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

Rapport de Travaux Pratiques

Luiz Gariglio Dos Santos
Helena Guachalla De Andrade
Santiago Florido Gomez
Franck Ulrich Kenfack Noumedem

Programme : Ingénieur STIC

Ecole Nationale des Techniques Avancees
ENSTA Paris
Fevrier 2026

Table des matières

1 Resume **2**

2 Introduction **2**

3 Evaluation des performances de différentes configurations de mémoires caches pour 4 algorithmes de multiplication de matrices **2**

3.1 Paramètres de configuration des caches utilisés dans gem5 3

3.2 Taux de défauts dans les différentes caches 3

4 Profiling de l'application **4**

4.1 pourcentage de chaque classe d'instructions 4

4.2 Catégorie d'instructions que nécessiterait une amélioration 7

4.3 Similitudes/divergences comportementales entre dijkstra, BlowFish, SSCA2-BCS, SHA-1 et le produit de polynômes 7

5 Impact de la taille des caches L1 **7**

5.1 Configuration de L1 performances pour le Cortex A7 8

5.2 Configuration de L1 performances pour le Cortex A15 14

6 Efficacité surfacique **16**

6.1 Paramètres de cache par défaut 16

6.2 Taille des deux coeurs (hors caches L1). 16

6.3 Surfaces totales du Cortex A7 et du Cortex A15 et l'efficacité surfacique de chaque processeur. 16

7 Efficacité énergetique. **19**

7.1 Puissance en mW consomme par chaque processeur à la fréquence maximale . . . 19

7.2 Configuration de L1 l'efficacité énergétique de chaque processeur (à fréquence maximale). 19

8 Architecture système big.LITTLE **20**

8.1 Dijkstra big.LITTLE 20

8.2 Blowfish big.LITTLE 21

9 Méthodologie de Spécification Architecturale **22**

10 Compromis et conclusion **23**

1 Resume

Ce rapport analyse l'influence de la hiérarchie mémoire et des contraintes ènergétiques sur les performances des processeurs. Dans un premier temps, nous èvaluons l'impact des caches L1/L2 sur des calculs matriciels, en mettant en èvidence le rôle clé de la localité et de l'organisation des données. Dans un second temps, nous menons une exploration de l'espace de conception d'une architecture *big.LITTLE* (Cortex-A7/A15) afin d'optimiser les configurations pour Dijkstra et Blowfish. Les performances sont mesurées avec Gem5 (IPC, taux de *miss*) et les coûts physiques des caches sont estimés avec CACTI, compilé en version 7.0 pour fiabiliser les résultats en 32 nm.

2 Introduction

Dans le contexte actuel des systèmes embarqués et du calcul haute performance, la conception de processeurs est confrontée à deux contraintes majeures : le *mur mémoire* (*Memory Wall*), qui limite les gains de performance lorsque la hiérarchie mémoire ne suit pas le rythme du calcul, et l'efficacité ènergétique, devenue déterminante pour les plateformes modernes. Ce rapport de travaux pratiques s'articule autour de deux ètudes complémentaires visant à caractériser et quantifier ces problématiques à travers des expérimentations reproductibles.

La première partie (Exercice 3) analyse l'impact de la hiérarchie mémoire (caches L1/L2) sur des algorithmes de calcul matriciel. L'objectif est de montrer que les performances ne dépendent pas uniquement de l'algorithme, mais aussi de la manière dont les données sont organisées et accédées en mémoire, notamment via la localité spatiale et temporelle.

La seconde partie (Exercice 4) porte sur une exploration de l'espace de conception (*Design Space Exploration*, DSE) appliquée à une architecture hétérogène *big.LITTLE*. Nous dimensionnons un cluster *LITTLE* (Cortex-A7) et un cluster *big* (Cortex-A15) pour des charges applicatives distinctes (Dijkstra et Blowfish), en mettant en èvidence les compromis entre performance, coût en surface et consommation.

Pour mener ces analyses, nous nous appuyons sur deux outils principaux : Gem5, utilisé en mode *Syscall Emulation* pour simuler l'exécution et extraire des indicateurs tels que l'IPC et les taux de *miss*, et CACTI, utilisé pour estimer la surface et l'énergie des caches. Enfin, afin de garantir la fiabilité des résultats de l'Exercice 4, nous avons automatisé les campagnes de simulation et pris l'initiative de compiler CACTI 7.0, ce qui a permis de corriger des erreurs critiques observées avec la version standard lors des estimations en technologies fines (notamment 32 nm).

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

3.1 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 1

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

3.2 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 2 – 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 3 – 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 4 – 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 2). 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.

4 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.

4.1 pourcentage de chaque classe d'instructions

Classe	**Dijkstra large (A7)**	**Dijkstra large (A15)**
Lecture (Load)	45 354 411 [22.2]	45 740 419 [22.3]
Écriture (Store)	19 449 403 [9.5]	19 639 263 [9.6]
Branchement	43 970 804 [21.6]	44 233 561 [21.5]
Références mémoire (MemRefs)	64 803 814 [31.8]	65 379 682 [31.8]
Calcul entier	139 193 914 [68.2]	140 143 451 [68.2]
Total d'instructions exécutées	**203 997 728**	**205 523 133**

TABLE 5 – Dijkstra large (Cortex-A7 vs Cortex-A15)

Classe	**Dijkstra small (A7)**	**Dijkstra small (A15)**
Lecture (Load)	10 282 770 [22.4]	10 455 962 [22.4]
Écriture (Store)	4 769 894 [10.4]	4 845 319 [10.4]
Branchement	9 839 620 [21.4]	9 990 687 [21.4]
Références mémoire (MemRefs)	15 052 664 [32.7]	15 301 281 [32.8]
Calcul entier	30 916 920 [67.3]	31 379 238 [67.2]
Total d'instructions exécutées	**45 969 584**	**46 680 519**

Table 6 – Dijkstra small (Cortex-A7 vs Cortex-A15)

Classe	**Blowfish small (A7)**	**Blowfish small (A15)**
Lecture (Load)	806 840 [22.5]	853 821 [23.9]
Écriture (Store)	430 428 [12.0]	418 788 [11.7]
Branchement	361 461 [10.1]	353 242 [9.9]
Références mémoire (MemRefs)	1 237 268 [34.5]	1 272 609 [35.7]
Calcul entier	2 348 046 [65.5]	2 294 903 [64.3]
Total d'instructions exécutées	**3 585 314**	**3 567 512**

Table 7 – Blowfish small (Cortex-A7 vs Cortex-A15)

Classe	**Blowfish large (A7)**	**Blowfish large (A15)**
Lecture (Load)	3 014 650 [22.9]	3 295 893 [24.4]
Écriture (Store)	1 677 678 [12.7]	1 679 901 [12.4]
Branchement	1 350 032 [10.2]	1 356 138 [10.0]
Références mémoire (MemRefs)	4 692 328 [35.6]	4 975 794 [36.9]
Calcul entier	8 499 974 [64.4]	8 518 974 [63.1]
Total d'instructions exécutées	**13 192 302**	**13 494 768**

Table 8 – Blowfish large (Cortex-A7 vs Cortex-A15)

Métrique	**SHA-1**
IntAlu (Calcul entier)	83.52 %
MemRead (Lecture)	11.77 %
MemWrite (Écriture)	4.71 %
Float (Flottants)	0.00 %
Profil identifié	*Compute Bound*

Table 9 – Répartition en pourcentage des classes d'instructions pour SHA-1

Il est important de souligner que, même si le *profiling* montre que les instructions dominantes dans les applications sont des instructions de calcul entier, le pourcentage élevé associé aux opérations mémoire crée néanmoins la nécessité d'évaluer la latence des processus mémoire. L'objectif est de déterminer si, malgré leur caractère non dominant, ces opérations peuvent constituer le goulot d'étranglement de l'application. Pour cela, on propose d'estimer l'effet des *misses* dans les caches L1 et L2, sous la forme d'un coût approximatif en cycles, afin d'évaluer ensuite la part de ces pertes dans le CPI et de conclure si ces résultats sont, ou non, représentatifs par rapport au pourcentage d'instructions liées aux calculs. Il convient de préciser que le calcul relatif au cache L1 est effectué uniquement à partir des *misses* du cache de données, car les *misses* du cache d'instructions ne sont pas considérés comme suffisamment représentatifs pour être pris en compte dans l'analyse.

Pour réaliser cette estimation, on cherche d'abord à déterminer le nombre de *ticks* par cycle dans la simulation, défini par l'équation (1) :

$$\mathrm{TPC} = \frac{\mathrm{simTicks}}{\mathrm{numCycles}} \tag{1}$$

La fréquence CPU correspondante est alors calculée à partir de l'équation (2) :

$$f_{\mathrm{cpu}} = \frac{\mathrm{simFreq}}{\mathrm{TPC}} \tag{2}$$

Ensuite, pour les simulations, on prévoit de calculer une pénalité de perte en L1 à l'aide de l'équation (3) :

$$P_{L1} = \frac{L_{L1}}{\mathrm{TPC}} \quad [\mathrm{cycles/miss}] \tag{3}$$

où L_{L1} représente la latence moyenne d'un *miss* en L1 exprimée en *ticks/miss*.

De la même manière, on applique l'équation correspondante pour L2, en gardant à l'esprit que le coût d'un *miss* L2 implique un accès à la mémoire principale (RAM) et est donc associé à un coût plus élevé, comme indiqué par l'équation (4) :

$$P_{L2} = \frac{L_{L2}}{\mathrm{TPC}} \quad [\mathrm{cycles/miss}] \tag{4}$$

où L_{L2} représente la latence moyenne d'un *miss* en L2 exprimée en *ticks/miss*.

On propose ensuite de calculer une fréquence de *misses* par tranche de 1000 instructions, pour les deux niveaux de mémoire, à l'aide de l'équation (5) :

$$\mathrm{MPKI} = \frac{\mathrm{misses}}{\mathrm{numInsts}} \cdot 1000 \tag{5}$$

Enfin, à partir de cela, on estime pour chaque niveau de cache (L1 et L2) le coût moyen de ces pertes sur le CPI. Pour L1, on utilise l'approximation donnée par l'équation (6) :

$$\Delta\mathrm{CPI}_{L1} \approx \frac{\mathrm{MPKI}_{L1}}{1000} \cdot P_{L1} \tag{6}$$

Case	**CPI**	**%CPI mem total**	**%CPI L1-only (approx)**	**%CPI L2-miss**
smalldijkstraA7	3.415	2.08%	1.71%	0.37%
smalldijkstraA15	0.863	7.64%	6.81%	0.83%
largedijkstraA7	3.424	2.50%	2.42%	0.08%
largedijkstraA15	0.874	9.99%	9.80%	0.19%

TABLE 10 – Contribution estimée de la pénalité mémoire au CPI pour les exécutions de Dijkstra

Case	**CPI**	**%CPI mem total**	**%CPI L1-only (approx)**	**%CPI L2-miss**
bfA7_small	3.362	3.97%	0.50%	3.47%
bfA15_small	0.705	18.64%	9.34%	9.30%
bfA7_large	3.303	1.11%	0.15%	0.96%
bfA15_large	0.668	5.30%	2.56%	2.75%

TABLE 11 – Contribution estimée de la pénalité mémoire au CPI pour les exécutions de Blowfish

4.2 Catégorie d'instructions que nécessiterait une amélioration

Le profiling indique que le goulot d'étranglement est d'abord côté cœur (OoO/ROB, dépendances, contrôle) : augmenter le parallélisme et le nombre d'ALU entières apporte le gain principal. Ensuite, si la mémoire domine, améliorer la localité et l'organisation des données réduit les misses et la latence des loads/stores. Enfin, la part de branchements justifie de restructurer les boucles et de limiter les sauts pour réduire les pénalités de contrôle.

4.3 Similitudes/divergences comportementales entre dijkstra, BlowFish, SSCA2-BCS, SHA-1 et le produit de polynômes

On observe aussi qu'après l'amélioration des unités de calcul et la capacité à exploiter davantage le parallélisme au niveau des instructions, lorsque le CPI diminue fortement, les *misses* de cache deviennent plus significatifs. Dans ce contexte, la mise en œuvre d'un design réduisant les *misses* de cache devient plus intéressante pour une implémentation ultérieure.

De plus, il convient de considérer que, dans le cas de Dijkstra, en raison de la nature de l'algorithme et des structures de données qu'il emploie, les branchements ont un effet important. Cela suggère qu'une amélioration du prédicteur de branchement, pour cette application et pour ces configurations, pourrait également représenter un gain significatif en IPC.

En termes de comparaison, on observe que **Dijkstra** / **SSCA2-BCS**, en tant qu'algorithmes sur graphes, présentent un profil mémoire caractérisé par de nombreux accès irréguliers : beaucoup de *loads/stores*, du *pointer chasing* et une faible localité. Cela rend les mécanismes de préchargement (*prefetching*) nettement moins efficaces. La nature de ces algorithmes implique également des branchements fréquents et souvent moins prédictibles (boucles et conditions dépendantes des données), ce qui rend la performance très sensible à la fenêtre OoO (ROB/LSQ), au prédicteur de branchement et à la capacité à masquer la latence.

De son côté, **Blowfish**, comme Dijkstra, est dominé par des instructions entières, mais s'en distingue par des accès souvent plus séquentiels et mieux localisés (bonne localité, caches plus efficaces). Cela se reflète dans le fait que, pour les jeux de données *large* de Blowfish, la part du CPI associée aux opérations mémoire est considérablement plus faible que celle induite par les *misses* dans Dijkstra. Il convient toutefois de noter que, pour les cas *small*, ces conditions ne sont pas nécessairement vérifiées : dans Blowfish, le dataset small est souvent dominé par des effets de démarrage et d'*overhead*. Par ailleurs, l'impact des branchements est globalement moins marqué dans Blowfish que dans Dijkstra.

Dans le cas de **SHA-1**, l'effet des instructions entières est encore plus dominant que dans les deux applications précédentes, ce qui met davantage en évidence la nature *compute-bound* de son implémentation. De plus, SHA-1 présente généralement un bon comportement de *streaming* sur les tampons (*buffers*).

Enfin, pour les **produits de polynômes** / **convolution**, les accès sont plutôt séquentiels (tableaux), ce qui conduit à des caches efficaces. Néanmoins, avec de très grands tableaux et une bonne vectorisation, l'exécution peut devenir limitée par la bande passante mémoire (*memory-bandwidth-bound*), en particulier lorsque les produits sont effectués en flottant, où le volume d'octets transférés par opération (octets par flop) devient élevé. Dans ce cas, une forte proportion d'opérations flottantes peut également conduire à un comportement *FP-bound*.

5 Impact de la taille des caches L1

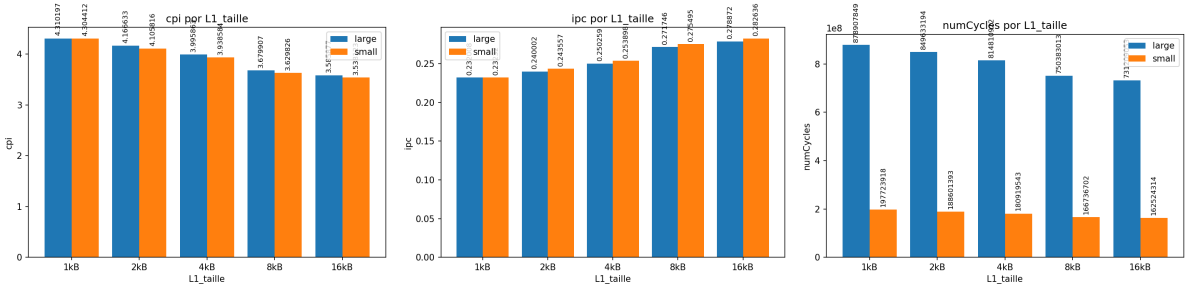
5.1 Configuration de L1 performances pour le Cortex A7

Paramètres de simulation Gem5 utilisés (campagne L1, Cortex-A7) :

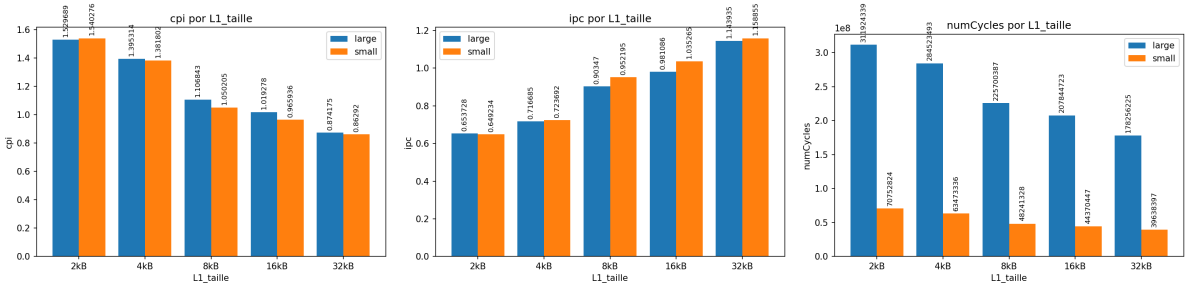
- **Coeur :** `DerivO3CPU` (profil Cortex-A7).
- **I-L1 (cache/bloc/assoc.) :** `{1,2,4,8,16}kB / 32B / 2`.
- **D-L1 (cache/bloc/assoc.) :** `{1,2,4,8,16}kB / 32B / 2`.
- **L2 (cache/bloc/assoc.) :** `512kB / 32B / 8`.
- **Prédicteur de branchement :** `BiModeBP`, `BTBEntries=256`.
- **Fetch queue :** `8`.
- **Decode/Issue/Commit :** `2 / 4 / 2`.
- **RUU/LSQ :** `2 / 8` (avec `LQ=8`, `SQ=8`).

Le paramètre `fetchQueueSize` implémenté est celui fourni dans le dépôt, tel qu'il apparaît dans le fichier de base distribué. Il convient de préciser qu'il ne correspond pas à la valeur indiquée dans le tableau, mais qu'il s'agit bien de la valeur imposée par le fichier de configuration fourni.

Les Figs. (1)-(3) et (4)-(6) présentent l'évolution des performances en fonction de la taille du cache L1 pour dijkstra applications et pour Blowfish applications respectivement.



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



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

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

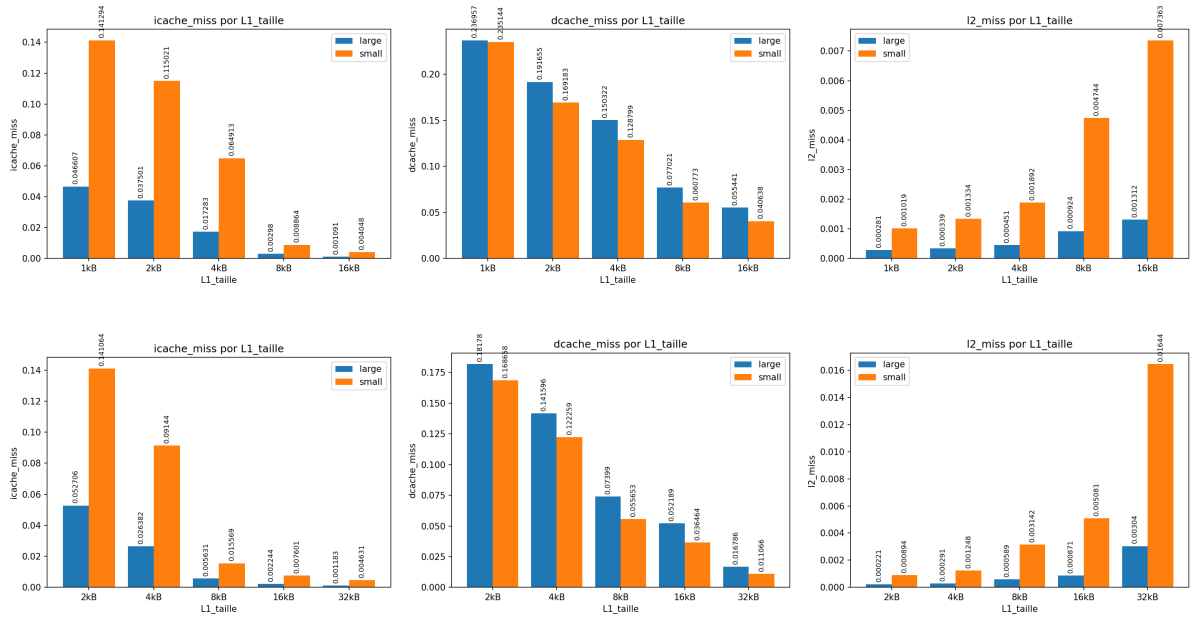


FIGURE 2 – Dijkstra : taux de défauts I-Cache, D-Cache et L2 en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

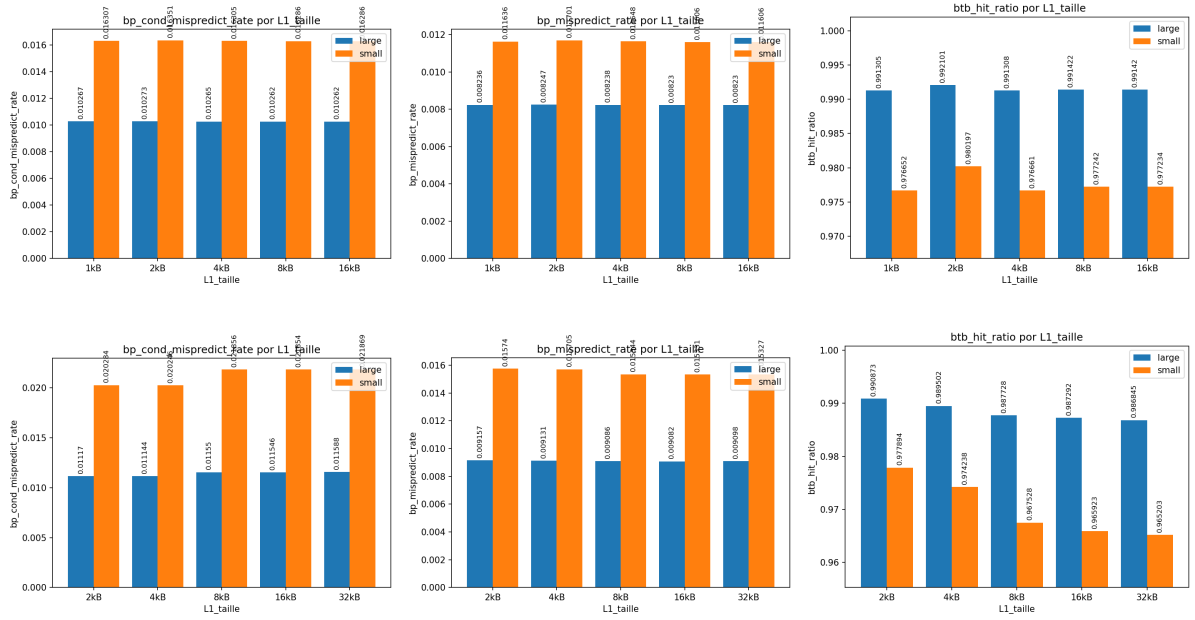
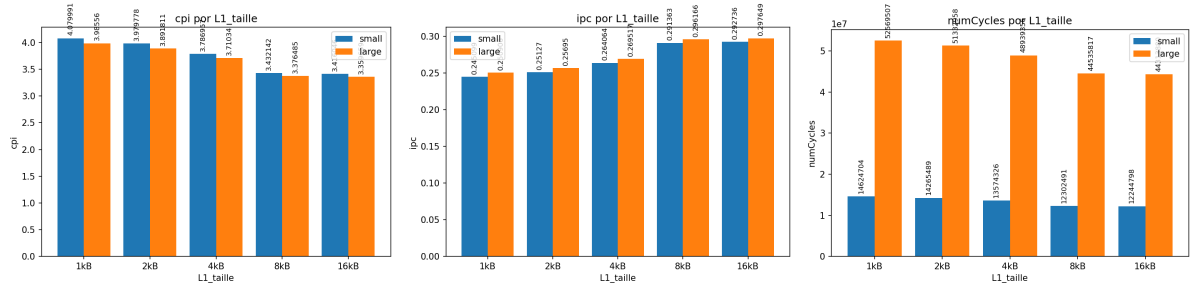
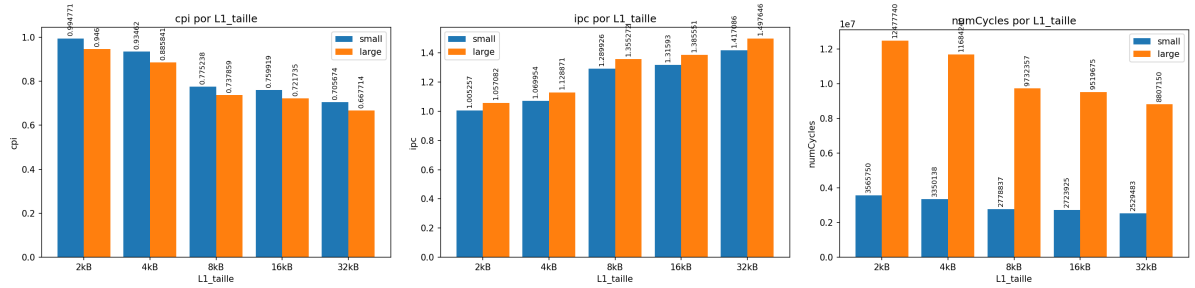


FIGURE 3 – Dijkstra : métriques de prédiction de branchement en fonction de la taille du cache L1 (A7 en haut, A15 en bas).



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



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

FIGURE 4 – Blowfish : performance générale en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

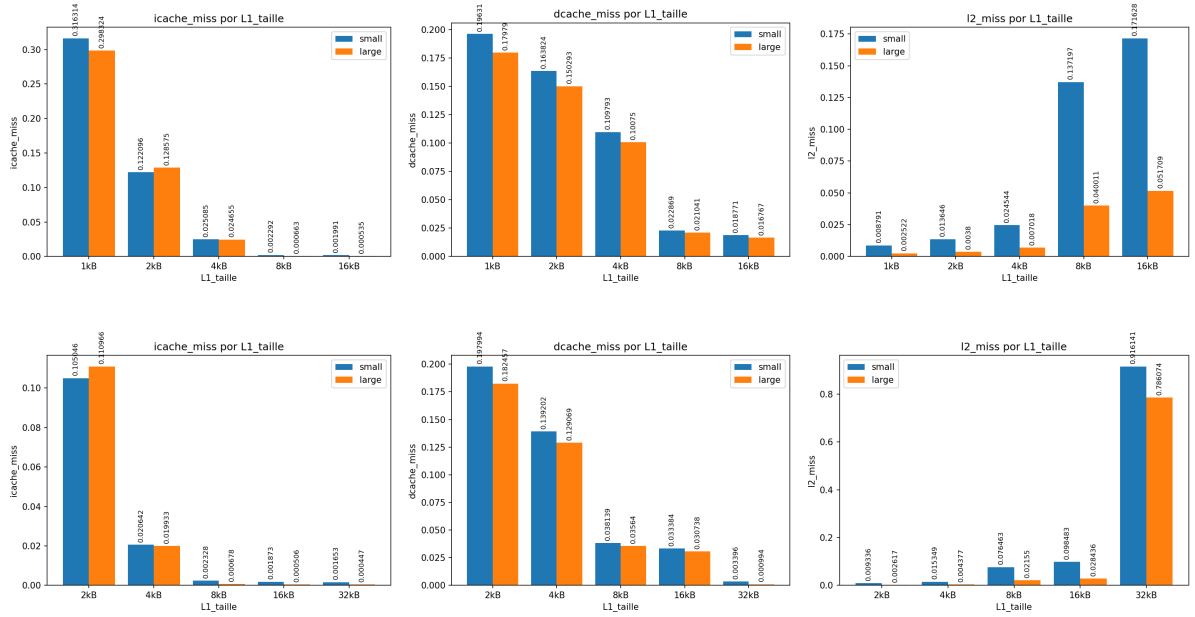


FIGURE 5 – Blowfish : taux de défauts I-Cache, D-Cache et L2 en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

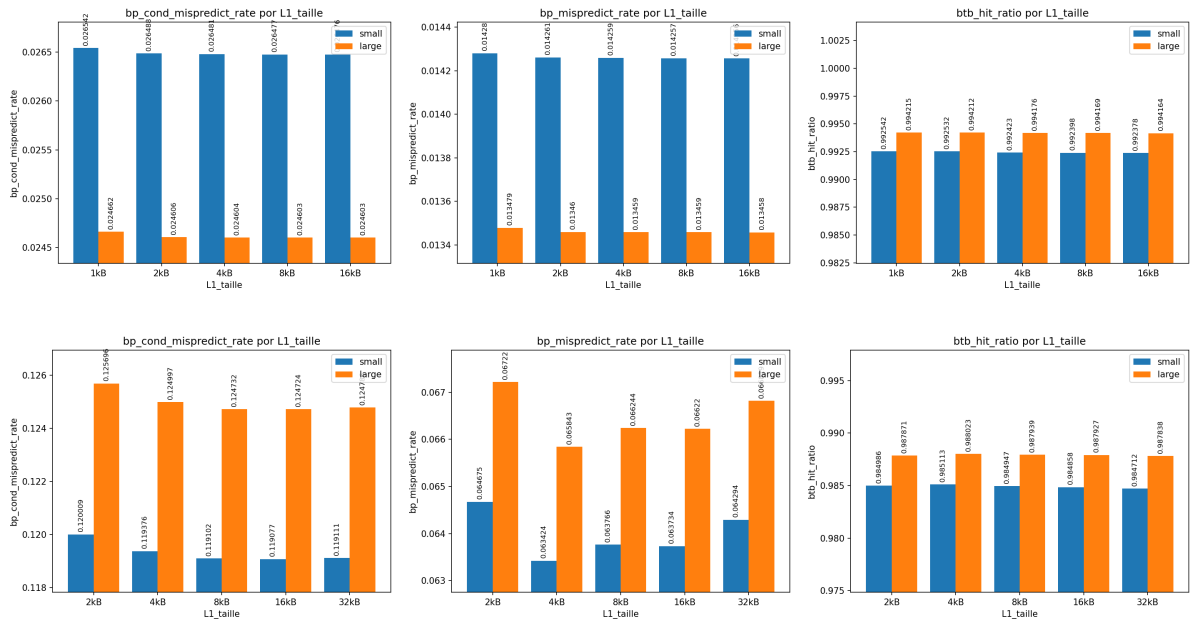


FIGURE 6 – Blowfish : métriques de prédiction de branchement en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

Pour les implémentations sur Cortex-A7, et plus particulièrement dans le cas des applications basées sur l'algorithme de Dijkstra, on observe, aussi bien pour le *large* que pour le *small*, une zone critique comprise entre 1 kB et 4 kB, dans laquelle les gains d'IPC liés à l'augmentation de la taille du cache L1 sont rapides. Cette zone correspond à la transition entre un L1 trop petit pour contenir les principales structures de données, et un L1 suffisamment grand pour supporter les accès répétés, principalement sur les données, au cours de l'exécution de l'application.

On constate qu'à 8 kB apparaît un point d'inflexion, où l'amélioration continue et atteint le saut de performance le plus notable par rapport à la taille de L1 précédente. De plus, on observe que les améliorations du cache d'instructions commencent à saturer, en atteignant un niveau de pertes minimal et un gain proche de 100% par rapport à la configuration avec un L1 de 1 kB. Parallèlement, la diminution des pertes dans le cache de données est la plus significative parmi les intervalles considérés, puis elle commence à montrer un comportement de saturation en suivant la trajectoire des gains.

Enfin, cette saturation se consolide vers 16 kB, où les gains d'IPC dus à l'augmentation de la taille du L1 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. Il convient également de souligner qu'il devient évident qu'il existe une dépendance entre l'impact des branchements et la taille du cache L1. Ce résultat est attendu, dans la mesure où le comportement des branchements dépend du prédicteur de branchement.

De manière complémentaire, on voit que, dans le cas du Dijkstra *small*, la perte de cache de données est plus faible que pour le *large*, ce qui est un résultat attendu, puisque, pour le *large*, le *working set* de données dépasse plus rapidement la capacité minimale, ce qui accentue les défauts pour les petites tailles. 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 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 que le code chaud est capturé.

De son côté, dans le cas de Blowfish sur le même Cortex-A7, la compacité du code et la localité des données exploitées par l'algorithme ont un effet clair sur la dynamique de diminution des pertes, principalement au niveau du cache de données, en comparaison avec la dynamique observée pour Dijkstra. Aussi bien pour Blowfish *large* que pour Blowfish *small*, on met en évidence une zone critique d'amélioration lorsque la taille du cache augmente de 1 kB à 4 kB.

On observe également une zone d'inflexion marquée autour d'une taille de 8 kB, pour laquelle on atteint la plus grande différence en gain d'IPC et la plus forte diminution des pertes, en particulier pour le cache de données, par rapport à la taille de cache immédiatement précédente. Au-delà de 8 kB, une saturation du système devient visible, avec des gains d'IPC marginaux et pratiquement négligeables. Cette dynamique, comme dans le cas de Dijkstra, est déterminée par la saturation des diminutions de pertes en cache.

Néanmoins, en raison des conditions déjà mentionnées de localité et des modes d'accès et de stockage en mémoire propres à cet algorithme, on constate que, du côté du cache d'instructions, le niveau de saturation est atteint pour des tailles de cache plus petites. De plus, pour les pertes du cache de données, Blowfish obtient de meilleurs résultats que Dijkstra à taille de cache identique, en atteignant une diminution de 90,44% par rapport à l'architecture avec un L1 de 1 kB. L'effet de la taille du cache sur les branchements, comme dans le cas de Dijkstra, ne présente pas de relation directe avec la taille du cache.

Une observation que l'on peut formuler concernant le comportement des algorithmes pour différentes tailles de cache est qu'elle met en évidence que le goulot d'étranglement de l'implémentation n'était pas nécessairement lié aux limitations mémoire. En effet, en ne faisant varier que les caches, on atteint une saturation des améliorations d'IPC pour des valeurs proches de seulement 20% par rapport à la configuration de base, tandis que la variation résultant du traitement du parallélisme sur A15, comparé à A7, produit des résultats beaucoup plus importants en termes d'IPC.

Ce constat renforce l'appréciation initiale selon laquelle la classe dominante d'instructions dans les applications correspond effectivement aux opérations de calcul entier. On propose donc d'analyser les comportements des algorithmes sur Cortex-A15 afin d'évaluer l'hypothèse formulée.

En somme, en tenant compte de l'effet de saturation observé en fonction de la taille du L1, la configuration retenue est de 8 kB. Il s'agit du "sweet spot" offrant le meilleur compromis performance/surface pour ce profil architectural, et ce, pour les deux applications.

L1	**Gain IPC (%)**	**Baisse miss I-Cache (%)**	**Baisse miss D-Cache (%)**
1 kB	0,00	0,00	0,00
2 kB	4,84	18,59	28,05
4 kB	9,29	54,06	45,23
8 kB	18,58	93,73	74,15
16 kB	21,66	97,14	82,72

Table 12 – Dijkstra small (Cortex-A7) : gains relatifs par rapport à 1 kB.

L1	Gain IPC (%)	Baisse miss I-Cache (%)	Baisse miss D-Cache (%)
1 kB	0,00	0,00	0,00
2 kB	3,45	19,54	19,12
4 kB	7,87	62,92	36,56
8 kB	17,13	93,61	67,50
16 kB	20,20	97,66	76,60

TABLE 13 – Dijkstra large (Cortex-A7) : gains relatifs par rapport à 1 kB.

L1	Gain IPC (%)	Baisse miss I-Cache (%)	Baisse miss D-Cache (%)
2 kB	0,00	0,00	0,00
4 kB	11,47	35,18	27,51
8 kB	46,66	88,96	67,00
16 kB	59,46	94,61	78,38
32 kB	78,50	96,72	93,44

TABLE 14 – Dijkstra small (Cortex-A15) : gains relatifs par rapport à 2 kB.

L1	Gain IPC (%)	Baisse miss I-Cache (%)	Baisse miss D-Cache (%)
2 kB	0,00	0,00	0,00
4 kB	9,63	49,94	22,11
8 kB	38,20	89,32	59,30
16 kB	50,08	95,74	71,29
32 kB	74,99	97,76	90,77

TABLE 15 – Dijkstra large (Cortex-A15) : gains relatifs par rapport à 2 kB.

L1	Gain IPC (%)	Baisse miss I-Cache (%)	Baisse miss D-Cache (%)
1 kB	0,00	0,00	0,00
2 kB	2,52	61,40	16,55
4 kB	7,74	92,07	44,07
8 kB	18,88	99,28	88,35
16 kB	19,44	99,37	90,44

TABLE 16 – Blowfish small (Cortex-A7) : gains relatifs par rapport à 1 kB.

L1	Gain IPC (%)	Baisse miss I-Cache (%)	Baisse miss D-Cache (%)
1 kB	0,00	0,00	0,00
2 kB	2,41	56,90	16,41
4 kB	7,42	91,74	43,96
8 kB	18,04	99,78	88,30
16 kB	18,63	99,82	90,67

TABLE 17 – Blowfish large (Cortex-A7) : gains relatifs par rapport à 1 kB.

L1	Gain IPC (%)	Baisse miss I-Cache (%)	Baisse miss D-Cache (%)
2 kB	0,00	0,00	0,00
4 kB	6,44	80,35	29,69
8 kB	28,32	97,78	80,74
16 kB	30,90	98,22	83,14
32 kB	40,97	98,43	98,28

TABLE 18 – Blowfish small (Cortex-A15) : gains relatifs par rapport à 2 kB.

L1	Gain IPC (%)	Baisse miss I-Cache (%)	Baisse miss D-Cache (%)
2 kB	0,00	0,00	0,00
4 kB	6,79	82,04	29,26
8 kB	28,21	99,39	80,47
16 kB	31,07	99,54	83,15
32 kB	41,68	99,60	99,46

TABLE 19 – Blowfish large (Cortex-A15) : gains relatifs par rapport à 2 kB.

5.2 Configuration de L1 performances pour le Cortex A15

Paramètres de simulation Gem5 utilisés (campagne L1, Cortex-A15) :

- **Coeur :** `DerivO3CPU` (profil Cortex-A15).
- **I-L1 (cache/bloc/assoc.) :** `{2,4,8,16,32}kB / 64B / 2`.
- **D-L1 (cache/bloc/assoc.) :** `{2,4,8,16,32}kB / 64B / 2`.
- **L2 (cache/bloc/assoc.) :** `512kB / 64B / 16`.
- **Prédicteur de branchement :** `LocalBP`, `BTBEntries=256`.
- **Fetch queue :** `15`.
- **Decode/Issue/Commit :** `4 / 8 / 4`.
- **RUU/LSQ :** `16 / 16` (avec `LQ=16`, `SQ=16`).

  Le paramètre `fetchQueueSize` implémenté est celui fourni dans le dépôt, tel qu'il apparaît dans le fichier de base distribué. Il convient de préciser qu'il ne correspond pas à la valeur indiquée dans le tableau, mais qu'il s'agit bien de la valeur imposée par le fichier de configuration fourni.

Pour le Cortex-A15, les tendances observées, une fois l'optimisation du traitement des opérations du cœur déjà mise en place, deviennent plus marquées : la comparaison de l'IPC entre les deux Cortex (A15 et A7) montre des gains plus élevés lorsque l'on implémente un L1 de plus grande taille. Dans ces cas, la zone critique s'étend typiquement de 2 kB à 16 kB, avec une croissance de l'IPC et une augmentation du pourcentage de diminution des pertes sur les données. Pour Dijkstra, on observe aussi qu'entre 16 kB et 32 kB, l'augmentation de l'IPC est très significative, accompagnée d'une diminution notable des *misses* de données en cache, ce qui est particulièrement important pour Dijkstra compte tenu de la manière dont les accès mémoire se produisent, en lien avec les structures de données employées par l'algorithme.

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 taux de *miss* 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'une ligne 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 taux de *miss* 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 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.

Dans le cas de Blowfish, le comportement est similaire à celui de Dijkstra, dans le sens où l'augmentation de l'IPC se maintient jusqu'à atteindre un point d'inflexion apparent autour de 32 kB. À partir de cette taille, la saturation de la diminution des pertes dans la D-cache laisse prévoir un ralentissement, voire un stagnation, des gains d'IPC. Néanmoins, il convient de souligner que, bien que les gains soient supérieurs à ceux obtenus sur A7, ils n'atteignent qu'environ 40% par rapport à l'IPC de la configuration de référence avec un L1 de 1 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 dès qu'une petite capacité est disponible ; l'augmentation de L1 a donc 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. On peut également constater que, pour les deux applications, le prédicteur de branchement en fonction de la taille du cache L1 ne présente pas de variations significatives associées à ce paramètre.

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 offrant 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 augmente, 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.

En somme, en tenant compte de l'effet de saturation observé en fonction de la taille du L1, la configuration retenue est de 32 kB. Il s'agit du pointe offrant le meilleur compromis performance/surface pour ce profil architectural, et ce, pour les deux applications.

6 Efficacité surfacique

6.1 Paramètres de cache par défaut

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.

6.2 Taille des deux coeurs (hors caches L1).

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 = 32 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 (32 kB)	0,06644	0,06644	0,13287
A15 (32 kB)	0,03319	0,03319	0,06639

TABLE 20 – Surfaces des caches L1 (I et D) pour une taille de 32 kB.

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

$$\%L1(A7) = \frac{0{,}13287}{0{,}45} \times 100 = 29{,}52\%,$$
$$\%L1(A15) = \frac{0{,}06639}{2} \times 100 = 3{,}32\%.$$

$$S_{\text{core hors L1}}(A7) = 0{,}45 - 0{,}13287 = 0{,}31713 \text{ mm}^2,$$
$$S_{\text{core hors L1}}(A15) = 2 - 0{,}06639 = 1{,}93361 \text{ mm}^2.$$

Le A7 consacre une fraction beaucoup plus importante de sa surface au L1 ($\approx 29{,}52\%$) que le A15 ($\approx 3{,}32\%$), ce qui reflète un cœur plus compact à taille de L1 comparable, le poids surfacique est plus élevé sur A7 et aussi Sur le Cortex-A15, la logique complexe (exécution *Out-of-Order*, renommage, etc.) domine massivement. Les caches L1 sont proportionnellement petits. .

6.3 Surfaces totales du Cortex A7 et du Cortex A15 et l'efficacité surfacique de chaque processeur.

En disposant déjà des conditions relatives à la surface du cœur (hors cache), il est possible, à l'aide de CACTI, de déterminer la surface du cache L2 en configurant les paramètres de simulation afin de définir un L2 de 512 kB. De plus, on simule également avec CACTI afin d'estimer la surface du cache L1 pour l'ensemble de ses dimensions, et ce, pour les deux microprocesseurs.

En appliquant l'équation (7), on obtient la surface totale du système :

$$S_{\text{système}} = S_{\text{cœur\_nu}} + (2 \times S_{L1,\ \text{variable}}) + S_{L2,\ \text{fixe}} \tag{7}$$

Les résultats sont présentés dans les Fig. (7)-(8).

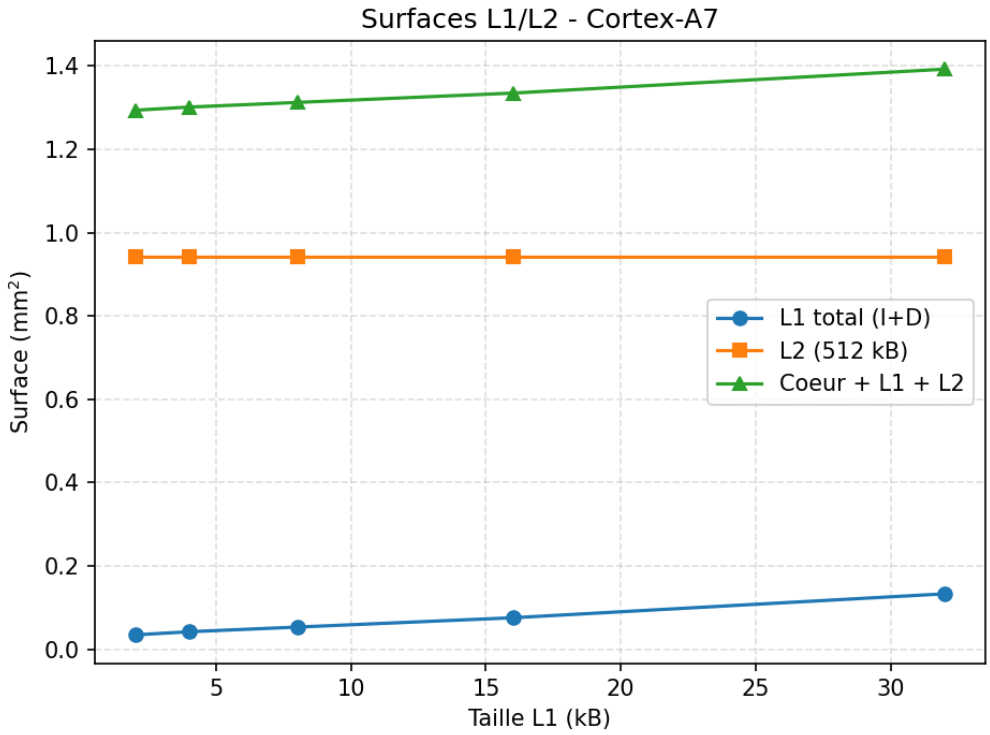


FIGURE 7 – Surfaces L1, L2 et cœur total pour Cortex-A7 (L2 = 512 kB).

L1 (kB)	S_{L1} (mm^2)	S_{L1}^{tot} (mm^2)	S_{L2} (mm^2)	$S_{\text{core+L1+L2}}$ (mm^2)
2	0,01719	0,03438	0,94241	1,29391
4	0,02094	0,04189	0,94241	1,30143
8	0,02652	0,05304	0,94241	1,31257
16	0,03774	0,07548	0,94241	1,33502
32	0,06644	0,13287	0,94241	1,39241

TABLE 21 – Surfaces calculées pour Cortex-A7 (L1 variable, L2 = 512 kB).

L1 (kB)	**Dijkstra small**	**Dijkstra large**	**Blowfish small**	**Blowfish large**
2	0,18823	0,18549	0,19419	0,19858
4	0,19509	0,19230	0,20290	0,20709
8	0,20989	0,20703	0,22198	0,22564
16	0,21171	0,20889	0,21928	0,22296
32	–	–	–	–

TABLE 22 – Efficacite surfacique (IPC/mm^2) pour Cortex-A7 (L2 = 512 kB).

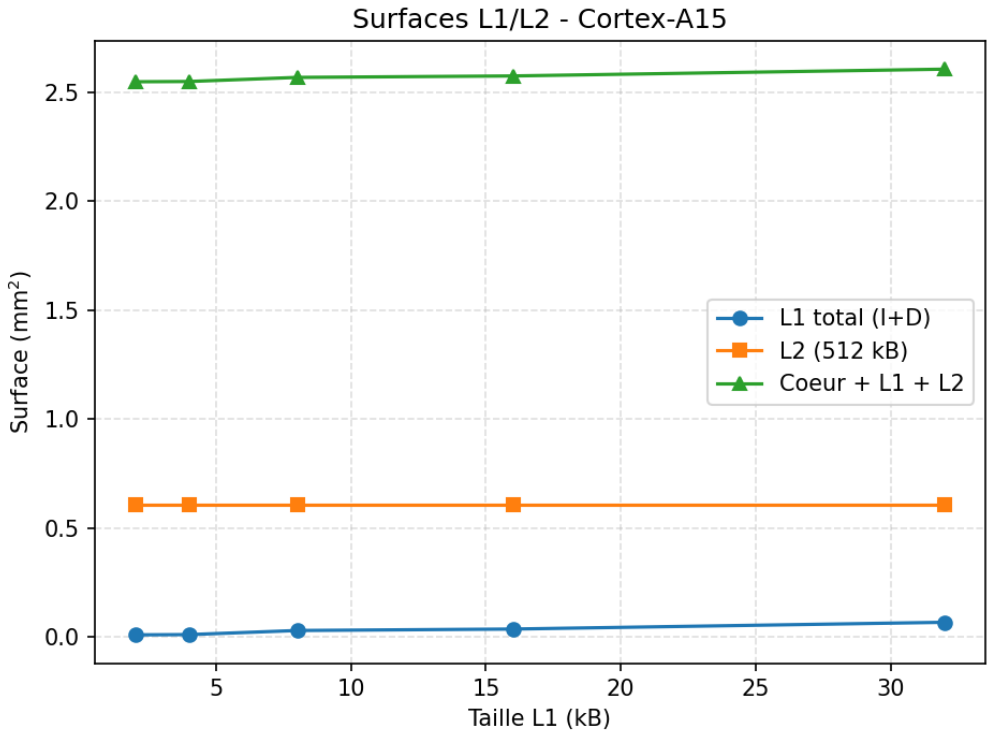


FIGURE 8 – Surfaces L1, L2 et cœur total pour Cortex-A15 (L2 = 512 kB).

L1 (kB)	S_{L1} (mm^2)	S_{L1}^{tot} (mm^2)	S_{L2} (mm^2)	$S_{\text{core+L1+L2}}$ (mm^2)
2	0,00433	0,00867	0,60544	2,54772
4	0,00495	0,00990	0,60544	2,54895
8	0,01432	0,02865	0,60544	2,56770
16	0,01767	0,03534	0,60544	2,57439
32	0,03319	0,06639	0,60544	2,60544

TABLE 23 – Surfaces calculées pour Cortex-A15 (L1 variable, L2 = 512 kB).

L1 (kB)	**Dijkstra small**	**Dijkstra large**	**Blowfish small**	**Blowfish large**
2	0,25483	0,25659	0,39457	0,41491
4	0,28392	0,28117	0,41976	0,44288
8	0,37084	0,35186	0,50237	0,52782
16	0,40214	0,38109	0,51116	0,53821
32	0,44478	0,43906	0,54390	0,57482

TABLE 24 – Efficacite surfacique (IPC/mm^2) pour Cortex-A15 (L2 = 512 kB).

En analysant l'efficacité surfacique des deux Cortex, on constate que, pour l'ensemble des algorithmes, l'efficacité surfacique est supérieure sur le Cortex-A15. De plus, aussi bien pour le Cortex-A7 que pour le Cortex-A15, l'efficacité surfacique conserve un comportement similaire à celui analysé pour l'IPC sur les deux processeurs : sur Cortex-A7, on observe une croissance jusqu'à 8 kB de cache, ce qui correspond à un point d'inflexion, puis, à partir de cette valeur, la saturation de la croissance de l'IPC identifiée précédemment se traduit par des diminutions de l'efficacité surfacique (comme dans le cas de Blowfish) ou, à défaut, par une stagnation de

l'efficacité surfacique (comme dans le cas de Dijkstra), dans la mesure où la saturation n'est pas encore atteinte de manière parfaitement ponctuelle.

De son côté, pour le Cortex-A15, la dynamique de croissance de l'IPC se traduit également par des augmentations de l'efficacité surfacique pour l'ensemble des applications, et ce jusqu'à 32 kB de cache. Ces éléments renforcent les choix de configuration proposés, lesquels avaient été initialement établis à partir de l'analyse de l'IPC dans la section précédente.

7 Efficacité énergetique.

7.1 Puissance en mW consomme par chaque processeur à la fréquence maximale

Les consommations énergétiques du Cortex-A7 et du Cortex-A15 sont de 0,10 mW/MHz et 0,20 mW/MHz, respectivement. De plus, en technologie 28 nm, les fréquences maximales du Cortex-A7 et du Cortex-A15 sont de 1,0 GHz et 2,5 GHz, respectivement. La puissance à fréquence maximale pour chaque microprocesseur peut être estimée comme suit.

Le calcul de la puissance à fréquence maximale ($P_{\max}$) est donné par l'équation (8) :

$$P_{\max} = \left(\frac{\text{mW}}{\text{MHz}}\right) \times (\text{MHz}) \tag{8}$$

1. Cortex-A7 : en appliquant l'équation (8), on obtient :

$$P_{A7} = 0{,}10 \ \frac{\text{mW}}{\text{MHz}} \times 1000 \ \text{MHz} = 100 \ \text{mW} \tag{9}$$

2. Cortex-A15 : de la même manière, on obtient :

$$P_{A15} = 0{,}20 \ \frac{\text{mW}}{\text{MHz}} \times 2500 \ \text{MHz} = 500 \ \text{mW} \tag{10}$$

À partir de cette puissance à fréquence maximale, il est possible d'estimer l'efficacité énergétique à l'aide de la formule donnée à l'équation (11) :

$$E = \text{IPC}\, P_{\max} \tag{11}$$

7.2 Configuration de L1 l'efficacité énergétique de chaque processeur (à fréquence maximale).

Cette expression est appliquée à toutes les applications, pour chacune des configurations dans lesquelles elles ont été simulées.

L1 (kB)	Dijkstra small	Dijkstra large	Blowfish small	Blowfish large
1	0,00232	0,00232	0,00245	0,00251
2	0,00244	0,00240	0,00251	0,00257
4	0,00254	0,00250	0,00264	0,00270
8	0,00275	0,00272	0,00291	0,00296
16	0,00283	0,00279	0,00293	0,00298

TABLE 25 – Efficacite energetique (IPC/mW) pour Cortex-A7 ($P_{\max} = 100\,\text{mW}$).

L1 (kB)	Dijkstra small	Dijkstra large	Blowfish small	Blowfish large
2	0,00130	0,00131	0,00201	0,00211
4	0,00145	0,00143	0,00214	0,00226
8	0,00190	0,00181	0,00258	0,00271
16	0,00207	0,00196	0,00263	0,00277
32	0,00232	0,00229	0,00283	0,00300

TABLE 26 – Efficacite energetique (IPC/mW) pour Cortex-A15 ($P_{\max} = 500\,\text{mW}$).

On a conclu qu'en définissant l'efficacité du projet à partir de l'IPC et de la puissance consommée à la fréquence maximale, on obtient les résultats attendus. En effet, plus la taille du cache augmente, plus l'IPC s'améliore et, avec un dénominateur constant (ou fixé), l'efficacité énergétique augmente pour les deux microprocesseurs.

Par ailleurs, il est important de souligner que, pour la plupart des tailles de cache comparables entre les deux architectures proposées, l'efficacité énergétique du microprocesseur A15 est inférieure à celle du microprocesseur A7.

8 Architecture système big.LITTLE

En s'inscrivant dans l'approche ARM *big.LITTLE*, on propose de sélectionner, pour chacune des applications, une configuration de cache L1 pouvant être intégrée dans chaque cluster, en traitant les applications séparément. Cette sélection s'appuie sur les résultats d'efficacité énergétique obtenus pour les différentes configurations, pour chacune des applications.

8.1 Dijkstra big.LITTLE

Ainsi, dans le cas de Dijkstra, pour le cluster *little*, c'est-à-dire le Cortex-A7, on propose une configuration avec une I-cache et une D-cache de 16 kB. En effet, c'est à 16 kB de D-L1 que l'on obtient la meilleure efficacité énergétique ainsi que la meilleure efficacité surfacique, aussi bien pour Dijkstra *Small* que pour Dijkstra *Large*.

Il convient de souligner que ces résultats sont cohérents avec ceux obtenus lors de l'analyse de performance du Cortex-A7 face aux variations de la taille du cache pour Dijkstra. Néanmoins, une nuance mérite d'être prise en compte. Lors de l'analyse de Dijkstra *Small* et *Large*, on a observé que le gain d'IPC entre un L1 de 8 kB et un L1 de 16 kB n'est pas suffisamment significatif pour constituer un critère déterminant en termes de performance.

Cela s'explique par le fait qu'un point d'inflexion apparaît autour de 8 kB et qu'à partir de 16 kB la progression de l'IPC commence à saturer. Par conséquent, puisque les différences d'efficacité surfacique et d'efficacité énergétique entre 8 kB et 16 kB ne sont pas considérables (les valeurs étant très proches), la sélection entre ces deux tailles peut être laissée à d'autres critères de conception, notamment des considérations de coût pour le fabricant ou des besoins spécifiques. En effet, les performances en simulation sont très similaires et, de plus, les indicateurs d'efficacité surfacique et d'efficacité énergétique restent très proches l'un de l'autre.

Pour le cluster *big*, c'est-à-dire le processeur Cortex-A15, dans le cas particulier de l'application Dijkstra, on propose une configuration de 32 kB pour le cache d'instructions et le cache de données, principalement pour trois raisons.

Premièrement, c'est avec cette configuration que l'on atteint la meilleure efficacité surfacique parmi toutes les configurations évaluées pour cette application sur Cortex-A15. Deuxièmement,

il s'agit également de la configuration offrant la meilleure efficacité énergétique pour ce microprocesseur et cette application, ce qui est cohérent avec les résultats de performance obtenus pour Dijkstra *Large* et *Small*. Troisièmement, la hausse de l'IPC atteint 74,99% lorsque la taille du cache est de 32 kB, et l'on n'observe pas encore une saturation complète de la diminution des pertes, en particulier pour la D-cache. Or, pour cette configuration et sur ce microprocesseur, c'est précisément la réduction des pertes en D-cache qui constitue la source principale du gain d'IPC.

Par conséquent, cette configuration représente l'option recommandée : elle maximise non seulement le gain d'IPC, mais aussi la diminution des pertes en cache de données et en cache d'instructions, tout en maximisant l'efficacité énergétique et l'efficacité surfacique.

8.2 Blowfish big.LITTLE

Dans le cas spécifique des applications Blowfish, *Small* comme *Large*, sur le cluster *little* (Cortex-A7), on obtient une situation particulière. En effet, l'efficacité surfacique maximale est atteinte avec une configuration de cache de 8 kB, tandis que l'efficacité énergétique maximale, dans le cadre d'une puissance supposée constante à la fréquence maximale de fonctionnement du microprocesseur, est obtenue avec une configuration de 16 kB.

Par conséquent, un troisième critère est retenu pour sélectionner la configuration recommandée : le comportement en performance observé lors de l'analyse du microprocesseur face à cette application pour différentes tailles de cache. On avait montré qu'à partir de 8 kB, un point d'inflexion est atteint et qu'augmenter la taille du cache ne produit plus de gains significatifs en termes d'IPC. De plus, on observe une saturation à la fois de la diminution des pertes en I-cache et de la diminution des pertes en D-cache, ce qui rend peu pertinent, du point de vue des performances applicatives, d'intégrer une configuration à cache plus grand.

C'est pourquoi, pour Blowfish sur Cortex-A7, on recommande une configuration de 8 kB pour la I-cache et de 8 kB pour la D-cache.

Pour Blowfish sur le cluster *big* (Cortex-A15), la configuration recommandée est de 32 kB pour la I-cache et 32 kB pour la D-cache, pour trois raisons principales.

Premièrement, avec cette configuration, on obtient la meilleure efficacité surfacique, dans le cadre d'une puissance supposée constante à la fréquence maximale. Deuxièmement, cette même configuration présente aussi la meilleure efficacité énergétique pour cette application, aussi bien en version *Small* qu'en version *Large*. Enfin, ces résultats sont cohérents avec l'analyse de performance, qui montre que le gain d'IPC continue jusqu'à la configuration L1 de 32 kB, atteignant une amélioration de 41,68%. Bien que cette taille mette en évidence une certaine saturation de la diminution des pertes en D-cache et en I-cache, elle ne montre pas encore une saturation (ou un état de stagnation) du gain d'IPC.

Par conséquent, il reste pertinent, en termes de performance, de mettre en œuvre un cache de 32 kB plutôt qu'un cache de 16 kB. Ce dernier n'apporte qu'un gain d'IPC de 31,017%, soit environ 10 points de pourcentage de moins. L'ensemble de ces éléments renforce ainsi la décision retenue précédemment.

Dans le cas des configurations du cluster *big*, elles sont équivalentes pour Blowfish comme pour Dijkstra. Cela s'explique par le fait que, dans les deux applications, le goulot d'étranglement se situait principalement au niveau des instructions de calcul entier. Ainsi, l'implémentation du Cortex-A15 et sa capacité à traiter davantage d'instructions en parallèle permettent des améliorations d'IPC considérables, ce qui rend la dynamique d'amélioration des accès aux données en mémoire plus visible.

Même si la nature des structures de données manipulées et les modes d'accès à la mémoire

diffèrent entre les deux algorithmes, les résultats ne sont pas nécessairement identiques, mais ils suivent un comportement incrémental du même type jusqu'à atteindre la configuration optimale retenue, à savoir 32 kB. Néanmoins, ces différences de structures et de schémas d'accès montrent qu'après l'adoption de ce microprocesseur, les gains d'IPC obtenus en augmentant la taille du cache sont plus représentatifs pour Dijkstra que pour Blowfish. Malgré cela, les deux applications conservent la même taille optimale de cache, c'est-à-dire la configuration à 32 kB.

De son côté, pour le cluster *little*, la configuration de cache recommandée n'est pas la même, en se basant principalement sur l'efficacité surfacique. En effet, pour Blowfish *Small* et *Large* sur Cortex-A7, l'efficacité surfacique est plus élevée lorsque la configuration du cache est de 8 kB, ce qui diffère des résultats obtenus pour Dijkstra sur ce même microprocesseur. Par conséquent, la recommandation initiale est différente.

Néanmoins, les performances obtenues avec les configurations à 8 kB restent comparables, et l'on pourrait même envisager l'utilisation d'une configuration de 8 kB pour Dijkstra. En effet, en termes de performance, une fois ce point d'inflexion dépassé, les gains d'IPC tendent à se saturer, ce qui rend les différences entre 8 kB et des tailles supérieures moins marquées.

Le point d'inflexion observé pour Dijkstra autour de 8 kB sur Cortex-A7 s'interprète comme une forme de saturation des gains d'IPC. Par conséquent, l'écart entre une configuration de 8 kB et une configuration de 16 kB n'est pas représentatif en termes de performance. De plus, l'efficacité énergétique demeure également comparable entre ces deux tailles.

Ainsi, même si la configuration recommandée n'est pas strictement équivalente, il serait envisageable d'implémenter Dijkstra avec une configuration de 8 kB sur Cortex-A7, puisqu'elle n'entraînerait ni un coût important en performance, ni un coût notable en efficacité énergétique, par comparaison avec la configuration recommandée.

9 Méthodologie de Spécification Architecturale

Pour spécifier une architecture dédiée à un nouveau domaine (par exemple l'intelligence artificielle ou le traitement vidéo), nous proposons la méthodologie rigoureuse suivante, dérivée de l'expérience acquise lors de ce TP :

1. **Caractérisation de la charge (*profiling*).** Utiliser des outils comme Gem5 (mode atomique) pour extraire le "mix d'instructions".

   Objectif : identifier si l'application est *compute-bound* (nécessitant des ALU/FPU plus larges) ou *memory-bound* (nécessitant des caches et de la bande passante).
2. **Exploration de l'espace de conception (*Design Space Exploration*, DSE).** Automatiser les simulations (via des scripts Python/Bash comme dans ce TP) en faisant varier les paramètres critiques : taille de cache, largeur de pipeline (*issue width*), prédicteurs de branchement. Mesurer l'IPC pour chaque point.
3. **Contraintes physiques (PPA : *Power, Performance, Area*).** Modéliser chaque configuration retenue avec un outil physique comme CACTI 7.0 (pour la surface et la latence) et McPAT (pour la puissance). Rejeter les configurations irréalistes (par exemple, latence > 1 cycle).
4. **Analyse de Pareto et choix final.** Tracer les courbes de compromis (par exemple, performance vs surface). Sélectionner les points du front de Pareto : ceux qui offrent le meilleur gain de performance pour un coût marginal en surface/ènergie acceptable. À titre d'exemple dans ce TP, le passage de 16 kB à 32 kB sur le Cortex-A15 était sur le front de Pareto (gain très important), alors que sur le Cortex-A7 il ne l'était pas (gain négligeable).

10 Compromis et conclusion

Les configurations proposées ne sont pas strictement équivalentes au sens où elles ne mènent pas au même choix de taille $L1$ pour toutes les applications et pour tous les clusters. Le dimensionnement optimal dépend à la fois (i) de la nature de l'application et de sa sensibilité à la hiérarchie mémoire, et (ii) de l'objectif de conception du cluster considéré (*little* orienté coût/énergie vs *big* orienté performance). Ainsi, on observe que Dijkstra et Blowfish ne conduisent pas aux mêmes compromis sur Cortex-A7, alors qu'elles convergent vers une même configuration sur Cortex-A15.

Pour Dijkstra sur le cluster *little* (Cortex-A7), la configuration retenue est 16 kB en I-cache et 16 kB en D-cache, car elle maximise l'efficacité énergétique et l'efficacité surfacique pour les versions *Small* et *Large*. Néanmoins, l'analyse de performance met en évidence un point d'inflexion autour de 8 kB : le gain d'IPC entre 8 kB et 16 kB reste faible et la progression tend à saturer. Par conséquent, un compromis alternatif cohérent consiste à considérer 8 kB comme une option possible si d'autres contraintes dominent (coût de fabrication, budget surface, intégration), puisque les performances simulées et les indicateurs d'efficacité restent proches entre 8 kB et 16 kB. Autrement dit, 16 kB est recommandé par les métriques d'efficacité, mais 8 kB peut être défendu par l'argument de saturation de l'IPC et par une logique de minimisation de ressources sur le cluster *little*.

Pour Dijkstra sur le cluster *big* (Cortex-A15), la configuration recommandée est 32 kB pour la I-cache et 32 kB pour la D-cache. Ce choix est motivé par le fait qu'il fournit simultanément la meilleure efficacité surfacique et la meilleure efficacité énergétique parmi les configurations évaluées, tout en maximisant le gain d'IPC. De plus, la réduction des pertes, en particulier en D-cache, constitue la source principale du gain d'IPC, et la diminution de ces pertes n'apparaît pas totalement saturée à 32 kB, ce qui renforce la pertinence de cette taille sur le cluster *big*.

Pour Blowfish sur le cluster *little* (Cortex-A7), les critères d'efficacité surfacique et énergétique pointent vers des tailles différentes (respectivement 8 kB et 16 kB). Le choix final se fait alors en intégrant le comportement en performance : l'IPC présente un point d'inflexion dès 8 kB et l'augmentation de taille au-delà n'apporte plus de gains significatifs, tandis que la diminution des pertes en I-cache et D-cache sature également. Par conséquent, 8 kB en I-cache et 8 kB en D-cache est retenu comme compromis le plus rationnel au regard des performances applicatives et du rôle « frugal » attendu du cluster *little*. Pour Blowfish sur le cluster *big* (Cortex-A15), la recommandation est à nouveau 32 kB/32 kB, car cette taille maximise les efficacités surfacique et énergétique et reste cohérente avec une hausse d'IPC qui continue jusqu'à 32 kB, supérieure à celle obtenue avec 16 kB.

En conclusion, les configurations proposées ne sont donc pas équivalentes au sens d'un choix unique et invariant : sur Cortex-A7, Dijkstra pousse vers 16 kB (avec l'alternative défendable de 8 kB compte tenu de la saturation de l'IPC), tandis que Blowfish se contente de 8 kB ; sur Cortex-A15, les deux applications convergent vers 32 kB. Ce résultat est cohérent avec l'idée que Dijkstra demeure plus sensible à la hiérarchie mémoire et que Blowfish atteint plus rapidement un plateau de performance, et il justifie une stratégie big.LITTLE où le cluster *big* est dimensionné pour capter les gains de performance, tandis que le cluster *little* privilégie des tailles proches du point d'inflexion afin d'optimiser le coût et l'efficacité.

Références

[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.