Mémoires caches - Evaluation des performances de différentes configurations de mémoires caches

1st Luiz Gariglio Dos Santos
Ingénieur Degree Programme in STIC
ENSTA Paris
Paris, France
email@ensta-paris.fr

2nd Helena Guachalla De Andrade
Ingénieur Degree Programme in STIC
ENSTA Paris
Paris, France
email@ensta-paris.fr

3rd Santiago Florido Gomez
Ingénieur Degree Programme in STIC
ENSTA Paris
Paris, France
santiago.florido@ensta-paris.fr

4th Franck Ulrich Kenfack Noumedem
Ingénieur Degree Programme in STIC
ENSTA Paris
Paris, France
email@ensta-paris.fr

***Abstract*—mamamammsmamammasaalsnoihrf eirfrf**
***Index Terms*—hsbuahbdwefwcwcbdwuguwcc**

I. Introduction

ahdubwyfwi ceuce hyec

II. Evaluation des performances de différentes configurations de mémoires caches pour 4 algorithmes de multiplication de matrices

A. Paramètres de configuration des caches utilisés dans gem5

Configuration	Instruction cache	Data cache	L2 cache	Block size
C1	4KB direct-mapped	4KB direct-mapped	32KB direct-mapped	32 bytes
C2	4KB direct-mapped	4KB 2-way set-asso	32KB 4-way set-asso	32 bytes

TABLE I

Les paramètres de configuration des caches dans le simulateur gem5 suivent le format suivant : `tag:n_lignes:taille_bloc:associativite:politique`.

Pour chaque cache, le nombre de lignes est obtenu en divisant la taille totale du cache par la taille d'un bloc. L'associativité correspond au nombre de voies, fixé à 1 pour un cache direct-mapped, 2 pour un cache 2-way set-associative, etc. Enfin, la politique de remplacement utilisée ici est LRU, notée `l`.

Ainsi, pour la configuration C1, par exemple, le cache d'instructions possède $\frac{4096}{32} = 128$ lignes, d'où `il1:128:32:1:l`. En utilisant la même logique pour les autres caches et configurations, le tableau suivant est rempli.

Configuration	IL1	DL1	UL2
C1	il1:128:32:1:l	dl1:128:32:1:l	ul2:1024:32:1:l
C2	il1:128:32:1:l	dl1:64:32:2:l	ul2:256:32:4:l

B. Taux de défauts dans les différentes caches

- Le taux de défauts dans le cache d'instructions il1 : `icache.overallMissRate`
- Le taux de défauts dans le cache de données dl1 : `dcache.overallMissRate`
- Le taux de défauts dans le cache unifié (L2) ul2 : `l2cache.overallMissRate`

Programmes	Configuration de caches	
	C1	C2
P1 normale	$0,00\%$	$0,00\%$
P2 (pointeur)	$0,00\%$	$0,00\%$
P3 (tempo)	$0,00\%$	$0,00\%$
P4 (unrol)	$0,00\%$	$0,00\%$

TABLE II: icache.overallMissRate

Programmes	Configuration de caches	
	C1	C2
P1 normale	$30,08\%$	$31,01\%$
P2 (pointeur)	$30,22\%$	$31,12\%$
P3 (tempo)	$30,23\%$	$31,12\%$
P4 (unrol)	$30,06\%$	$31,10\%$

TABLE III: dcache.overallMissRate

Programmes	Configuration de caches	
	C1	C2
P1 normale	$43,85\%$	$42,20\%$
P2 (pointeur)	$43,63\%$	$42,21\%$
P3 (tempo)	$43,62\%$	$42,20\%$
P4 (unrol)	$43,65\%$	$42,02\%$

TABLE IV: l2cache.overallMissRate

Nous pouvons vérifier que les quatre algorithmes présentent une bonne localité de référence pour le code. En effet, le taux de défaut du cache d'instructions est nul ($\approx 0,00\%$) pour tous les programmes et pour les deux configurations de caches (Table II). Cela signifie que le flux d'instructions tient entièrement dans le cache d'instructions et que les boucles de

multiplication de matrices réutilisent toujours le même petit bloc de code, ce qui exploite très bien la localité spatiale et temporelle du code.

III. Profiling de l'application

Pour procéder à l'évaluation des configurations de cache pour chacun des algorithmes proposés et analyser leurs performances, il est proposé de réaliser un *profiling* de l'application à l'aide du simulateur gem5. Le *profiling* est essentiel, car il permet de quantifier des éléments du comportement du programme afin de prendre des décisions d'optimisation et de microconception architecturale fondées sur des données, principalement issues de la simulation [1]. Il permet également d'identifier des *hotspots* sur lesquels concentrer la conception et l'optimisation, c'est-à-dire de cibler en priorité les composantes qui contribuent le plus au temps d'exécution. Enfin, il fournit une première approximation de la caractérisation de la charge de travail (*workload*) d'un programme, ce qui facilite l'orientation des choix de conception.

Classe	Dijkstra large (A7)	Dijkstra large (A15)
Lecture (Load)	45 516 963 [28.4]	45 905 506 [28.5]
Écriture (Store)	19 439 553 [12.1]	19 593 718 [12.1]
Branchement	43 904 570 [21.5]	44 122 872 [21.5]
Calcul entier (Int)	95 334 242 [59.5]	95 780 142 [59.4]
Calcul flottant (Fp)	0 [0.0]	0 [0.0]
Total d'instructions exécutées	**204 195 328**	**205 402 238**

TABLE V: Dijkstra large (Cortex-A7 vs Cortex-A15).

Classe	Dijkstra small (A7)	Dijkstra small (A15)
Lecture (Load)	10 313 882 [28.5]	10 474 419 [28.5]
Écriture (Store)	4 759 916 [13.2]	4 850 175 [13.2]
Branchement	9 823 729 [21.4]	9 978 854 [21.4]
Calcul entier (Int)	21 106 947 [58.3]	21 363 899 [58.2]
Calcul flottant (Fp)	0 [0.0]	0 [0.0]
Total d'instructions exécutées	**46 004 474**	**46 667 347**

TABLE VI: Dijkstra small (Cortex-A7 vs Cortex-A15).

Classe	Blowfish (A7)	Blowfish (A15)
Lecture (Load)	19 141 [21.8]	19 769 [21.3]
Écriture (Store)	5 516 [6.3]	5 671 [6.1]
Branchement	29 760 [25.3]	30 060 [24.4]
Calcul entier (Int)	63 342 [72.0]	67 560 [72.6]
Calcul flottant (Fp)	0 [0.0]	0 [0.0]
Total d'instructions exécutées	**117 759**	**123 060**

TABLE VII: Blowfish (Cortex-A7 vs Cortex-A15).

Le *profiling* indique que la classe dominante dans les deux programmes est le calcul entier ($\approx 60\,\%$ pour Dijkstra et $\approx 72\,\%$ pour Blowfish) ; ainsi, une micro-architecture disposant de davantage d'ALU entières ou d'un parallélisme accru sur l'entier peut apporter le gain le plus important. Les branchements représentent également une part notable : restructurer les boucles ou réduire les sauts peut donc limiter les pénalités de contrôle. Côté mémoire, Dijkstra est plus sensible car il combine de nombreux accès, souvent irréguliers ; des améliorations de cache et de latence ont donc un impact plus marqué. Blowfish présente généralement des accès plus séquentiels et une meilleure localité : la hiérarchie mémoire aide, mais le goulot principal demeure le *throughput* du calcul entier.

La comparaison montre que, tant pour Dijkstra/SSCA2-BCS que pour le produit de polynômes et Blowfish, le profil d'exécution est fortement dominé par les instructions de calcul : dans Dijkstra et Blowfish, le calcul entier prédomine, tandis que, pour un produit de polynômes implémenté en flottant, on observe une fraction élevée de calcul FP. Par conséquent, une amélioration architecturale transversale consiste à accroître le parallélisme d'exécution dans la classe dominante (davantage d'ALU entières pour les applications entières et davantage d'unités FP si le noyau est flottant), car cette approche cible directement la plus grande proportion d'instructions.

La principale divergence concerne le comportement mémoire. Dijkstra/SSCA2-BCS (graphes) exécute généralement de nombreux *loads/stores* et, surtout, avec des accès irréguliers (dépendants de structures chaînées ou de motifs non linéaires). Cela dégrade la localité, augmente les *miss* en cache et rend les performances très sensibles aux modifications de la hiérarchie mémoire (tailles/associativité des caches, latences, politiques, etc.). À l'inverse, le produit de polynômes et, dans une large mesure, Blowfish présentent des accès plus séquentiels et localisés, ce qui favorise les caches et réduit la pénalité mémoire ; dans ces cas, le goulot d'étranglement se déplace davantage vers le *throughput* de calcul.

Enfin, même si leur poids relatif varie, toutes ces applications peuvent bénéficier d'améliorations du contrôle de flux (branchements) : réduire les sauts via la restructuration des boucles et la simplification des conditions, ou améliorer la prédiction de branchement, diminue les pénalités de mauvaise prédiction et contribue à abaisser le CPI. En synthèse, le meilleur scénario combine (1) davantage de capacité de calcul dans la classe dominante, (2) une optimisation mémoire particulièrement critique pour les graphes, et (3) des améliorations de prédiction/codage apportant des gains additionnels de manière générale.

A. Impact de la taille des caches L1

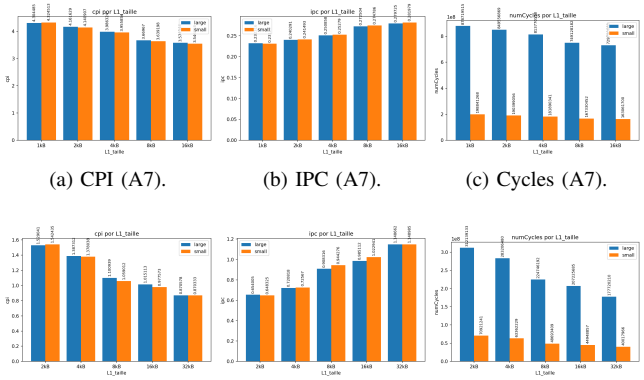
Les figures suivantes présentent l'évolution des performances en fonction de la taille du cache L1. Pour Dijkstra, les deux jeux de données (small/large) sont affichés dans le même graphique. Les résultats sont regroupés par catégorie : performance générale, hiérarchie mémoire et prédiction de branchement.

a) Performance générale (CPI, IPC, cycles).:

b) Hiérarchie mémoire (taux de défauts).:

c) Prédiction de branchement.:

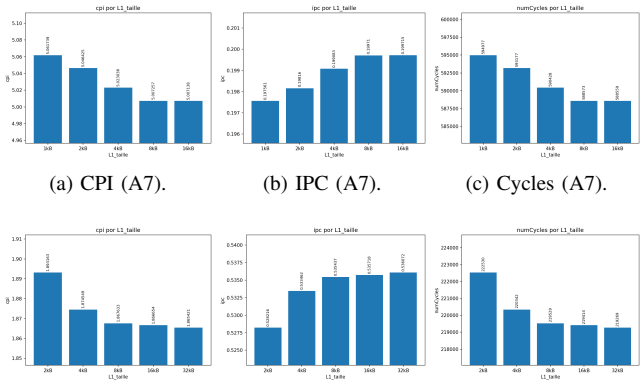
d) Analyse synthétique.: Quand la taille du L1 augmente, les taux de défauts I/D diminuent nettement, ce qui réduit le CPI et augmente l'IPC (amélioration plus marquée sur Dijkstra, plus sensible à la mémoire). Les métriques de prédiction de branchement restent globalement stables, ce qui est attendu car le prédicteur ne change pas entre configurations. Pour le Cortex-A7, la meilleure performance observée est obtenue



(a) CPI (A7). (b) IPC (A7). (c) Cycles (A7).

(d) CPI (A15). (e) IPC (A15). (f) Cycles (A15).

Fig. 1: Dijkstra : performance générale en fonction de la taille du cache L1 (A7 en haut, A15 en bas).



(a) CPI (A7). (b) IPC (A7). (c) Cycles (A7).

(d) CPI (A15). (e) IPC (A15). (f) Cycles (A15).

Fig. 2: Blowfish : performance générale en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

avec **L1 = 16 kB** pour Dijkstra (small et large) et pour Blowfish (CPI minimal et IPC maximal, avec un gain qui se stabilise entre 8 kB et 16 kB).

e) Gains relatifs (IPC et miss rates).: Les pourcentages ci-dessous sont calculés par rapport à la plus petite taille de L1 (A7 : 1 kB, A15 : 2 kB).

L1	Gain IPC (%)	Baisse I-Cache (%)	Baisse D-Cache (%)
1 kB	0,00	0,00	0,00
2 kB	4,43	18,49	25,57
4 kB	9,32	50,43	44,39
8 kB	18,83	93,89	74,00
16 kB	21,94	97,18	82,56

TABLE VIII: Dijkstra small (Cortex-A7) : gains relatifs par rapport à 1 kB.

Pour le Cortex-A7, les résultats issus des simulations montrent une progression très nette des performances lorsque la taille du cache L1 augmente, mais cette progression n'est ni linéaire ni illimitée. En prenant la plus petite taille comme référence, on observe d'abord une zone critique très marquée

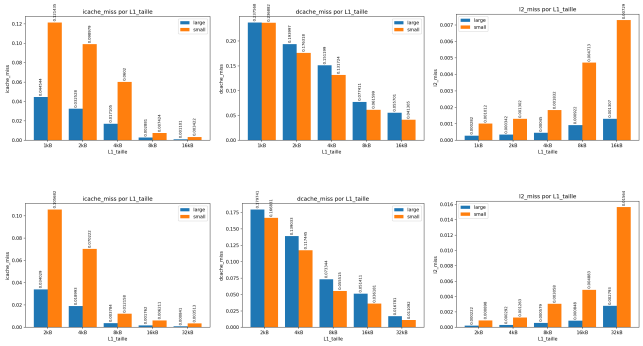


Fig. 3: Dijkstra : taux de défauts I-Cache, D-Cache et L2 en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

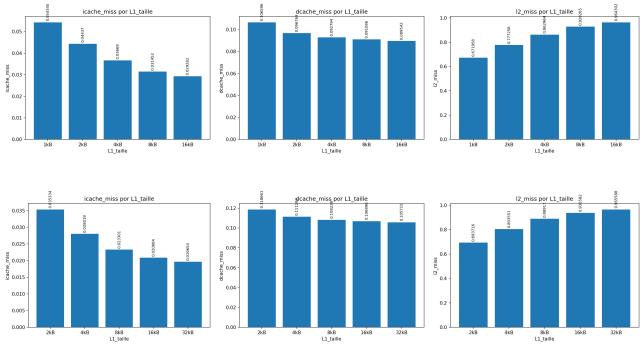


Fig. 4: Blowfish : taux de défauts I-Cache, D-Cache et L2 en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

entre 1 kB et 4 kB, où les gains d'IPC sont rapides et où les miss rates d'instruction et de données chutent fortement. Cette zone critique correspond à la transition entre un L1 trop petit pour contenir le code chaud et les structures de travail principales, et un L1 suffisamment grand pour absorber une partie significative des accès répétitifs. Le point d'inflexion apparaît ensuite autour de 8 kB : l'amélioration continue, mais la pente diminue clairement, ce qui signifie que le cache commence à capturer l'essentiel des réutilisations utiles. La saturation se constate vers 16 kB, où les gains d'IPC supplémentaires deviennent marginaux et où la réduction des défauts n'apporte plus que des améliorations modestes. Ce comportement est cohérent avec la micro-architecture A7, qui possède des buffers plus modestes et une largeur d'exécution plus réduite : lorsque le L1 est trop petit, chaque défaut se traduit par un blocage visible de la chaîne de traitement, mais dès que les structures de travail tiennent majoritairement dans le cache, les limites se déplacent vers le débit de calcul plutôt que vers la mémoire. Pour Dijkstra, la sensibilité est particulièrement forte car l'algorithme manipule des structures de graphes et des files de priorité, avec des accès irréguliers à des nœuds et à des poids d'arêtes. Ce motif introduit beaucoup de défauts de données lorsque la capacité est faible, et c'est pourquoi la diminution des miss rates D-cache est

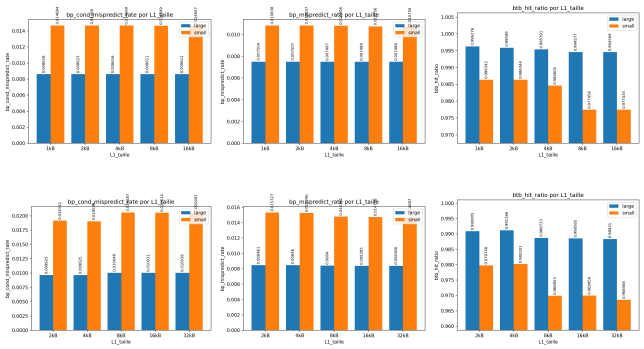


Fig. 5: Dijkstra : métriques de prédiction de branchement en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

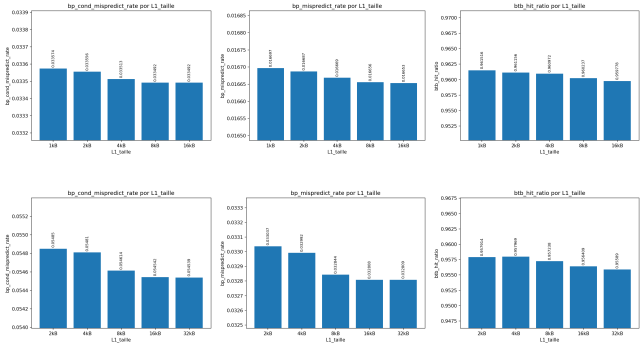


Fig. 6: Blowfish : métriques de prédiction de branchement en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

spectaculaire en passant de 1 kB à 8 kB. La baisse des miss rates I-cache est également importante, mais elle se stabilise plus tôt, car le noyau de Dijkstra repose sur un ensemble d'instructions relativement compact ; une fois ce noyau en cache, le gain est surtout lié au data working set. La distinction entre Dijkstra small et Dijkstra large renforce cette lecture : dans les deux cas, la zone critique est similaire, mais l'instance large profite davantage des augmentations initiales parce que son working set de données dépasse plus vite la capacité minimale, ce qui accentue les défauts pour les petites tailles. En revanche, au-delà de 8 kB, les courbes des deux jeux de données s'aplanissent, signe que les conflits et défauts de capacité résiduels deviennent moins dominants que le coût intrinsèque des calculs et du contrôle de flux. Il faut aussi tenir compte de la taille de ligne de 32 octets sur A7 : ce choix favorise la localité spatiale sans trop pénaliser les conflits, mais il limite la quantité de données capturées par chaque remplissage, ce qui rend les petites tailles de cache plus sensibles aux accès à pas irrégulier. L'associativité 2-ways réduit une partie des conflits, mais elle ne suffit pas à éliminer les collisions lorsque des structures chaînées ou des tables de distances sont activement parcourues ; l'augmentation de la taille agit donc à la fois sur la capacité et sur la probabilité de conflit, d'où la baisse rapide des défauts dans la zone

L1	Gain IPC (%)	Baisse I-Cache (%)	Baisse D-Cache (%)
1 kB	0,00	0,00	0,00
2 kB	3,43	26,98	18,34
4 kB	7,98	61,60	36,36
8 kB	17,30	93,53	67,42
16 kB	20,41	97,53	76,55

TABLE IX: Dijkstra large (Cortex-A7) : gains relatifs par rapport à 1 kB.

L1	Gain IPC (%)	Baisse I-Cache (%)	Baisse D-Cache (%)
1 kB	0,00	0,00	0,00
2 kB	0,30	18,35	9,22
4 kB	0,77	32,49	12,95
8 kB	1,09	42,13	14,55
16 kB	1,09	46,12	16,00

TABLE X: Blowfish (Cortex-A7) : gains relatifs par rapport à 1 kB.

critique. On observe aussi que l'amélioration de la I-cache est plus rapide que celle de la D-cache, ce qui confirme que, pour Dijkstra, la performance est principalement limitée par les données et non par l'instruction fetch, une fois le code chaud capturé.

Pour Blowfish sur A7, les résultats sont beaucoup plus modérés, ce qui est logique pour un algorithme de chiffrement symétrique qui effectue un grand nombre d'opérations arithmétiques régulières sur des blocs relativement petits. Le code est compact, les tables d'expansion et de substitution tiennent assez rapidement en cache, et les accès aux données ont une bonne localité temporelle. Ainsi, la zone critique est courte, essentiellement entre 1 kB et 4 kB, et le point d'inflexion apparaît dès 4 à 8 kB. La saturation est visible vers 8–16 kB, où l'IPC varie très peu et où les diminutions de miss rates deviennent marginales. Dans ce contexte, l'augmentation du L1 ne peut pas compenser un goulet d'étranglement qui se situe davantage dans le débit de calcul et la structure du pipeline que dans la mémoire. Les métriques de prédiction de branchement restent quasiment constantes pour toutes les tailles, ce qui est attendu puisque le prédicteur est identique et que la dynamique des branches du programme ne change pas : la taille du L1 n'influence pas le nombre de branches ni leur difficulté intrinsèque, elle ne fait que réduire les stalls de fetch ou de données. Cela explique pourquoi les gains d'IPC se traduisent surtout par la disparition de stalls dus aux misses et non par une amélioration des comportements de contrôle. Au niveau micro-architectural, le Cortex-A7, avec sa largeur d'exécution plus faible, bénéficie davantage de la réduction des latences mémoire relatives : un défaut de cache occupe une proportion plus importante du temps de cycle effectif, et chaque réduction de miss rate se reflète donc de manière visible sur l'IPC. Mais une fois que le cache est suffisamment grand, les pertes restantes proviennent de limites structurelles (largeur d'issue, profondeur des files, contraintes de bande passante interne), qui ne sont pas affectées par la taille du L1. Pour Blowfish, la taille de ligne de 32 octets est déjà suffisante pour amortir les accès séquentiels, et les données manipulées restent limitées à quelques tables et blocs d'entrée,

L1	Gain IPC (%)	Baisse I-Cache (%)	Baisse D-Cache (%)
2 kB	0,00	0,00	0,00
4 kB	11,93	33,55	29,60
8 kB	45,65	88,50	66,72
16 kB	57,78	94,12	78,31
32 kB	77,22	96,68	93,35

TABLE XI: Dijkstra small (Cortex-A15) : gains relatifs par rapport à 2 kB.

L1	Gain IPC (%)	Baisse I-Cache (%)	Baisse D-Cache (%)
2 kB	0,00	0,00	0,00
4 kB	10,22	44,19	22,65
8 kB	38,89	88,88	59,19
16 kB	50,63	94,82	71,40
32 kB	75,64	97,53	90,66

TABLE XII: Dijkstra large (Cortex-A15) : gains relatifs par rapport à 2 kB.

ce qui explique la rapide saturation observée. En résumé, pour A7, la zone critique est très claire et l'effet de saturation est rapide, ce qui suggère qu'un dimensionnement modéré du L1 est déjà suffisant pour capter l'essentiel des bénéfices sur Dijkstra et qu'au-delà de 8–16 kB, les gains deviennent coûteux par rapport à la surface.

Pour le Cortex-A15, les tendances sont similaires mais amplifiées, avec des gains plus élevés sur la performance lorsque la taille du L1 augmente. La zone critique se situe ici entre 2 kB et 8 kB : l'IPC progresse fortement et les miss rates en I-cache et D-cache chutent de manière drastique. Le point d'inflexion apparaît autour de 8–16 kB, et la saturation devient nette vers 32 kB. Cette différence par rapport au Cortex-A7 s'explique par une micro-architecture plus agressive, dotée d'une largeur d'exécution plus importante, de buffers plus profonds et d'une capacité supérieure à exploiter le parallélisme d'instruction. Dans ces conditions, la latence mémoire devient un frein majeur dès que le cache est trop petit, car le cœur a la capacité de remplir rapidement ses fenêtres d'instruction et d'exposer davantage de dépendances mémoire. Un L1 plus grand réduit ces latences apparentes et permet au cœur de maintenir un débit élevé, d'où les gains d'IPC plus importants observés pour Dijkstra. Les diminutions de miss rates suivent aussi une courbe caractéristique : la I-cache bénéficie très tôt de l'augmentation de capacité, car le code critique est rapidement capturé ; la D-cache continue à s'améliorer jusqu'à 16 kB et au-delà, car Dijkstra, avec ses accès irréguliers et son working set de données dispersé, génère de nombreux défauts de capacité et de conflit lorsque le cache est trop petit. La présence d'un bloc de 64 octets sur A15 augmente l'efficacité de la localité spatiale pour certains accès séquentiels, ce qui peut contribuer à accélérer la baisse des miss rates pour des tailles intermédiaires. Cette taille de ligne plus large peut toutefois amplifier le coût d'un miss individuel, ce qui rend la zone critique particulièrement sensible au dimensionnement du L1. L'associativité 2-ways réduit une partie des conflits, mais la densité d'accès du jeu de données large fait encore apparaître des collisions lorsque le cache est trop petit, d'où les gains importants observés entre 2 kB et 8 kB. La distinction

L1	Gain IPC (%)	Baisse I-Cache (%)	Baisse D-Cache (%)
2 kB	0,00	0,00	0,00
4 kB	0,99	20,70	6,21
8 kB	1,37	34,06	8,79
16 kB	1,42	40,90	9,91
32 kB	1,49	44,38	10,91

TABLE XIII: Blowfish (Cortex-A15) : gains relatifs par rapport à 2 kB.

entre Dijkstra small et Dijkstra large reste visible : le jeu de données large profite davantage des tailles intermédiaires, mais la convergence vers la saturation se produit dans les deux cas, indiquant que la majeure partie du working set pertinent finit par être contenue dans le L1 ou par être amortie par les mécanismes de prédiction et de préchargement implicites du pipeline.

Pour Blowfish sur A15, les gains restent modestes, bien que légèrement supérieurs à ceux observés sur A7. La zone critique est surtout entre 2 kB et 4 kB, avec un point d'inflexion vers 8 kB, puis une saturation marquée entre 16 et 32 kB. Cela s'explique par la nature essentiellement calculatoire de l'algorithme et par la régularité de ses accès, qui rendent les défauts de cache moins fréquents une fois une petite capacité disponible. Le cœur A15, plus performant en calcul, peut absorber davantage d'instructions par cycle, mais il n'y a pas suffisamment de pression mémoire pour justifier un L1 très grand ; l'amélioration supplémentaire apportée par la taille se traduit donc par des gains d'IPC très limités. Les mesures de prédiction de branchement restent stables, comme pour A7, ce qui confirme que les variations de performance proviennent principalement des effets de capacité mémoire et non d'un changement de comportement de contrôle. En pratique, l'augmentation de L1 au-delà de 8–16 kB pour Blowfish n'apporte qu'un rendement décroissant, ce qui est cohérent avec un code compact et un jeu de données qui tient rapidement en cache. Dans un contexte micro-architectural plus large, il faut aussi considérer que le A15 dispose de plus de ressources pour masquer certaines latences, par exemple grâce à une fenêtre d'instruction plus grande et à des files de chargement plus profondes. Cela signifie que la baisse des miss rates se traduit plus directement en IPC lorsque le cache est petit, mais que l'effet marginal diminue quand les autres ressources deviennent le facteur limitant. En synthèse, le Cortex-A15 bénéficie fortement d'un L1 suffisamment dimensionné pour Dijkstra, où la zone critique et le point d'inflexion montrent que la mémoire est le facteur limitant, alors que pour Blowfish, le facteur dominant reste le calcul, et l'augmentation de L1 a un impact limité. Cette lecture met en évidence une logique d'optimisation différenciée : un L1 plus grand est pertinent lorsque la charge de travail présente des accès irréguliers et une faible localité de données, tandis qu'une charge plus régulière et calculatoire atteint rapidement la saturation, rendant l'augmentation de capacité moins justifiable en termes de surface et d'énergie. Enfin, la cohérence des tendances sur A15 indique que le comportement est robuste à la variation de taille et qu'il est possible d'identifier un

compromis autour de 16 kB qui offre un bon équilibre entre gain de performance et coût en surface, même si la saturation finale se situe vers 32 kB. On peut également interpréter ces tendances au regard de la hiérarchie mémoire globale : lorsque le L1 est petit, la D-cache délègue plus souvent vers le L2, et même si le L2 présente des taux de défauts faibles, la simple latence d'accès suffit à dégrader l'IPC sur un cœur aussi large. À l'inverse, quand le L1 grossit, la pression sur le L2 diminue, ce qui stabilise le temps de service et rend les files de chargement moins congestionnées. Cette stabilisation réduit aussi les variations d'IPC entre exécutions, signe que la phase de calcul redevient dominante. Du point de vue de l'efficacité surfacique, ces gains ont un coût : un L1 plus grand augmente la surface et la consommation, et l'intérêt marginal de passer de 16 kB à 32 kB est relativement faible pour Blowfish et modéré pour Dijkstra. Ainsi, si l'objectif est un compromis performance-surface, on peut justifier un L1 intermédiaire, tandis que si l'objectif est de maximiser la performance absolue sur des charges mémoire, la taille maximale reste préférable malgré la saturation progressive.

B. *Efficacité surfacique*

Observant le fichier `cache.cfg`, on constate que la capacité totale des données pouvant être stockées dans le cache, sans compter les métadonnées telles que les *tags*, identifiée comme la taille du cache, est de 131072 bytes, soit 128 KiB. De son côté, l'unité minimale chargée depuis la mémoire ou depuis le niveau `L2` vers le cache correspond à la taille de bloc, qui est de 64 bytes. La configuration standard comporte 2 voies pour le placement des blocs à l'intérieur des ensembles (*sets*). Enfin, la technologie utilisée par défaut est de 0,090 μm, soit 90 nm.

Les surfaces proviennent des sorties CACTI (`cacti/result_L1_*`) via la ligne `Cache height x width (mm)`. En supposant les tailles du Tableau 12 : L1I = L1D = 16 kB (A7) et 32 kB (A15), on obtient :

Cœur	S_{L1I} (mm^2)	S_{L1D} (mm^2)	$S_{L1} = S_{L1I} + S_{L1D}$ (mm^2)
A7 (16 kB)	0,03774	0,03774	0,07548
A15 (32 kB)	0,03319	0,03319	0,06639

Avec $S_{\text{core+L1}}(A7) = 0{,}45$ mm^2 et $S_{\text{core+L1}}(A15) = 2$ mm^2 :

$$\%L1(A7) = \frac{0{,}07548}{0{,}45} \times 100 = 16{,}77\%, \qquad \%L1(A15) = \frac{0{,}06639}{2} \times 100 = 3{,}32\%.$$

$$S_{\text{core hors L1}}(A7) = 0{,}45 - 0{,}07548 = 0{,}37452 \text{ mm}^2, \quad S_{\text{core hors L1}}(A15) = 2 - 0{,}06639 = 1{,}93361 \text{ mm}^2.$$

Analyse. Le A7 consacre une fraction beaucoup plus importante de sa surface au L1 ($\approx 16{,}8\%$) que le A15 ($\approx 3{,}3\%$), ce qui reflète un cœur plus compact : à taille de L1 comparable, le poids surfacique est plus élevé sur A7.

a) Q8.: Les surfaces L1 (I et D) sont celles de CACTI (`cacti/result_L1_*`). Le L2 fixé à 512 kB donne, avec les paramètres gem5 (A7: 32 B, 8-way ; A15: 64 B, 16-way),

$$S_{L2}(A7) = 0{,}94241 \text{ mm}^2, \qquad S_{L2}(A15) = 0{,}94019 \text{ mm}^2.$$

Pour chaque taille L1, on utilise :

$$S_{\text{core+L1+L2}} = S_{\text{core hors L1}} + 2S_{L1} + S_{L2}.$$

L1 (kB)	S_{L1} (mm^2)	S_{L1}^{tot} (mm^2)	$S_{\text{core+L1+L2}}$ (mm^2)
Cortex-A7 (L2=512 kB, 32 B, 8-way)			
1	–	–	–
2	0,01719	0,03438	1,35130
4	0,02095	0,04189	1,35882
8	0,02652	0,05304	1,36996
16	0,03774	0,07548	1,39241
Cortex-A15 (L2=512 kB, 64 B, 16-way)			
2	0,00433	0,00867	2,88247
4	0,00495	0,00990	2,88370
8	0,01432	0,02865	2,90245
16	0,01767	0,03534	2,90914
32	0,03319	0,06639	2,94019

Note : CACTI ne trouve pas d'organisation valide pour A7 à 1 kB (d'où "–").

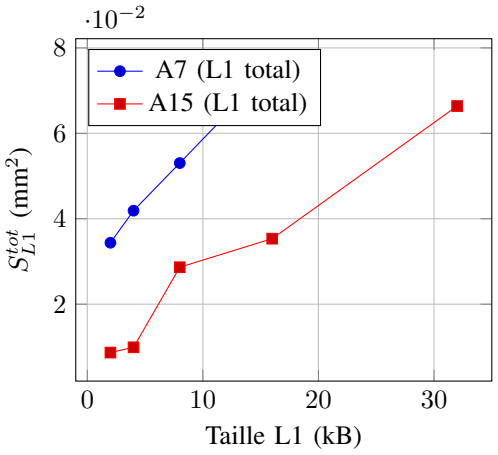


Fig. 7: Surface totale des L1 (I+D) en fonction de la taille.

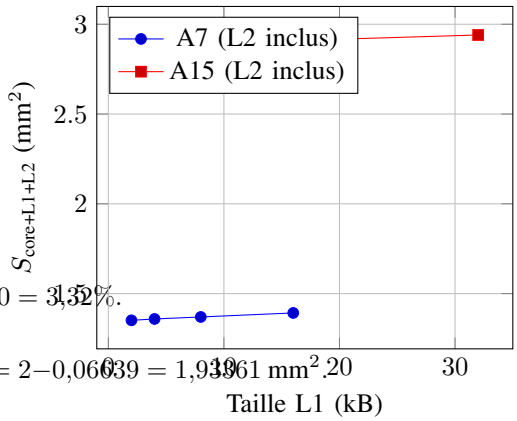


Fig. 8: Surface totale des cœurs (L1 variable + L2=512 kB).

b) Graphes (pgfplots).:

c) Efficacité surfacique.: On utilise les IPC issus des fichiers CSV (par processus et par taille de L1) et la surface totale du cœur avec L2 fixe (512 kB) :

$$\eta = \frac{IPC}{S_{\text{total}}} \quad \text{avec} \quad S_{\text{total}} = S_{\text{core hors L1}} + 2S_{L1} + S_{L2}.$$

Ici $S_{L2}(A7) = 0{,}94241$ mm^2 et $S_{L2}(A15) = 0{,}94019$ mm^2. Les valeurs suivantes sont donc en IPC/mm^2.

L1 (kB)	Dijkstra small	Dijkstra large	Blowfish
Cortex-A7			
1	–	–	–
2	0,1787	0,1778	0,1466
4	0,1860	0,1846	0,1465
8	0,2006	0,1989	0,1458
16	0,2025	0,2009	0,1434

TABLE XIV: Efficacité surfacique du Cortex-A7 (L2=512 kB).

L1 (kB)	Dijkstra small	Dijkstra large	Blowfish
Cortex-A15			
2	0,2249	0,2269	0,1833
4	0,2516	0,2500	0,1850
8	0,3253	0,3129	0,1845
16	0,3516	0,3386	0,1841
32	0,3908	0,3907	0,1823

TABLE XV: Efficacité surfacique du Cortex-A15 (L2=512 kB).

Note : CACTI ne fournit pas d'organisation valide pour A7 à 1 kB, d'où "–".

d) Efficacité surfacique.: On utilise les IPC issus des fichiers CSV (par processus et par taille de L1) et la surface totale du cœur avec L2 fixe (512 kB) :

$$\eta = \frac{IPC}{S_{\text{total}}} \quad \text{avec} \quad S_{\text{total}} = S_{\text{core hors L1}} + 2S_{L1} + S_{L2}.$$

Ici $S_{L2}(A7) = 0{,}94241$ mm^2 et $S_{L2}(A15) = 0{,}94019$ mm^2. Les valeurs suivantes sont donc en IPC/mm^2.

L1 (kB)	Dijkstra small	Dijkstra large	Blowfish
Cortex-A7			
1	–	–	–
2	0,1787	0,1778	0,1466
4	0,1860	0,1846	0,1465
8	0,2006	0,1989	0,1458
16	0,2025	0,2009	0,1434

TABLE XVI: Efficacité surfacique du Cortex-A7 (L2=512 kB).

L1 (kB)	Dijkstra small	Dijkstra large	Blowfish
Cortex-A15			
2	0,2249	0,2269	0,1833
4	0,2516	0,2500	0,1850
8	0,3253	0,3129	0,1845
16	0,3516	0,3386	0,1841
32	0,3908	0,3907	0,1823

TABLE XVII: Efficacité surfacique du Cortex-A15 (L2=512 kB).

Note : CACTI ne fournit pas d'organisation valide pour A7 à 1 kB, d'où "–".

REFERENCES

[1] M. J. P. (University of York), "Profiling," *Lecture Notes (4th Year HPC), University of York*. [Online]. Available: https://www-users.york.ac.uk/˜mijp1/teaching/4th_year_HPC/lecture_notes/Profiling.pdf. Accessed: Feb. 9, 2026.