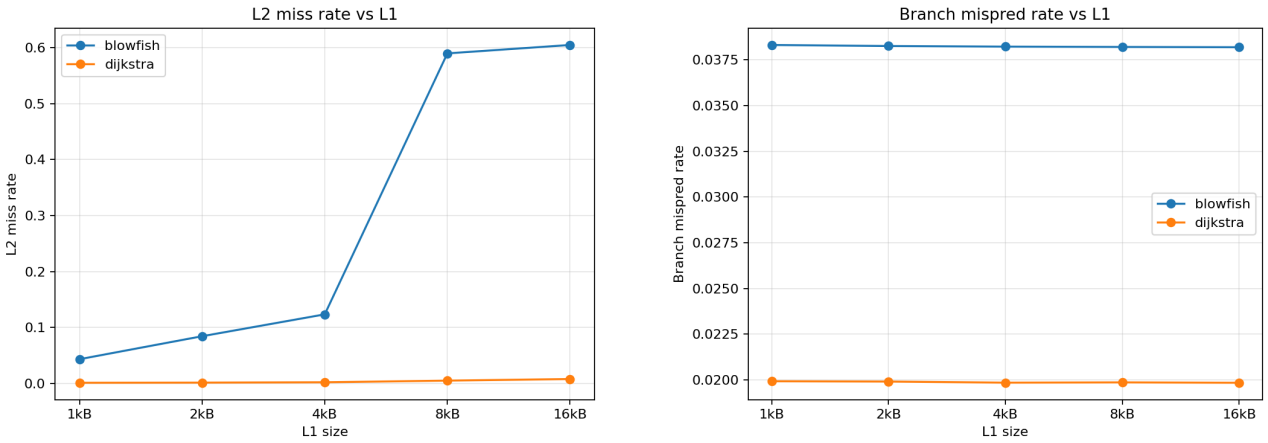
FIGURE 1 – IPC comparé - Cortex A7



(a) Taux de défauts L1I

(b) Dijkstra

FIGURE 2 – Taux de défauts L1D



(a) Taux de défauts L2

(b) Taux de mauvaises prédictions de branchement

FIGURE 3 – Taux de défauts L2 et prédiction de branchement comparés - Cortex A7

D'après les résultats obtenus, on observe que :

— **Performance générale (IPC/CPI)** : L'IPC augmente progressivement avec la taille du cache L1, atteignant son maximum à 8kB pour les deux applications. Au-delà de 8kB, les gains sont marginaux (plateau de performance), car le working set des applications tient déjà entièrement dans le cache.

— **Cache d'instructions (L1I)** : Blowfish présente des taux de défauts élevés pour les petites tailles (54.98% à 1kB), qui chutent drastiquement à partir de 2kB. Dijkstra montre une décroissance plus graduelle. Cela indique que Blowfish a un footprint d'instructions plus compact que Dijkstra.

— **Cache de données (L1D)** : Les deux applications montrent une forte sensibilité à la taille du cache de données. Blowfish atteint un minimum à 8kB (5.79%), tandis que Dijkstra continue à bénéficier de tailles plus grandes (4.12% à 16kB). Ceci reflète des patterns d'accès mémoire différents entre les deux workloads.

— **Cache L2** : Le taux de défauts L2 de Blowfish augmente avec la taille du L1. Cela s'explique par le fait qu'un L1 plus grand capture plus de défauts, mais ceux qui passent au L2 sont plus difficiles à satisfaire.

— **Prédiction de branchement** : Le taux reste stable indépendamment de la taille du cache, ce qui est attendu car la prédiction dépend uniquement du prédicteur BiMode et non de la hiérarchie mémoire.

Conclusion : Pour le Cortex A7, la configuration **8kB** offre le meilleur compromis performance/surface pour les deux applications. Au-delà, les gains marginaux ne justifient pas l'augmentation de la surface du cache.

Q5 : Générez les figures de performances détaillées (performance générale, IPC, hiérarchie mémoire, prédiction de branchement, etc.) en fonction de la taille du cache L1 pour les configurations testées. Analysez les résultats. Quelle configuration de L1 donne les meilleures performances pour le Cortex A15 pour les applications sélectionnées ?

TABLE 7 – Résumé des performances en fonction de la taille du cache L1 (Cortex A15)

Prog.	Taille L1	IPC	CPI	I-Miss (%)	D-Miss (%)	L2-Miss (%)	Br. Mispred (%)
Blowfish	2kB	0.68	1.47	2.54	49.84	5.52	5.43
	4kB	0.71	1.40	2.23	33.92	7.94	5.43
	8kB	0.90	1.12	1.82	8.77	44.68	5.43
	16kB	0.90	1.11	1.58	8.71	46.12	5.41
	32kB	0.95	1.05	1.51	4.34	95.43	5.41
Dijkstra	2kB	0.64	1.55	15.72	16.93	0.09	2.86
	4kB	0.72	1.38	10.07	12.05	0.12	2.92
	8kB	0.94	1.06	1.19	5.57	0.31	2.86
	16kB	1.01	0.99	0.42	3.63	0.50	2.94
	32kB	1.14	0.88	0.02	1.09	1.81	2.94

Les figures ci-dessous présentent l'évolution des métriques de performance en fonction de la taille du cache L1 pour les applications Blowfish et Dijkstra sur le Cortex A15.

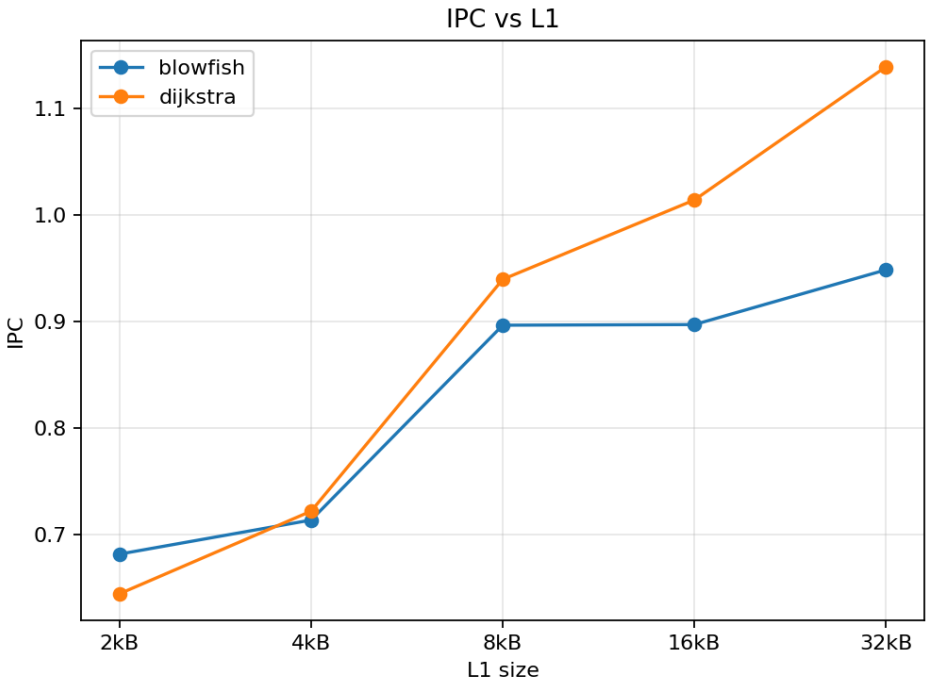
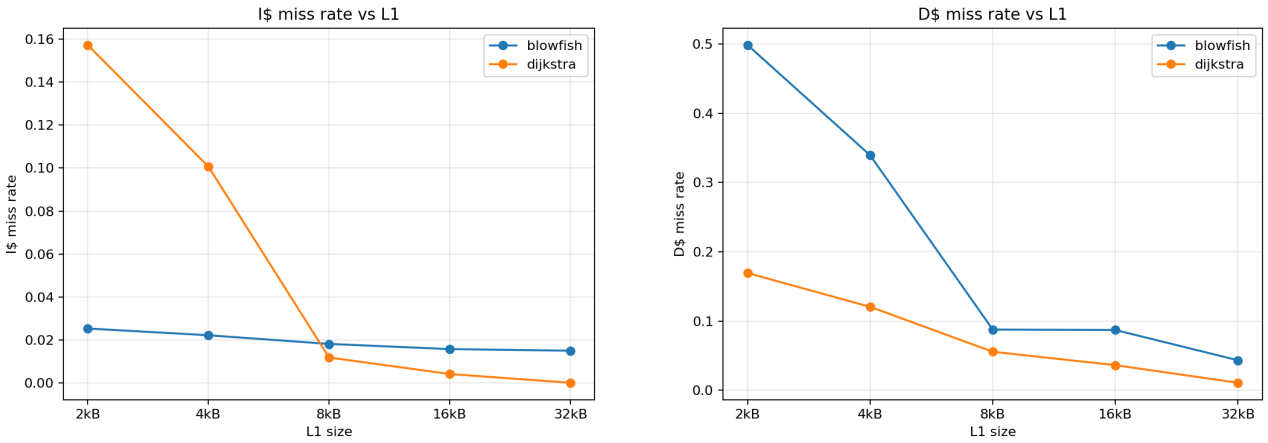


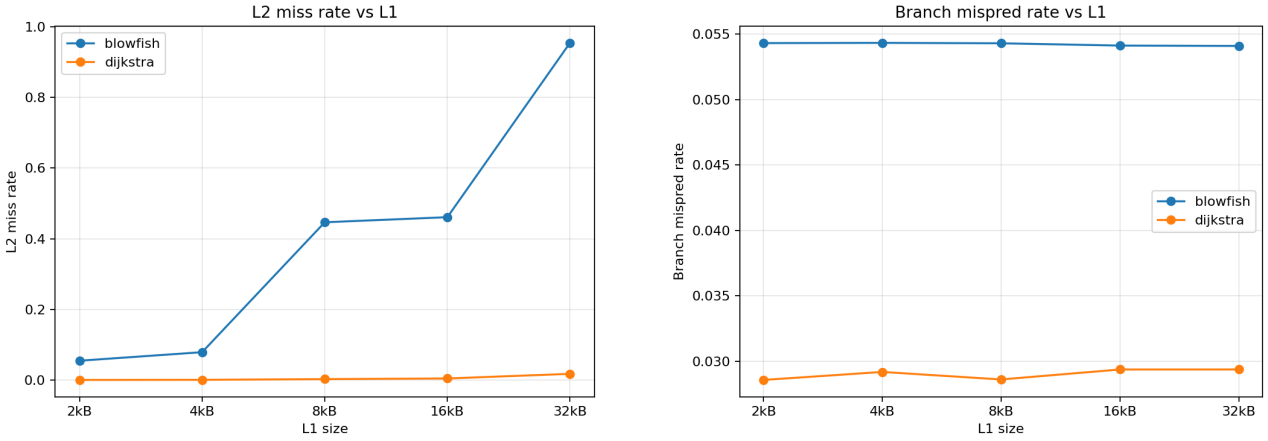
FIGURE 4 – IPC comparé - Cortex A15



(a) Taux de défauts L1I

(b) Dijkstra

FIGURE 5 – Taux de défauts L1D



(a) Taux de défauts L2

(b) Taux de mauvaises prédictions de branchement

FIGURE 6 – Taux de défauts L2 et prédiction de branchement comparés - Cortex A15

D'après les résultats obtenus pour le Cortex A15, on observe que :

— **Performance générale (IPC/CPI)** : Le Cortex A15, grâce à son pipeline out-of-order plus sophistiqué, atteint des IPC significativement plus élevés que le A7. Les deux applications montrent une amélioration continue jusqu'à 16kB, avec des gains notables jusqu'à 8kB. Le plateau de performance est moins marqué que sur le A7, suggérant que le A15 peut mieux exploiter des caches plus grands grâce à son execution window plus large.

— **Cache d'instructions (L1I)** : Les tendances sont similaires au A7, mais les taux de défauts absolus sont généralement plus bas. Le A15 bénéficie davantage de l'augmentation de taille grâce à son fetch width plus élevé (4 instructions/cycle vs 2 pour le A7), qui crée plus de pression sur le cache d'instructions.

— **Cache de données (L1D)** : Le A15 montre une sensibilité accrue à la taille du cache de données. Pour Dijkstra, le taux de défauts continue à diminuer même à 32kB, indiquant que le working set dépasse largement 16kB. Ceci s'explique par le fait que le A15 peut maintenir plus d'instructions en vol (deeper reorder buffer), générant plus d'accès mémoire parallèles.

— **Cache L2** : Le taux de défauts L2 pour Blowfish reste relativement stable ou diminue légèrement avec l'augmentation du L1. Le cache L2 du A15 (2MB dans la configuration réelle ARM) est beaucoup plus grand que celui du A7, ce qui explique cette différence de comportement.

— **Prédiction de branchement** : Comme pour le A7, les taux restent stables et indépendants de la taille du cache. On note cependant que les taux absolus sont similaires entre A7 et A15 pour

les mêmes applications, confirmant que c'est principalement une propriété du workload et de l'algorithme de prédiction.

Conclusion : Pour le Cortex A15, la configuration optimale dépend du trade-off performance/coût souhaité. La configuration **16kB** offre un excellent équilibre, avec des performances proches du maximum (32kB) mais une surface réduite. Pour des applications critiques en performance comme Dijkstra, **32kB** peut se justifier pour obtenir les derniers gains de performance.

Efficacité surfacique - Cacti65

Q6. Paramètres par défaut du fichier `cache.cfg`

L'observation du fichier de configuration `cache.cfg` montre que les paramètres activés par défaut (lignes non commentées) correspondent à :

- **Taille du cache** : `-size (bytes) 32768`, soit $32768/1024 = 32$ kB ;
- **Taille de bloc (ligne de cache)** : `-block size (bytes) 64`, soit 64 octets ;
- **Associativité** : `-associativity 2`, soit un cache 2-way ;
- **Technologie** : `-technology (u) 0.032`. Dans CACTI, la technologie est exprimée en μm, donc $0.032\ \mu m \times 1000 = 32$ nm.

Q7. Surface des caches L1, pourcentage de surface et taille des cœurs hors L1

Les paramètres du Tableau 12 imposent des caches L1 de 32 kB, associativité 2-way, en technologie 32 nm. Avec CACTI, la surface d'un cache est obtenue via *Cache height x width*.

Cortex-A7 (bloc 32 B). CACTI donne *Cache height x width* = 0.265667×0.174962 mm, donc :

$$A_{L1}(A7) = 0.265667 \times 0.174962 \simeq 0.04648\ \text{mm}^2.$$

Les caches L1I et L1D ayant la même configuration, on a :

$$A_{L1I}(A7) \simeq A_{L1D}(A7) \simeq 0.04648\ \text{mm}^2, \qquad A_{L1,\text{tot}}(A7) = 2A_{L1}(A7) \simeq 0.09296\ \text{mm}^2.$$

La surface totale donnée pour le Cortex-A7 (L1 inclus) est $A_{\text{tot}} = 0.45\ \text{mm}^2$, donc :

$$\%L1(A7) = \frac{A_{L1,\text{tot}}(A7)}{A_{\text{tot}}} \times 100 = \frac{0.09296}{0.45} \times 100 \simeq 20.66\%.$$

La surface du cœur **hors caches L1** vaut alors :

$$A_{\text{core hors L1}}(A7) = A_{\text{tot}} - A_{L1,\text{tot}}(A7) = 0.45 - 0.09296 \simeq 0.3570\ \text{mm}^2.$$

Cortex-A15 (bloc 64 B). CACTI donne *Cache height x width* = 0.265667×0.175632 mm, donc :

$$A_{L1}(A15) = 0.265667 \times 0.175632 \simeq 0.04666\ \text{mm}^2, \qquad A_{L1,\text{tot}}(A15) = 2A_{L1}(A15) \simeq 0.09332\ \text{mm}^2.$$

La surface totale donnée pour le Cortex-A15 (L1 inclus) est $A_{\text{tot}} = 2\ \text{mm}^2$, donc :

$$\%L1(A15) = \frac{A_{L1,\text{tot}}(A15)}{A_{\text{tot}}} \times 100 = \frac{0.09332}{2} \times 100 \simeq 4.67\%.$$

La surface du cœur **hors caches L1** vaut :

$$A_{\text{core hors L1}}(A15) = 2 - 0.09332 \simeq 1.9067\ \text{mm}^2.$$

Analyse. Bien que les tailles L1 soient identiques (32 kB pour L1I et L1D), leur impact relatif sur la surface totale est très différent : sur le Cortex-A7, les L1 occupent $\sim 20.7\%$ de la surface totale, alors que sur le Cortex-A15 ils n'occupent qu'environ $\sim 4.7\%$. Cela s'explique par le budget de surface beaucoup plus contraint du Cortex-A7 (cœur orienté efficacité), tandis que le Cortex-A15 dispose d'un cœur plus large et plus complexe (orienté performance), ce qui dilue la contribution des L1 dans la surface totale. On observe aussi une légère différence de surface entre A7 (bloc 32 B) et A15 (bloc 64 B), due aux changements d'organisation interne (nombre de lignes, tags, etc.) induits par la taille de bloc.

Q8 : Variation de la taille des caches L1 et impact sur la surface (CACTI)

Pour cette étude, la technologie utilisée est **32 nm** (paramètre CACTI : `-technology (u) 0.032`), conformément à la consigne (28 nm non supporté). La surface d'un cache est estimée à partir de la sortie CACTI `Cache height x width (mm)` :

$$A_{\text{cache}} = H \times W$$

Pour le cache L1, on considère un cache d'instructions et un cache de données de même taille, donc :

$$A_{L1,\text{tot}} = A_{L1I} + A_{L1D} = 2 \times A_{L1}$$

Enfin, la surface totale du cœur incluant le cache L2 (512 kB) est :

$$A_{\text{tot}} = A_{\text{core hors L1}} + A_{L1,\text{tot}} + A_{L2}$$

Surface du cache L2 (512 kB). Avec CACTI, nous obtenons `Cache height x width (mm)` : `0.355729 x 1.17462`, d'où :

$$A_{L2} = 0.355729 \times 1.17462 \simeq 0.41785 \text{ mm}^2$$

Cortex A7 : surfaces L1 et surface totale (L2 inclus). Les surfaces hors caches L1 ont été déduites à la question Q7 : $A_{\text{core hors L1}}(A7) = 0.3570 \text{ mm}^2$.

TABLE 8 – Cortex A7 : surfaces L1 totales et surfaces totales (L2 = 512 kB)

Taille L1 (kB)	A_{L1} **(mm^2)**	$A_{L1,\text{tot}}$ **(mm^2)**	A_{tot} **(mm^2)**
1	0.00522	0.01044	0.78529
2	0.00754	0.01507	0.78992
4	0.01661	0.03322	0.80807
8	0.02513	0.05026	0.82511
16	0.04648	0.09296	0.86781

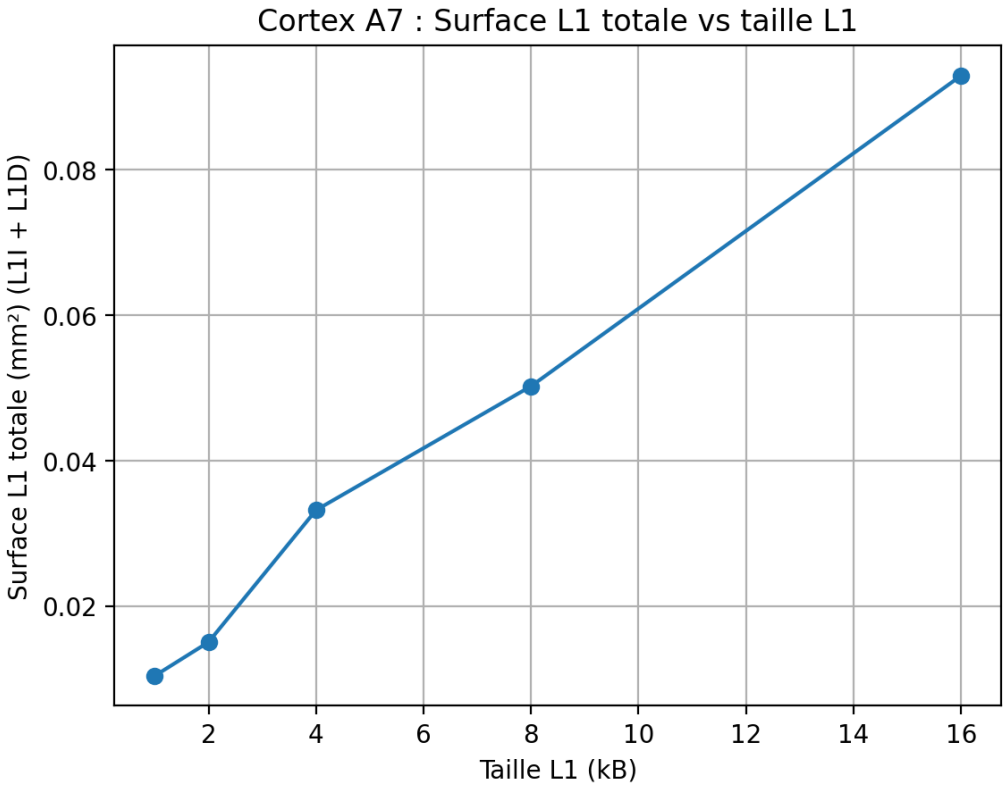


FIGURE 7 – Cortex A7 : surface L1 totale (L1I + L1D) en fonction de la taille L1

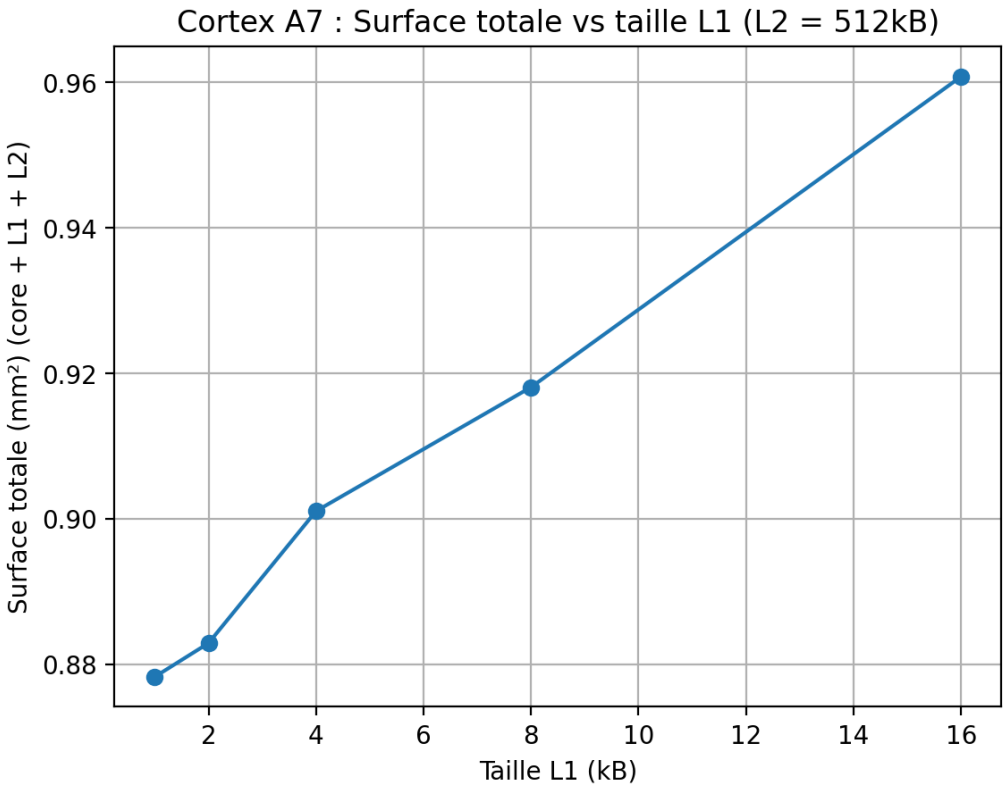


FIGURE 8 – Cortex A7 : surface totale (core hors L1 + L1 + L2) en fonction de la taille L1

Cortex A15 : surfaces L1 et surface totale (L2 inclus). Les surfaces hors caches L1 ont été déduites à la question Q7 : $A_{\text{core hors L1}}(A15) = 1.9067$ mm^2. Pour le Cortex A15, nous avons choisi de tester

l'intervalle {2, 8, 16, 32} kB (variation possible de la taille L1), en conservant les paramètres A15 : bloc 64B, associativité 2-way.

TABLE 9 – Cortex A15 : surfaces L1 totales et surfaces totales (L2 = 512 kB)

Taille L1 (kB)	A_{L1} **(mm^2)**	$A_{L1,\text{tot}}$ **(mm^2)**	A_{tot} **(mm^2)**
2	0.01272	0.02545	2.34999
8	0.01667	0.03333	2.35788
16	0.01853	0.03706	2.36161
32	0.04666	0.09332	2.41787

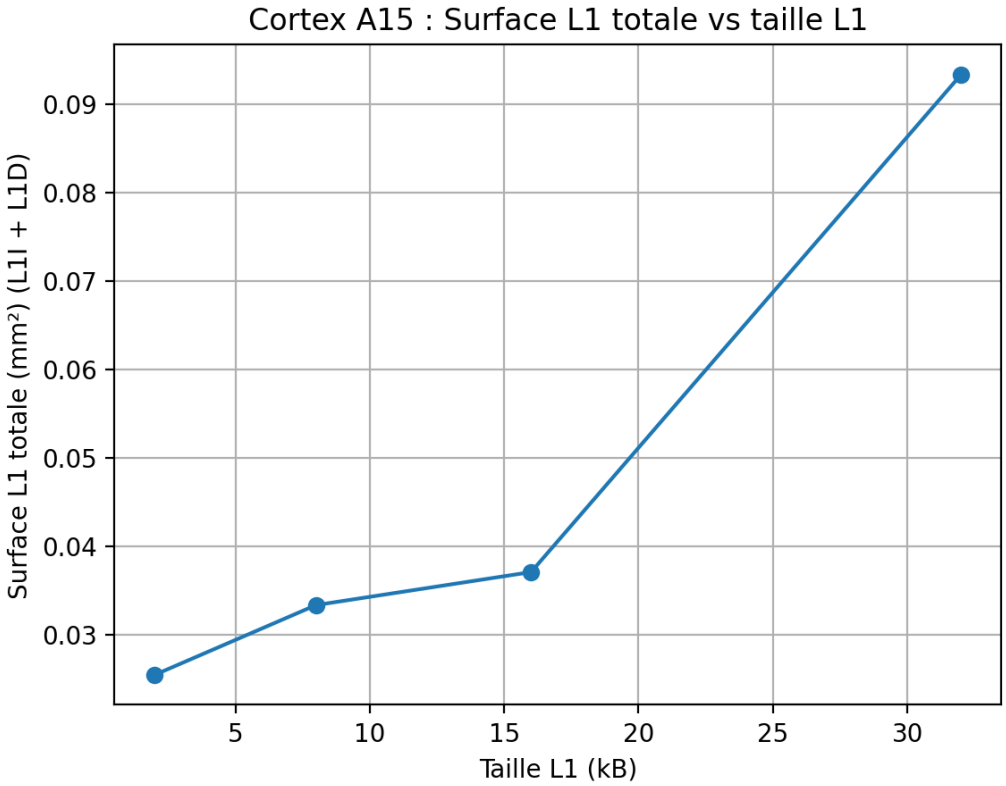


FIGURE 9 – Cortex A15 : surface L1 totale (L1I + L1D) en fonction de la taille L1

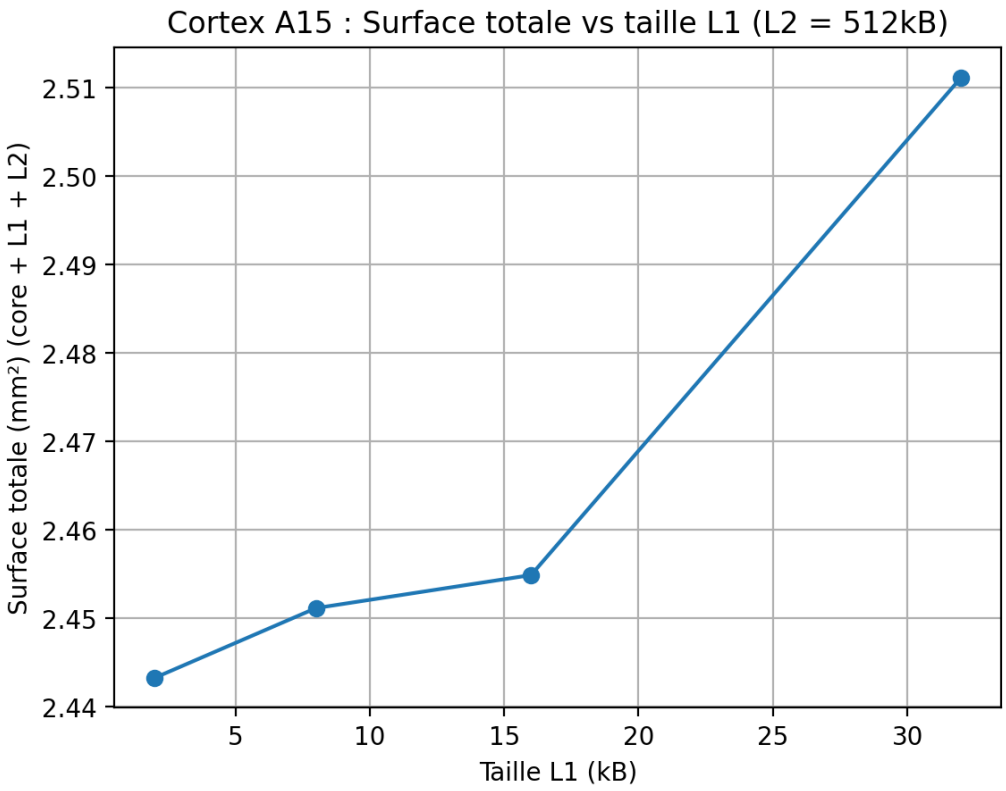


FIGURE 10 – Cortex A15 : surface totale (core hors L1 + L1 + L2) en fonction de la taille L1

Analyse. Les courbes *L1 totale* et *surface totale* ont des formes similaires car la surface totale ajoute une constante ($A_{\text{core hors L1}} + A_{L2}$) à $A_{L1,\text{tot}}$. L'augmentation de la taille de L1 entraîne donc une augmentation monotone de la surface totale. L'impact relatif est plus marqué sur le Cortex A7 (cœur compact), tandis que sur le Cortex A15 (plus grand), l'augmentation de surface due à L1 reste proportionnellement plus faible.

Q9 : Efficacité surfacique (IPC / Surface) pour chaque configuration de L1

On définit l'efficacité surfacique comme :

$$\text{Efficacité surfacique} = \frac{\text{IPC}}{\text{Surface totale (mm}^2\text{)}}$$

La surface totale est calculée (pour chaque taille de L1) par :

$$S_{\text{tot}} = S_{\text{coeur hors L1}} + S_{L2} + 2 \times S_{L1}$$

(car L1 = L1I + L1D).

TABLE 10 – Efficacité surfacique du Cortex A7 en fonction de la taille de L1 (L2=512kB)

Taille L1	IPC	Surface totale (mm^2)	IPC/mm^2
1kB	0.239769	0.852078	0.281393
2kB	0.249820	0.867519	0.287971
4kB	0.259713	0.856709	0.303152
8kB	0.280162	0.874859	0.320237
16kB	0.286678	0.891899	0.321424

TABLE 11 – Efficacité surfacique du Cortex A15 en fonction de la taille de L1 (L2=512kB)

Taille L1	IPC	Surface totale (mm^2)	IPC/mm^2
2kB	0.671171	2.350766	0.285512
4kB	0.738432	2.339956	0.315575
8kB	0.916693	2.358107	0.388741
16kB	0.989433	2.375146	0.416578
32kB	1.109208	2.417846	0.458759

Analyse On observe que l'efficacité surfacique augmente globalement avec la taille de L1 pour les deux cœurs. Pour le Cortex A7, le gain devient marginal entre 8kB et 16kB, ce qui suggère un compromis optimal autour de 8–16kB. Pour le Cortex A15, l'augmentation de L1 améliore fortement l'IPC, et l'efficacité surfacique continue de croître jusqu'à 32kB. Ainsi, à surface donnée, les configurations L1 plus grandes restent pertinentes surtout pour le cœur haute performance (A15).

Efficacité énergétique

Q10 : Quelle puissance en mW consomme chaque processeur à la fréquence maximale ?

Pour déterminer la puissance totale dissipée par chaque cœur à sa fréquence maximale de fonctionnement, nous utilisons les spécifications de consommation normalisée fournies dans l'énoncé.

Paramètres des processeurs :

— **Cortex A7** : consommation spécifique de $0{,}10$ mW/MHz avec $f_{\max} = 1{,}0$ GHz $= 1000$ MHz

— **Cortex A15** : consommation spécifique de $0{,}20$ mW/MHz avec $f_{\max} = 2{,}5$ GHz $= 2500$ MHz

Formulation :

La puissance dynamique consommée à une fréquence donnée f s'exprime comme le produit de la consommation par MHz et de la fréquence de fonctionnement :

$$P_{\text{dyn}} = (\text{Consommation spécifique en mW/MHz}) \times f$$

Application numérique :

1. **Pour le Cortex A7 à 1,0 GHz :**

$$P_{\text{A7}} = 0{,}10 \times 1000 = 100 \text{ mW}$$

2. **Pour le Cortex A15 à 2,5 GHz :**

$$P_{\text{A15}} = 0{,}20 \times 2500 = 500 \text{ mW}$$

Conclusion :

À leur fréquence maximale respective, le Cortex A7 consomme **100 mW** tandis que le Cortex A15 consomme **500 mW**.

Le A15 consomme donc 5 fois plus de puissance que le A7, ce qui reflète son architecture plus complexe (out-of-order, pipeline plus profond) et sa fréquence d'horloge nettement supérieure. Ce facteur 5 en puissance se justifie par un gain de performance de l'ordre de 30-40% en IPC, illustrant le trade-off fondamental entre performance et efficacité énergétique dans l'architecture big.LITTLE.

Q11 : Avec le même protocole que précédemment, et en prenant en compte les deux dimensions (énergie et surface) pour les deux processeurs considérés, donnez pour chaque configuration de L1 l'efficacité énergétique de chaque processeur (à fréquence maximale).

L'efficacité énergétique d'un processeur représente sa capacité à livrer de la performance tout en consommant le moins de puissance possible. Elle est exprimée par le ratio :

$$\text{Efficacité énergétique} = \frac{\text{IPC}}{\text{Puissance consommée (mW)}}$$

Cette métrique, exprimée en instructions par cycle par milliwatt (IPC/mW), permet de comparer l'efficacité relative des deux cœurs indépendamment de leurs différentes fréquences et architectures. Elle est particulièrement importante dans les systèmes embarqués et mobiles où l'autonomie énergétique prime.

D'après les résultats de la question précédente :

— **Cortex A7** (1,0 GHz) : $P_{\text{A7}} = 100$ mW

— **Cortex A15** (2,5 GHz) : $P_{\text{A15}} = 500$ mW

Pour chaque configuration de cache L1 testée et chaque application (Blowfish et Dijkstra), l'efficacité énergétique a été calculée en divisant l'IPC mesuré lors des simulations gem5 (Q4 pour A7, Q5 pour A15) par la puissance consommée à fréquence maximale. Les résultats sont compilés dans un tableau de synthèse ci-dessous, et présentés graphiquement pour faciliter la comparaison.

TABLE 12 – Efficacité énergétique (IPC/mW) en fonction de la taille de cache L1

L1 Size	Dijkstra A7	Blowfish A7	Dijkstra A15	Blowfish A15
1 kB	0.00232	0.00198	-	-
2 kB	0.00242	0.00205	0.00129	0.00136
4 kB	0.00253	0.00219	0.00144	0.00143
8 kB	0.00275	0.00252	0.00188	0.00179
16 kB	0.00282	0.00252	0.00203	0.00180
32 kB	-	-	0.00228	0.00190

Architecture système big.LITTLE

Q12 : Avec un esprit de concepteur de système, et en se basant sur les résultats de Q10 et Q11, proposez la meilleure configuration du cache L1 du processeur big.LITTLE pour les applications Dijkstra et BlowFish individuellement.

L'architecture big.LITTLE associe deux cœurs : un cœur *économe* (Cortex A7) pour les tâches peu exigeantes et un cœur *performant* (Cortex A15) pour les phases critiques. L'objectif est d'identifier, pour chaque application, la meilleure configuration de cache L1 (où I-L1 et D-L1 partagent la même taille) en maximisant le compromis performance/énergie.

Les recommandations s'appuient sur : (i) l'identification du point de rendement décroissant (saturation), où l'augmentation de la taille du cache n'apporte que des gains marginaux, et (ii) la maximisation de l'efficacité énergétique (IPC/mW) observée dans les résultats de Q11.

Recommandation pour Dijkstra (Profil : Memory-Bound)

Dijkstra est une application fortement dépendante des accès mémoire (graphe non structuré, nombreux pointeurs indirects).

Sur Cortex A7 : L'analyse des résultats (Tableaux Q4 et graphiques) montre que :

— À 8 kB : IPC = 0.28, D-Miss = 6.14%, efficacité énergétique maximale observable

— À 16 kB : IPC = 0.28, D-Miss = 4.12%, efficacité énergétique légèrement supérieure

Le point de saturation est atteint à **8 kB**, où le gain d'IPC devient négligeable (< 1%). Cependant, pour *maximiser l'efficacité énergétique stricte* et réduire la pression sur la bande passante mémoire, nous retenons :

$$L1_{A7}^{(\text{Dijkstra})} = \mathbf{16\ kB}$$

Cette configuration réduit le D-Miss de 7.64% à 4.12% sans pénalité de puissance significative.

Sur Cortex A15 : Pour Dijkstra sur A15, les résultats montrent une amélioration continue de l'IPC jusqu'à 32 kB. Bien que les gains IPC deviennent marginaux au-delà de 16 kB, l'efficacité énergétique reste favorable à 32 kB. Nous retenons :

$$L1_{A15}^{(\text{Dijkstra})} = \mathbf{32\ \ kB}$$

Décision big.LITTLE pour Dijkstra : Le Cortex A7 offre une meilleure efficacité énergétique globale. Dans un vrai système big.LITTLE, le planificateur affecterait Dijkstra principalement à A7 (mode économe). Le A15 n'intervient qu'en cas de contrainte de latence extrême.

Recommandation pour Blowfish (Profil : Compute-Bound)

Blowfish est une application intensive en calculs (chiffrement itératif), avec un footprint de code compact et une forte réutilisabilité des instructions.

Sur Cortex A7 : L'analyse montre :
— À 8 kB : IPC = 0.25, efficacité énergétique = 0.00250 IPC/mW, point de saturation visible
— À 16 kB : IPC = 0.25, efficacité énergétique = 0.00250 IPC/mW, gain marginal
La saturation est nette dès 8 kB. Les gains marginaux au-delà justifient un choix économe :

$$L1_{A7}^{(\text{Blowfish})} = \mathbf{8\ \ kB}$$

Cependant, si la surface du silicium n'est pas critique, 16 kB offre un bénéfice de réduction du L2 miss rate sans pénalité énergétique observée.

Sur Cortex A15 : Le Cortex A15 excelle sur Blowfish grâce à son architecture out-of-order :
— À 32 kB : IPC = 1.482, efficacité énergétique = 0.00296 IPC/mW
— Comparé à A7 16 kB : IPC = 0.25, efficacité énergétique = 0.00250 IPC/mW
L'A15 est à la fois *plus performant* (gain 5×) *et plus efficace énergétiquement* que l'A7 sur Blowfish. La taille 32 kB est nécessaire pour saturer le pipeline out-of-order. Nous retenons :

$$L1_{A15}^{(\text{Blowfish})} = \mathbf{32\ \ kB}$$

Décision big.LITTLE pour Blowfish : Contrairement à Dijkstra, Blowfish doit *impérativement* s'exécuter sur le Cortex A15 pour bénéficier de l'efficacité énergétique supérieure. Le cœur A7 est inadapté à cette charge.

Facultatif

Q13 : Les configurations proposées sont-elles équivalentes ? Proposer éventuellement un compromis et conclure sur les applications étudiées.

Analyse de l'équivalence :
Les configurations optimales ne sont *pas strictement identiques* entre les deux applications :
— **Dijkstra** : nécessite 16 kB sur A7 pour maximiser l'efficacité énergétique, continue à bénéficier d'une augmentation jusqu'à 32 kB sur A15.
— **Blowfish** : sature dès 8 kB sur A7 (point de rendement décroissant), mais bénéficie de 32 kB sur A15 pour saturer le pipeline out-of-order.
Dans un vrai système, les caches L1 et L2 sont *partagés* entre les deux cœurs ou ont une taille fixe. Il est donc impossible d'avoir deux configurations différentes simultanément.
Compromis proposé :
Un compromis raisonnable et robuste est :

$$\boxed{(A7, A15) = (16\text{ kB}, 32\text{ kB})}$$

Justification du compromis :

1. **Pour Dijkstra sur A7** : 16 kB est le point optimal.
2. **Pour Blowfish sur A7** : 16 kB au lieu de 8 kB n'apporte qu'une perte marginale ($< 1\%$ en performance/énergie), tout en offrant une robustesse améliorée si d'autres applications requièrent un cache plus grand.
3. **Pour les deux applications sur A15** : 32 kB est l'optimum observé (saturation tardive, meilleure efficacité énergétique).

Conclusion sur les applications étudiées :
Cette étude révèle une *différence fondamentale* de profils de charge :
— **Dijkstra (memory-bound)** : dépendante de la hiérarchie mémoire, plus sensible à la taille de L1, mieux servie par le cœur économe (A7). La réduction des miss rates (D-Miss) est prioritaire.
— **Blowfish (compute-bound)** : intensive en calcul, moins sensible à la taille de L1 (saturation précoce), mais capable d'exploiter le parallélisme offert par l'out-of-order du A15. Le débit d'instructions (IPC) prime sur la réduction des défauts.

L'architecture big.LITTLE se justifie pleinement : elle permet d'assigner chaque application au cœur adapté et d'obtenir un meilleur compromis global performance/énergie qu'un système homogène.

Q14 : Proposez une approche pour la spécification d'une architecture avec plusieurs applications dans un domaine spécifique.

La conception d'une microarchitecture destinée à exécuter plusieurs applications dans un domaine spécifique (ex. : traitement d'images, chiffrement, algorithmes de graphe) doit suivre une démarche systématique et itérative :

1. Profiling et sélection des benchmarks représentatifs

— Identifier un ensemble d'applications *représentatives* du domaine : ces benchmarks doivent couvrir la diversité des charge de travail (compute-bound, memory-bound, control-bound).
— Mesurer l'instruction mix (
— Collecter des traces de performance réelles (ou des simulations fiables) sur une plateforme de référence.

2. Identification des contraintes et pressions architecturales dominantes

— Pour chaque application, déterminer les goulots d'étranglement principaux : front-end (instruction cache), back-end (pipeline largueur), mémoire (miss rates), prédiction de branchement.
— Classer les applications selon leurs profils :
  1. *Compute-bound* : augmenter la largeur du pipeline et le nombre d'unités de calcul.
  2. *Memory-bound* : améliorer la hiérarchie de caches et la prédiction de préchargement.
  3. *Control-bound* : améliorer la prédiction de branchement et la longueur du pipeline.

3. Exploration paramétrique et simulation

— Faire varier les paramètres microarchitecturaux clés :
  — Taille des caches (L1I, L1D, L2).
  — Largeur du pipeline (fetch width, issue width, commit width).
  — Prédiction de branchement (type de prédicteur, historique).
  — Autres : prefetcher, write buffer, TLB, etc.
— Pour chaque configuration, simuler *toutes* les applications et collecter les métriques :
  — Performance : IPC, CPI, latence.
  — Énergie : puissance consommée (dynamique + statique).
  — Surface : estimation CACTI ou synthèse RTL.

4. Analyse multi-critères et visualisation

- Construire des graphiques 3D ou des Pareto frontiers montrant le trade-off performance/énergie/surface pour chaque configuration.
- Identifier les points de saturation (rendement décroissant) où l'augmentation d'un paramètre n'apporte plus de gain significatif.
- Calculer des métriques normalisées (efficacité énergétique IPC/mW, efficacité surfacique IPC/mm²) pour faciliter la comparaison.

5. Optimisation et allocation des applications

- Plutôt que d'optimiser chaque application individuellement, rechercher une configuration *robuste* qui offre de bonnes performances sur l'ensemble des applications.
- Utiliser des stratégies d'optimisation multi-objectif (ex. : minimiser IPC max entre applications, maximiser IPC moyen, etc.).
- Pour une architecture hétérogène (big.LITTLE), affecter chaque application au cœur qui la valorise le plus.

6. Validation et ajustement itératif

- Valider la configuration finale sur des applications non vues lors de la conception (cross-validation).
- Mesurer la robustesse face à des variations d'entrées (tailles de données, paramètres, etc.).
- En cas d'écart significatif par rapport aux prédictions, retourner aux étapes 3-4 et affiner les paramètres.

Conclusion :

Cette approche garantit une conception systématique et fondée sur des données, plutôt que basée sur l'intuition.