

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

ahdubwyfw i 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 1.

Ainsi, pour la configuration C1, par exemple, le cache d'instructions possède $\frac{4096}{32} = 128$ lignes, d'où `i1l:128:32:1:1`. En utilisant la même logique pour les autres caches et configurations, le tableau suivant est rempli.

| Configuration | IL1 | DL1 | UL2 |
|---------------|----------------|----------------|-----------------|
| C1 | i1l:128:32:1:1 | dl1:128:32:1:1 | ul2:1024:32:1:1 |
| C2 | i1l:128:32:1:1 | dl1:64:32:2:1 | ul2:256:32:4:1 |

B. Taux de défauts dans les différentes caches

- Le taux de défauts dans le cache d'instructions i1l : `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) | 45 354 411 [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 V: 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 VI: 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 VII: 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 VIII: Blowfish large (Cortex-A7 vs Cortex-A15)

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 goulet 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) :

$$TPC = \frac{\text{simTicks}}{\text{numCycles}} \quad (1)$$

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

$$f_{\text{cpu}} = \frac{\text{simFreq}}{\text{TPC}} \quad (2)$$

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

$$P_{L1} = \frac{L_{L1}}{\text{TPC}} \quad [\text{cycles/miss}] \quad (3)$$

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) :

$$P_{L2} = \frac{L_{L2}}{\text{TPC}} \quad [\text{cycles/miss}] \quad (4)$$

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) :

$$\text{MPKI} = \frac{\text{misses}}{\text{numInsts}} \cdot 1000 \quad (5)$$

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) :

$$\Delta \text{CPI}_{L1} \approx \frac{\text{MPKI}_{L1}}{1000} \cdot P_{L1} \quad (6)$$

Les résultats du *profiling*, combinés avec le coût des opérations mémoire dans les deux applications, indiquent que le goulet d'étranglement principal se situe davantage du côté du cœur (fenêtre OoO/ROB, files mémoire, dépendances et contrôle de flux). Par conséquent, une micro-architecture disposant de davantage d'ALU entières, ou d'un parallélisme

| 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 IX: 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 X: Contribution estimée de la pénalité mémoire au CPI pour les exécutions de Blowfish

accru sur les opérations entières, 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.

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 précharge (*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*.

A. Impact de la taille des caches L1

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.

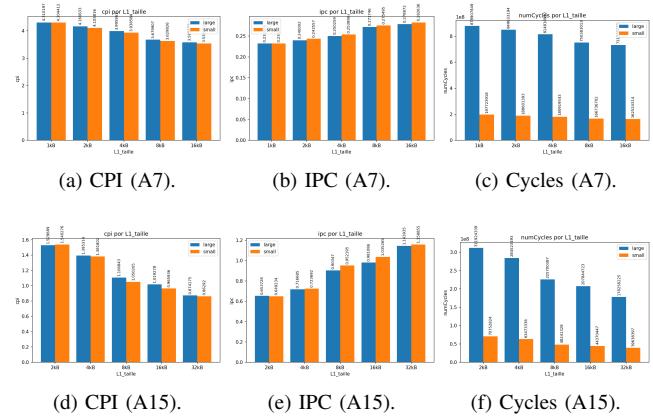


Fig. 1: Dijkstra : performance générale en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

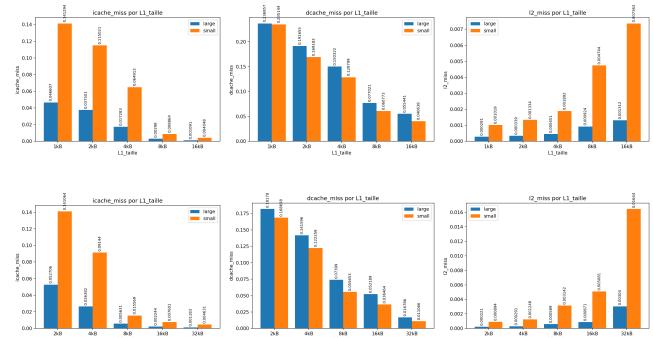


Fig. 2: Dijkstra : taux de défauts I-Cache, D-Cache et L2 en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

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

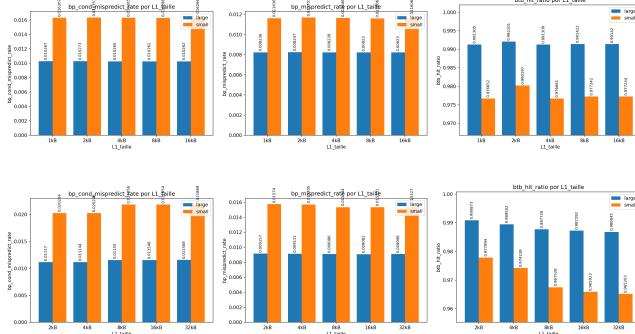


Fig. 3: Dijkstra : métriques de prédition de branchement en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

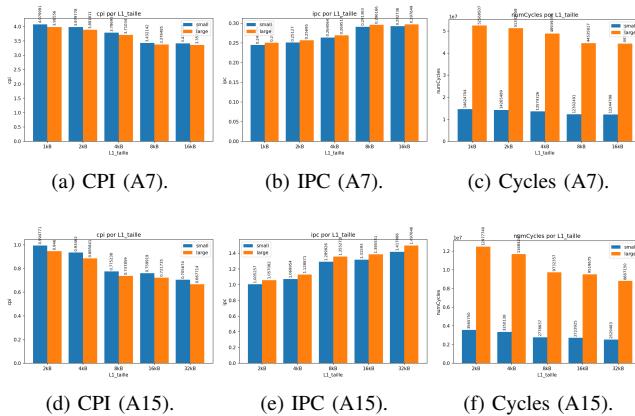


Fig. 4: Blowfish : performance générale en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

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

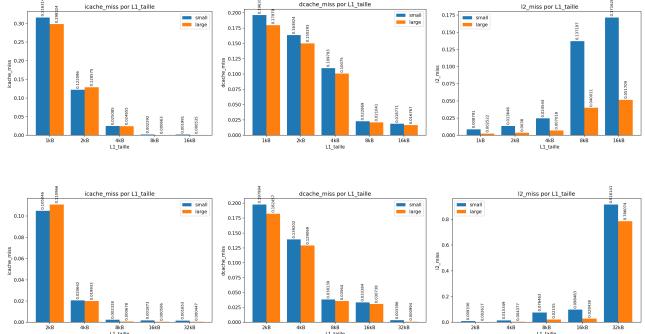


Fig. 5: Blowfish : taux de défauts I-Cache, D-Cache et L2 en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

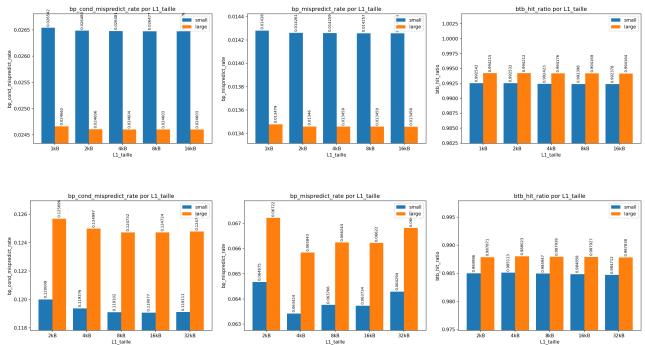


Fig. 6: Blowfish : métriques de prédition de branchement en fonction de la taille du cache L1 (A7 en haut, A15 en bas).

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'infexion 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.

Pour le Cortex-A15, les tendances observées, une fois l'optimisation du traitement des opérations du cœur déjà mise

| 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 XI: 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 XII: Dijkstra large (Cortex-A7) : gains relatifs par rapport à 1 kB.

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

| 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 XIII: 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 XIV: Dijkstra large (Cortex-A15) : gains relatifs par rapport à 2 kB.

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 précharge 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'infexion 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.

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

| 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 XV: 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 XVI: Blowfish large (Cortex-A7) : gains relatifs par rapport à 1 kB.

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.

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 × width (mm)`. En supposant les tailles du Tableau 12 : $L1 = L1D = 16 \text{ kB}$ (A7) et 32 kB (A15), on obtient :

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

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

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

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),

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

| 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 XVII: 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 XVIII: Blowfish large (Cortex-A15) : gains relatifs par rapport à 2 kB.

Pour chaque taille L1, on utilise :

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

Note : CACTI ne trouve pas d'organisation valide pour A7 à 1 kB (d'où “-”).

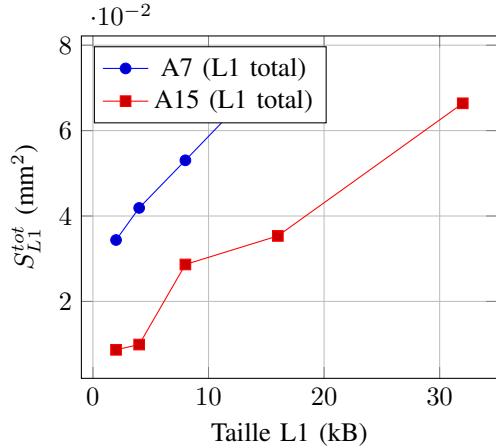


Fig. 7: Surface totale des L1 (I+D) en fonction de la taille.

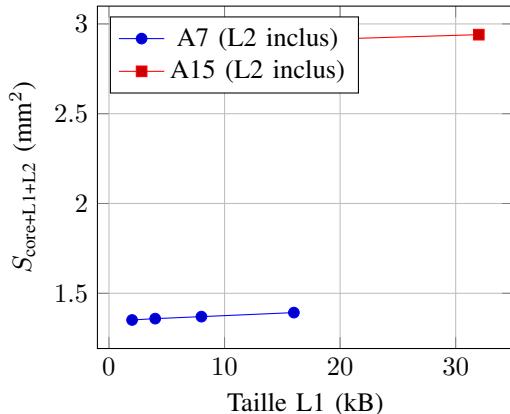


Fig. 8: Surface totale des coeurs (L1 variable + L2=512 kB).

b) Graphes (pgfplots).:

| 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 |
| 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 |

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 $S_{L2}(A7) = 0,94241$ mm² et $S_{L2}(A15) = 0,94019$ mm². Les valeurs suivantes sont donc en IPC/mm^2 .

| L1 (kB) | Dijkstra small | Dijkstra large | Blowfish small | Blowfish large |
|------------------|----------------|----------------|----------------|----------------|
| Cortex-A7 | | | | |
| 1 | — | — | — | — |
| 2 | 0,1787 | 0,1778 | 0,1859 | 0,1902 |
| 4 | 0,1860 | 0,1846 | 0,1943 | 0,1983 |
| 8 | 0,2006 | 0,1989 | 0,2127 | 0,2162 |
| 16 | 0,2025 | 0,2009 | 0,2102 | 0,2138 |

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

| L1 (kB) | Dijkstra small | Dijkstra large | Blowfish small | Blowfish large |
|-------------------|----------------|----------------|----------------|----------------|
| Cortex-A15 | | | | |
| 2 | 0,2249 | 0,2269 | 0,3487 | 0,3667 |
| 4 | 0,2516 | 0,2500 | 0,3710 | 0,3915 |
| 8 | 0,3253 | 0,3129 | 0,4444 | 0,4669 |
| 16 | 0,3516 | 0,3386 | 0,4523 | 0,4763 |
| 32 | 0,3908 | 0,3907 | 0,4820 | 0,5094 |

TABLE XX: 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 $S_{L2}(A7) = 0,94241$ mm² et $S_{L2}(A15) = 0,94019$ mm². Les valeurs suivantes sont donc en IPC/mm^2 .

Note : CACTI ne fournit pas d'organisation valide pour A7 à 1 kB, d'où “-”.

| L1 (kB) | Dijkstra small | Dijkstra large | Blowfish small | Blowfish large |
|------------------|----------------|----------------|----------------|----------------|
| Cortex-A7 | | | | |
| 1 | — | — | — | — |
| 2 | 0,1787 | 0,1778 | 0,1859 | 0,1902 |
| 4 | 0,1860 | 0,1846 | 0,1943 | 0,1983 |
| 8 | 0,2006 | 0,1989 | 0,2127 | 0,2162 |
| 16 | 0,2025 | 0,2009 | 0,2102 | 0,2138 |

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

| L1 (kB) | Dijkstra small | Dijkstra large | Blowfish small | Blowfish large |
|-------------------|----------------|----------------|----------------|----------------|
| Cortex-A15 | | | | |
| 2 | 0,2249 | 0,2269 | 0,3487 | 0,3667 |
| 4 | 0,2516 | 0,2500 | 0,3710 | 0,3915 |
| 8 | 0,3253 | 0,3129 | 0,4444 | 0,4669 |
| 16 | 0,3516 | 0,3386 | 0,4523 | 0,4763 |
| 32 | 0,3908 | 0,3907 | 0,4820 | 0,5094 |

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

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.