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

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

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

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

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

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

I. Introduction

ahdubwyfwi ceuce hyec

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

A. Paramètres de configuration des caches utilisés dans gem5

TABLE I

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

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

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

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

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

B. Taux de défauts dans les différentes caches

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

TABLE II: icache.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 III: dcache.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 IV: l2cache.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)	$43,65\%$	$42,02\%$

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

caches (Table II). Cela signifie que le flux d'instructions tient entièrement dans le cache d'instructions et que les boucles de multiplication de matrices réutilisent toujours le même petit bloc de code, ce qui exploite très bien la localité spatiale et temporelle du code.

III. Profiling de l'application

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

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

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

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

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

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

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

Le *profiling* montre que le calcul entier domine dans tous les cas : $\approx 60\,\%$ pour Dijkstra et $\approx 72\,\%$ pour Blowfish. Ainsi, un processeur doté de davantage d'ALU pour paralléliser ces opérations pourrait offrir un gain notable. De plus, une restructuration des boucles afin de réduire les branchements, également significatifs, pourrait améliorer les performances. Enfin, l'optimisation des accès mémoire (notamment la localité et les accès contigus) est pertinente, car les *loads* dépassent $20\,\%$. Ces constats orientent directement les choix de micro-architecture et d'optimisation logicielle.

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.