École Nationale Supérieure de Techniques Avancées

Rapport projet ES201 : Architecture des microprocesseurs

Réalisé par:

Davide Brambilla
Alice Phe
Manel Ben Youssef
Aymeric Basset

Année universitaire: 2018/2019

Table des matières

1 Présentation du TP

Il existe de nombreux types de composants microprocesseurs actuellement en vente sur le marché. Les gammes s'élargissent de plus en plus, jouant toujours entre les performances de ces derniers et les consommations énergétiques (ainsi que la surface du composant).Le but de ce TP est d'analyser les performances de deux types de coeurs ARM : Cortex A7 et Cortex A15.

2 Profiling de l'application

Nous allons effectuer un profiling de l'application Dijkstra et BlowFish en utilisant *sim-profile*. Pour compiler les programmes nous avons utilisés le commande :
sslittle-na-sstrix-gcc <progr>.c -o <progr>.ss.

2.1 Q1

Tout d'abord, nous avons généré le pourcentage de chaque classe d'instructions de ces applications avec le commande :
sim-profile -iclass <progr>.ss :

Dijkstra_small avec input.dat

Classes des instructions	Nombre d'instructions	Pourcentage
load	26485327	28.78
store	6383433	6.94
uncond branch	5382776	5.85
cond branch	9396219	10.21
int computation	44369686	48.22
fp compuation	0	0.00
trap	239	0.00

BlowFish avec imput_small.asc

Classes des instructions	Nombre d'instructions	Pourcentage
load	623	8.12
store	3501	45.61
uncond branch	167	2.18
cond branch	873	11.37
int computation	2504	32.62
fp compuation	0	0.00
trap	8	0.10

2.2 Q2

Pour le programme Dijkstra, la classe d'instruction la plus gourmande en processus est **int computation**(48.22%).Cela peut s'expliquer par l'utilisation des boucles **for** dans le programme qui néccessite plus de calculs que le reste. Afin d'améliorer cela, on peut utiliser

un processeur superscalaire avec plusieurs ALUs pour paralléliser les calculs (avec un bon prédicteur de branchements etc...).

Pour le programme BlowFish, on retrouve aussi une part importante de calcul du même type que ceux analysés dans Dijkstra, cependant on voit apparaître une nouvelle classe prédominante qui est **store**. Ainsi ce programme a en grande partie des instructions d'accès à la mémoire, et plus précisément de stockage.

Enfin pour comparer les deux programmes : Dijkstra est plutôt **CPU bound** où l'on doit améliorer la classe int computation ie la vitesse du processeur, tandis que BlowFish serait plutôt **memory bound** où l'on doit améliorer store, ie l'accès mémoire.

2.3 Q3

Classes des instructions	Dijkstra_ small	BlowFish	Produit de Polynômes	SSCA2-BCS	SHA-1
load	28.78	8.12	13.30	37.11	16.72
store	6.94	45.61	6.28	9.23	8.12
uncond branch	5.85	2.18	7.96	5.95	0.08
cond branch	10.21	11.37	16.13	6.65	6.10
int computation	48.22	32.62	45.76	37.62	68.99
fp compuation	0	0.00	10.56	3.44	0
trap	0	0.10	0	0.12	0

En utilisant les valeurs obtenues au TP2 pour le produit du polynômes et pour SSCA2-BCS et en calculant les performances du SHA-1, on peut observer que :

- Pour tous les algorithmes, la classe d'instruction la plus proéminente est celle des calculs sur les entiers (aux alentours de 50%), ce qui s'explique par l'utilisation des boucles for comme dit précédemment ;
- Dans BlowFish, on a une considérable partie des instructions d'écriture en mémoire que nous n'avons pas dans les autres opérations ;
- Le seul programme qui a un considérable utilisation des instructions pour les calculs sur des types float est *Produit de Polynômes*.

3 Evaluation Des Performances

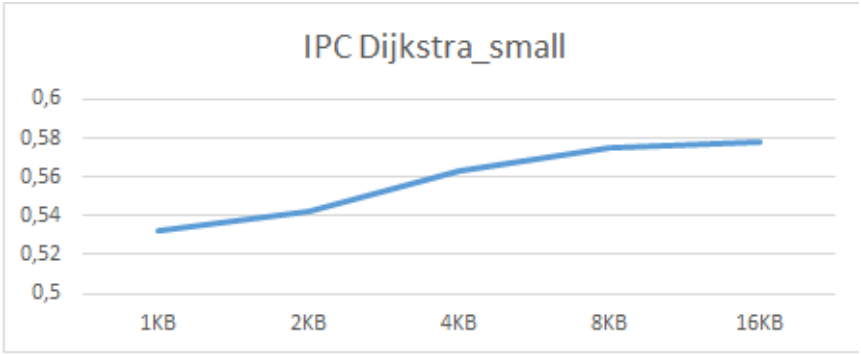
3.1 Q4

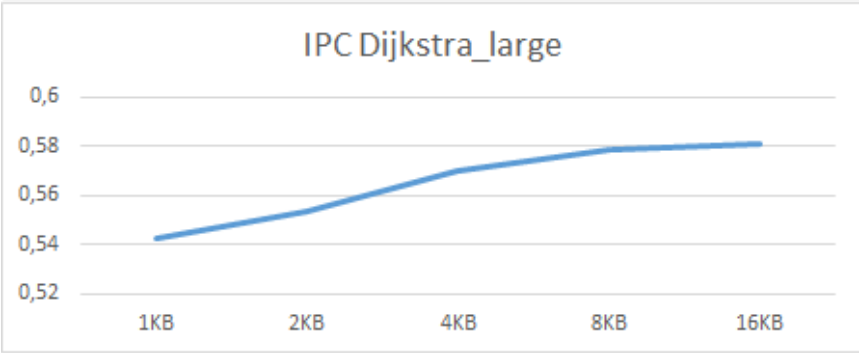
Pour cette partie nous avons essayé d'automatiser la récupération de l'output de notre commande à l'aide d'un script bash.

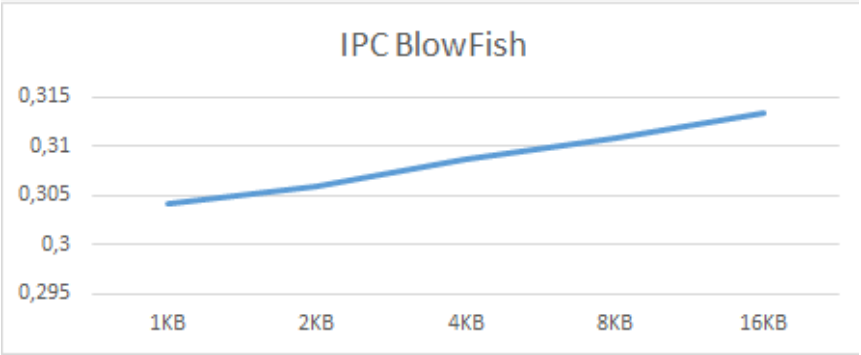
Cortex A7 A taille de cache L2 fixe (512KB) et en faisant varier simultanément la taille des caches L1 d'instructions et de données, nous avons effectué une évaluation des performances en utilisant le simulateur **sim-outorder**. Vu que les valeurs de taille du bloc et d'associativité étaient données, nous avons pu faire varier le nombre d'ensemble du cache (<**nsets**>) et on a obtenu que pour avoir une taille du cache de 1KB, 2KB, 4KB, 8KB, 16KB on devait respectivement avoir un nombre d'ensemble égal a 16, 32, 64, 128 et 256.
Les paramètres d'exécutions utilisés ont été choisis pour simuler le comportement de A7 :

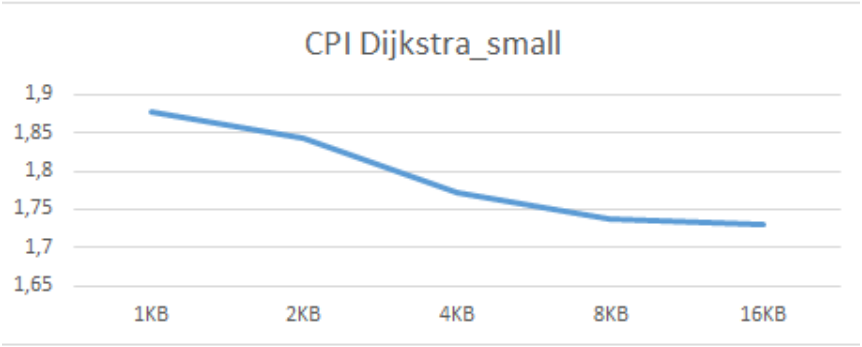
- -bpred bimod -bpred :bimod 256
- -fetch :mplat 8
- -fetch :ifqsize 4
- -decode :width 2
- -issue :width 4
- -commit :width 2
- -ruu :size 2
- -lsq :size 8
- -res :ialu 1
- -res :fpalu 1
- -res :imult 1
- -res :fpmult 1
- -cache :il1 il1 :<**nsets**> :32 :2 :l
- -cache :dl1 dl1 :<**nsets**> :32 :2 :l
- -cache :il2 dl2 -cache :dl2 dl2 :2048 :32 :8 :l

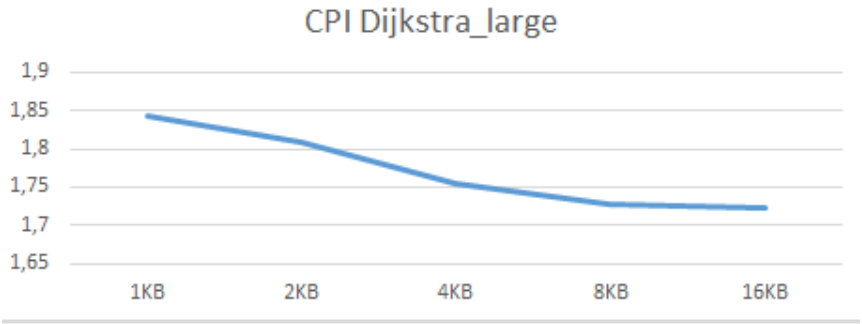
Pour l'évaluation des performances on a choisi d'évaluer *IPC*, *CPI*, le *nombre des cycles et des instructions*, le *miss rate* des mémoires cache et *l'address-prediction rate* et le *direction-prediction rate* pour évaluer les performances du branchement.
Nous avons obtenu les graphes de performance suivant :

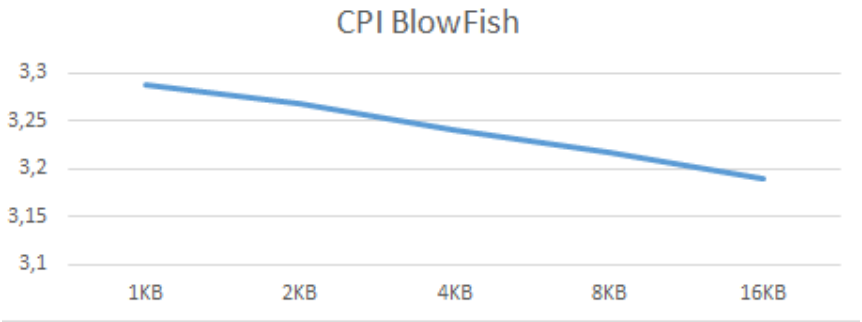

IPC Dijkstra_small
0,6
0,58
0,56
0,54
0,52
0,5
1KB
2KB
4KB
8KB
16KB

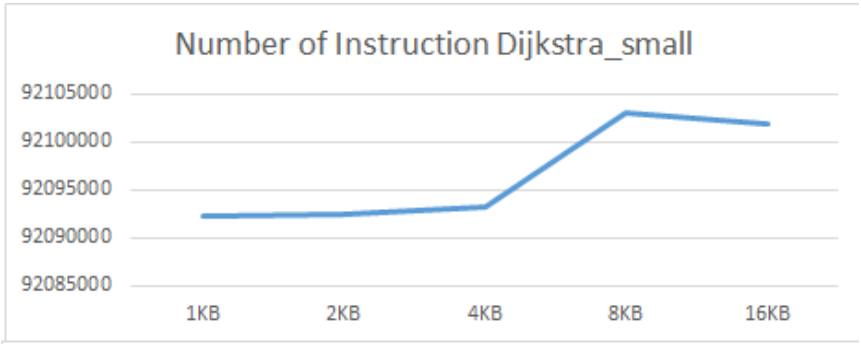

IPC Dijkstra_large
0,6
0,58
0,56
0,54
0,52
1KB
2KB
4KB
8KB
16KB

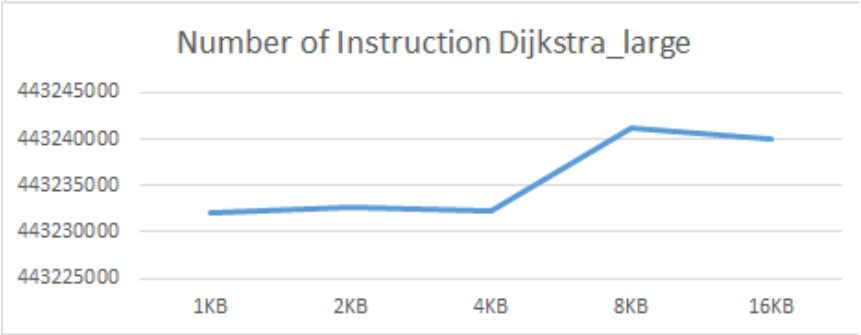

IPC BlowFish
0,315
0,31
0,305
0,3
0,295
1KB
2KB
4KB
8KB
16KB

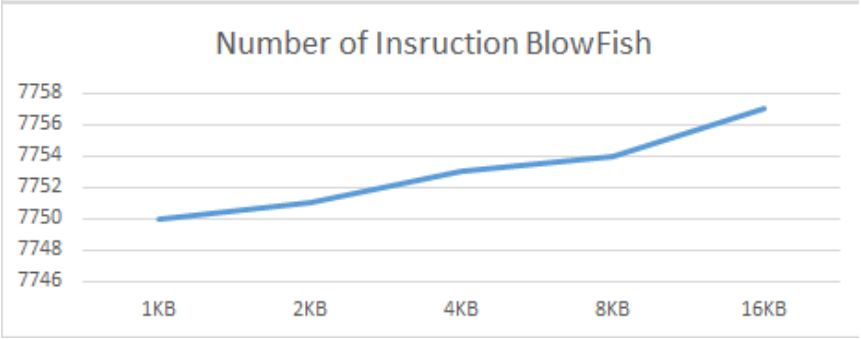

CPI Dijkstra_small
1,9
1,85
1,8
1,75
1,7
1,65
1KB
2KB
4KB
8KB
16KB

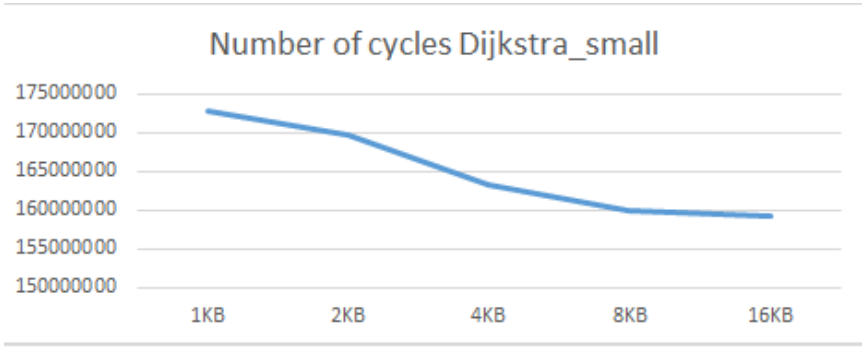

CPI Dijkstra_large
1,9
1,85
1,8
1,75
1,7
1,65
1KB
2KB
4KB
8KB
16KB

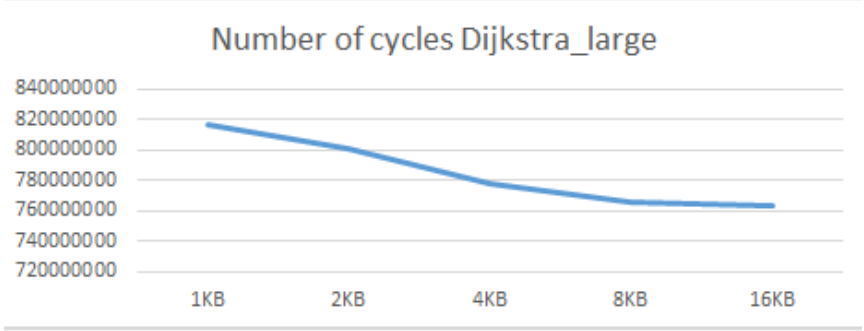

CPI BlowFish
3,3
3,25
3,2
3,15
3,1
1KB
2KB
4KB
8KB
16KB

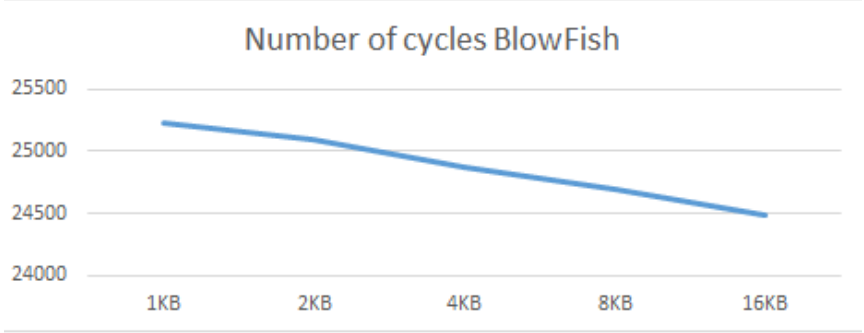

Number of Instruction Dijkstra_small
92105000
92100000
92095000
92090000
92085000
1KB
2KB
4KB
8KB
16KB

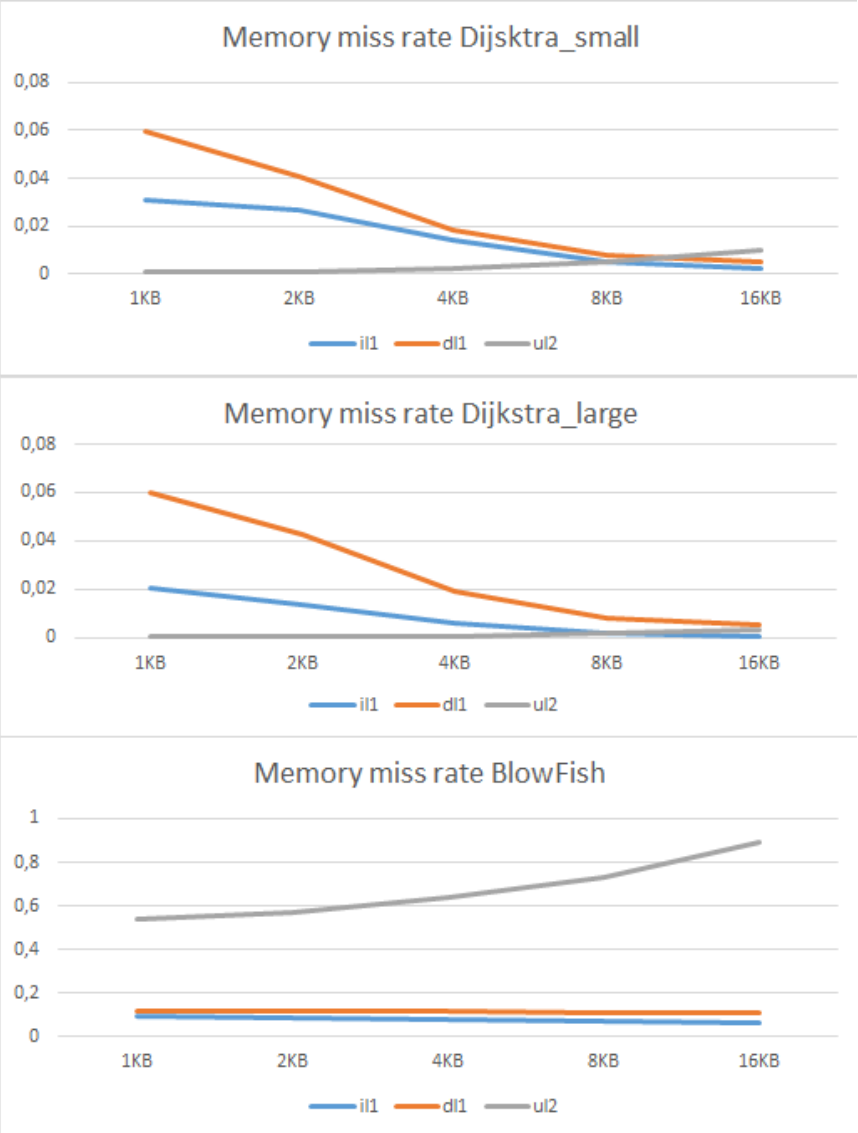

Number of Instruction Dijkstra_large
443245000
443240000
443235000
443230000
443225000
1KB
2KB
4KB
8KB
16KB

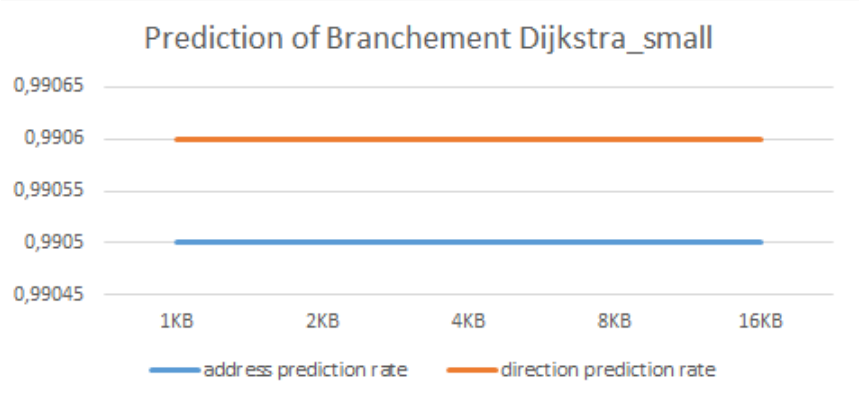

Number of Insruction BlowFish
7758
7756
7754
7752
7750
7748
7746
1KB
2KB
4KB
8KB
16KB

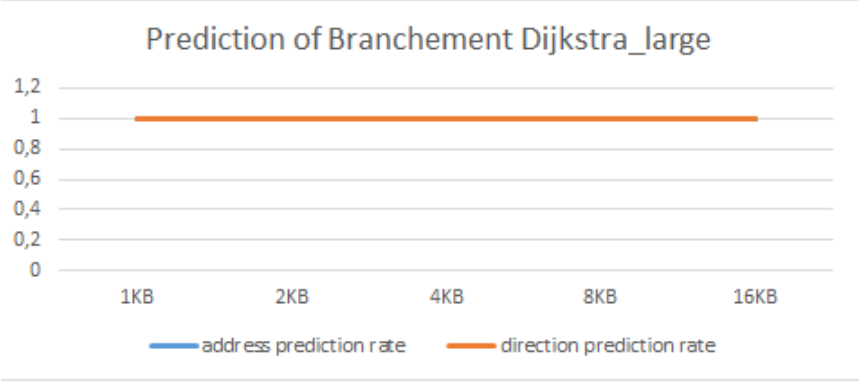

Number of cycles Dijkstra_small
175000000
170000000
165000000
160000000
155000000
150000000
1KB
2KB
4KB
8KB
16KB

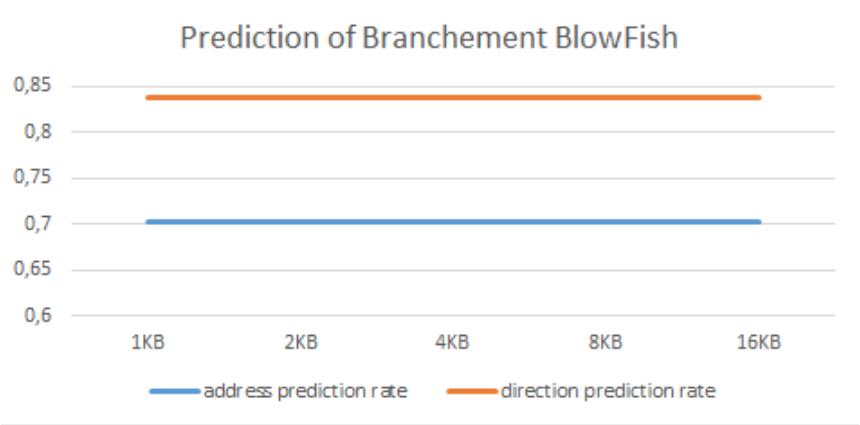

Number of cycles Dijkstra_large
840000000
820000000
800000000
780000000
760000000
740000000
720000000
1KB
2KB
4KB
8KB
16KB


Number of cycles BlowFish
25500
25000
24500
24000
1KB
2KB
4KB
8KB
16KB


Memory miss rate Dijsktra_small
0,08
0,06
0,04
0,02
0
1KB
2KB
4KB
8KB
16KB
il1
dl1
ul2
Memory miss rate Dijkstra_large
0,08
0,06
0,04
0,02
0
1KB
2KB
4KB
8KB
16KB
il1
dl1
ul2
Memory miss rate BlowFish
1
0,8
0,6
0,4
0,2
0
1KB
2KB
4KB
8KB
16KB
il1
dl1
ul2


Prediction of Branchement Dijkstra_small
0,99065
0,9906
0,99055
0,9905
0,99045
1KB
2KB
4KB
8KB
16KB
address prediction rate
direction prediction rate


Prediction of Branchement Dijkstra_large
1,2
1
0,8
0,6
0,4
0,2
0
1KB
2KB
4KB
8KB
16KB
address prediction rate
direction prediction rate


Prediction of Branchement BlowFish
0,85
0,8
0,75
0,7
0,65
0,6
1KB
2KB
4KB
8KB
16KB
address prediction rate
direction prediction rate

Conclusions Pour les performances de la mémoire *cache* nous voyons qu'une taille plus grande du cache L1 permet un *miss-rate* réduit. Pour le *branchement*, nous n'avons pas noté un changement de performances. En général pour le **CortexA7**, nous avons pu conclure que avoir à disposition une taille de mémoire cache plus grande améliore les performances. Pour *Dijkstra* nous n'avons pas observé de grandes améliorations entre une taille de L1 de 8KB et une taille de 16KB donc nous pensons que la meilleure taille est 8KB en considérant que elle occupe moins de surface. Pour *BlowFish* la meilleure taille est de 16KB puisqu'elle présente le meilleur CPI.

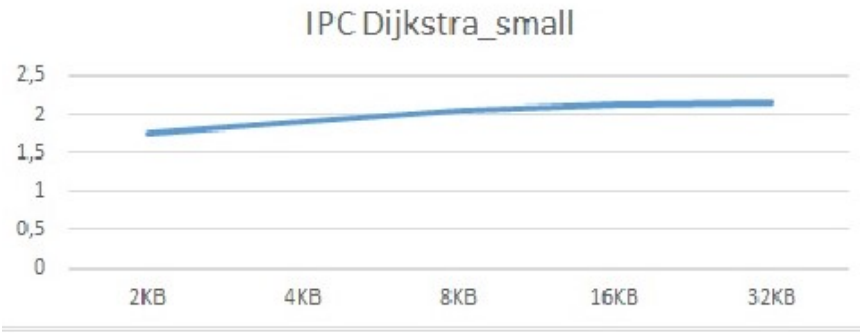
3.2 Q5

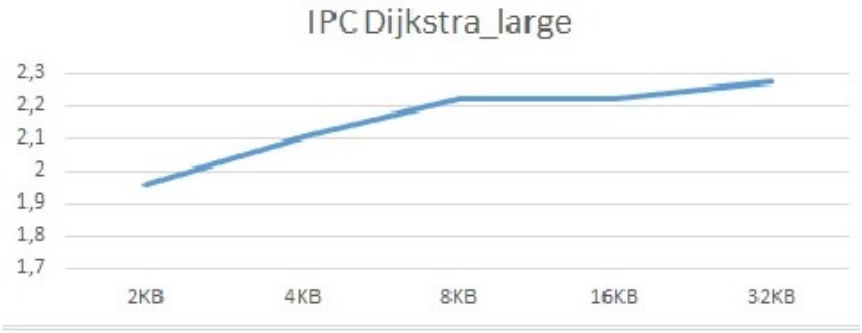
Cortex A15 Selon le même principe que pour la question 4, nous avons simulé le comportement des différents applications avec le commande *sim-outorder* Les paramètres d'exécution utilisés ici sont :

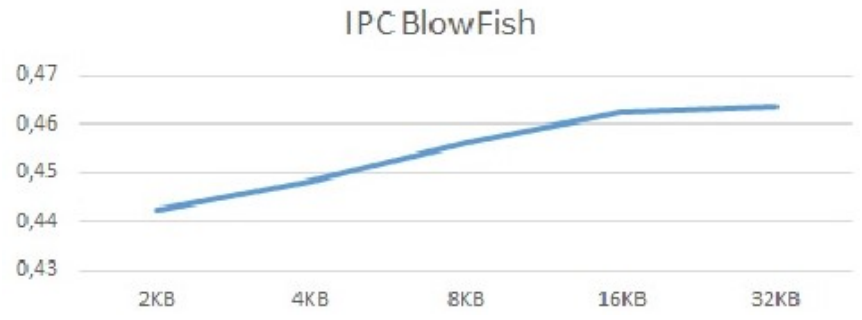
- -bpred 2lev -bpred :bimod 256
- -fetch :mplat 15
- -fetch :ifqsize 8
- -decode :width 4
- -issue :width 8
- -commit :width 4
- -ruu :size 16
- -lsq :size 16
- -res :ialu 5
- -res :fpalu 1
- -res :imult 1
- -res :fpmult 1
- -cache :il1 il1 :<**nsets**> :64 :2 :l
- -cache :dl1 dl1 :<**nsets**> :64 :2 :l
- -cache :il2 dl2 -cache :dl2 dl2 :512 :64 :16 :l

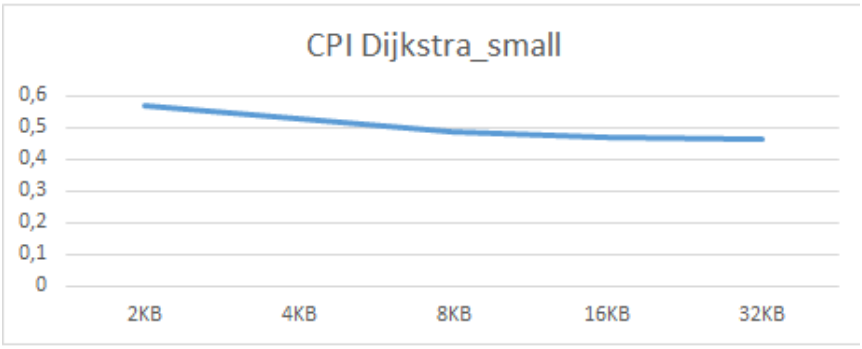
Les valeurs utilisées pour <**nsets**> sont encore 16, 32, 64, 128, 256 puisque que la question est de faire varier simultanément la taille des caches L1 d'instructions et de données de 2KB à 32 KB.

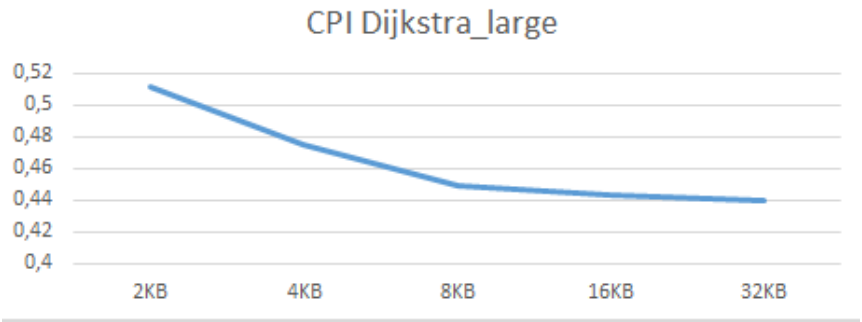
Nous avons obtenu les graphes de performance suivants :

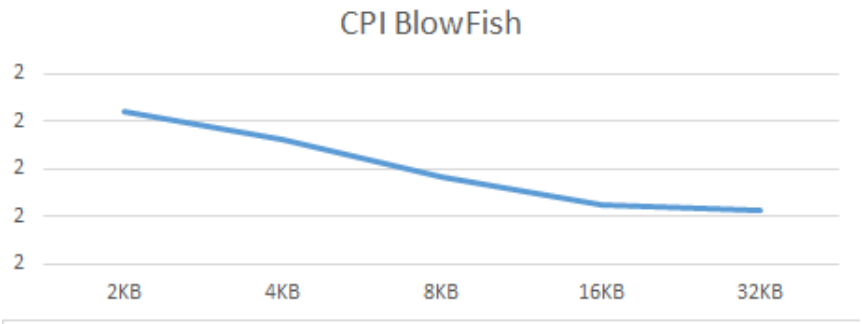

IPC Dijkstra_small
2,5
2
1,5
1
0,5
0
2KB
4KB
8KB
16KB
32KB

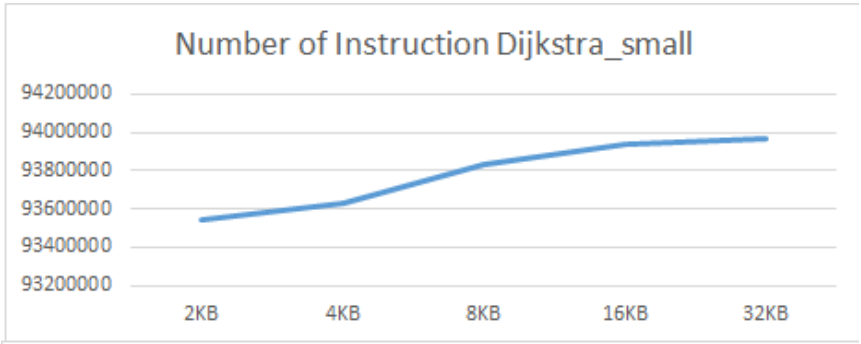

IPC Dijkstra_large
2,3
2,2
2,1
2
1,9
1,8
1,7
2KB
4KB
8KB
16KB
32KB

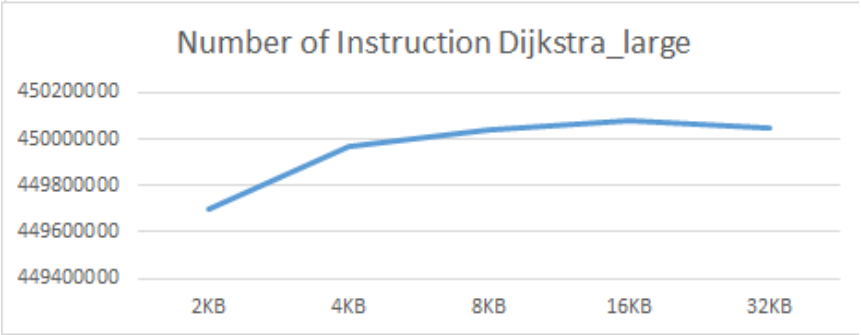

IPC BlowFish
0,47
0,46
0,45
0,44
0,43
2KB
4KB
8KB
16KB
32KB

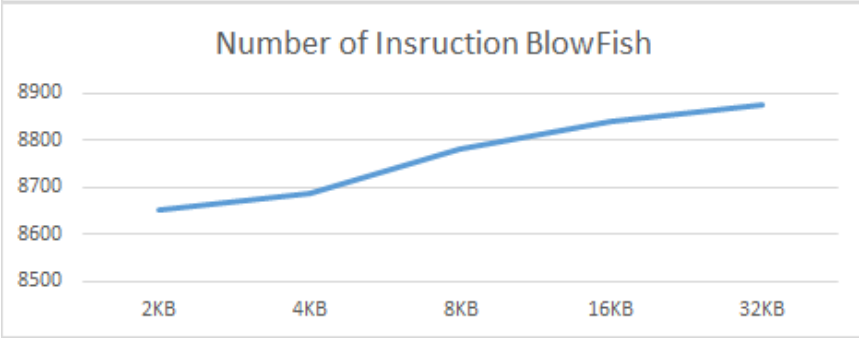

CPI Dijkstra_small
0,6
0,5
0,4
0,3
0,2
0,1
0
2KB
4KB
8KB
16KB
32KB

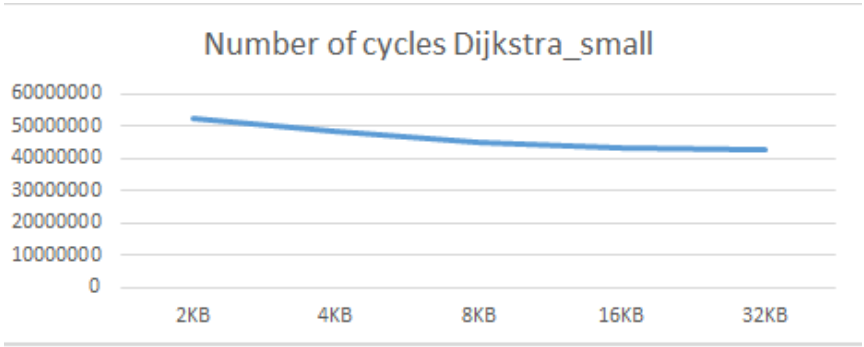

CPI Dijkstra_large
0,52
0,5
0,48
0,46
0,44
0,42
0,4
2KB
4KB
8KB
16KB
32KB

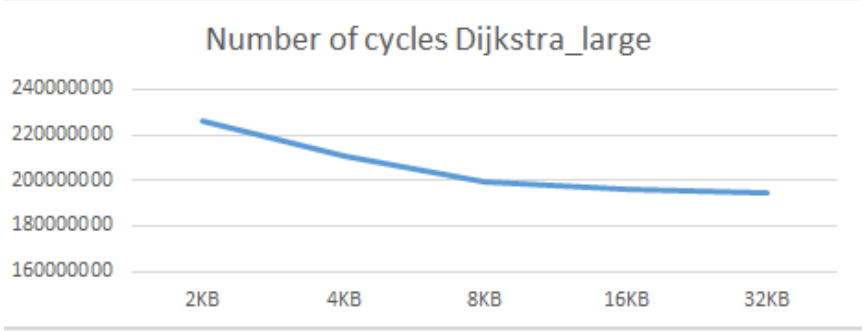

CPI BlowFish
2
2
2
2
2
2KB
4KB
8KB
16KB
32KB

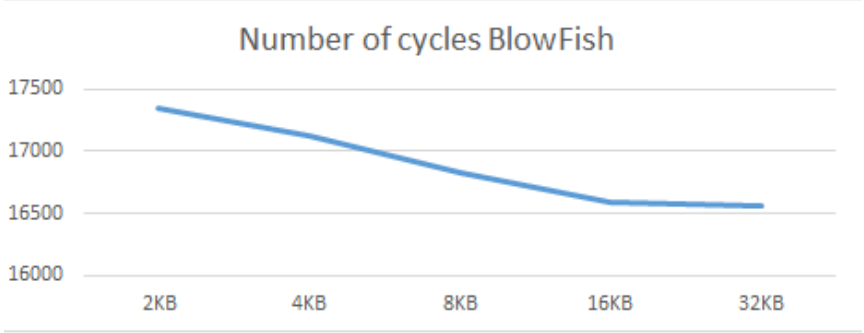

Number of Instruction Dijkstra_small
94200000
94000000
93800000
93600000
93400000
93200000
2KB
4KB
8KB
16KB
32KB

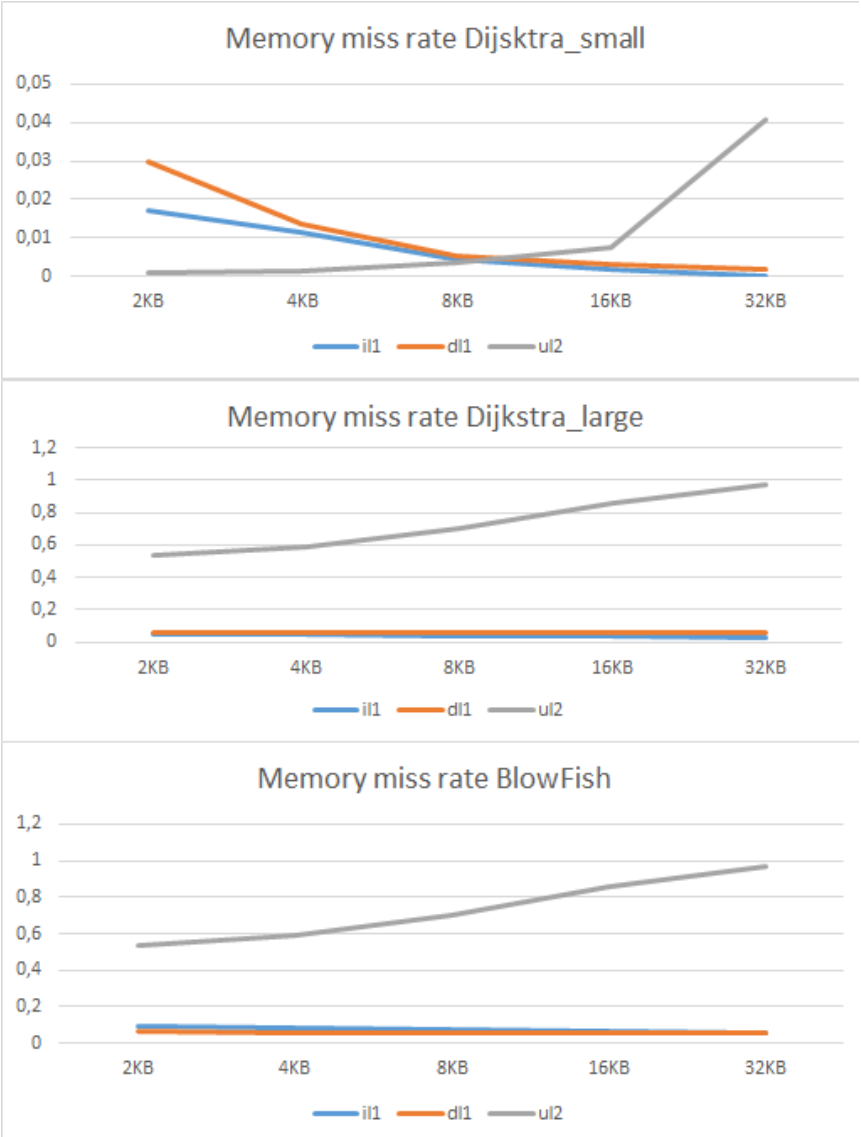

Number of Instruction Dijkstra_large
450200000
450000000
449800000
449600000
449400000
2KB
4KB
8KB
16KB
32KB

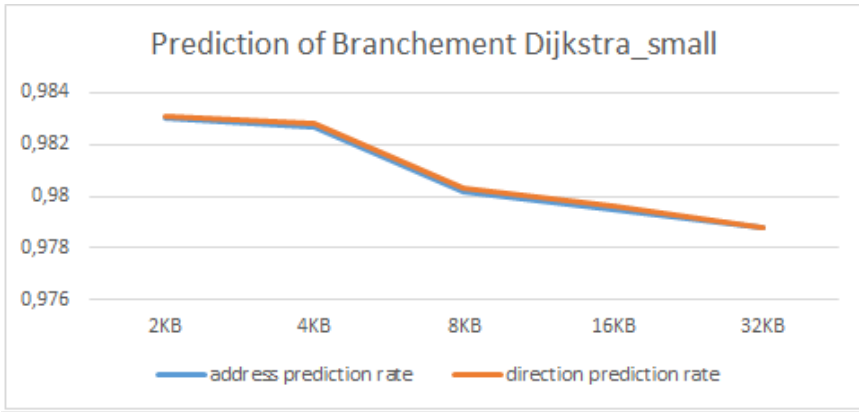

Number of Insruction BlowFish
8900
8800
8700
8600
8500
2KB
4KB
8KB
16KB
32KB

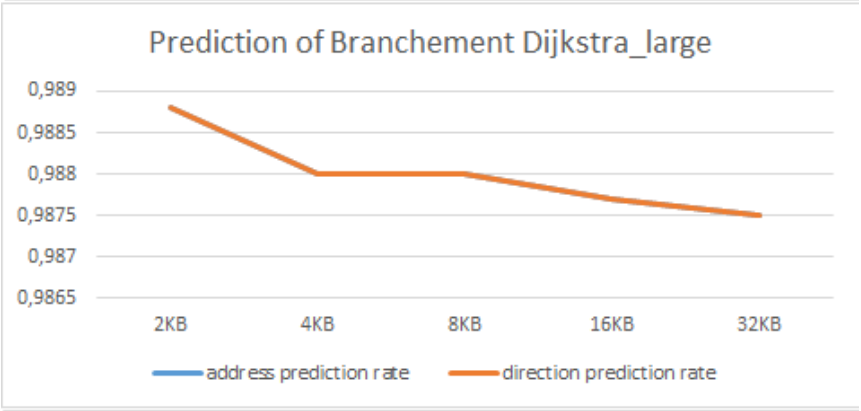

Number of cycles Dijkstra_small
60000000
50000000
40000000
30000000
20000000
10000000
0
2KB
4KB
8KB
16KB
32KB

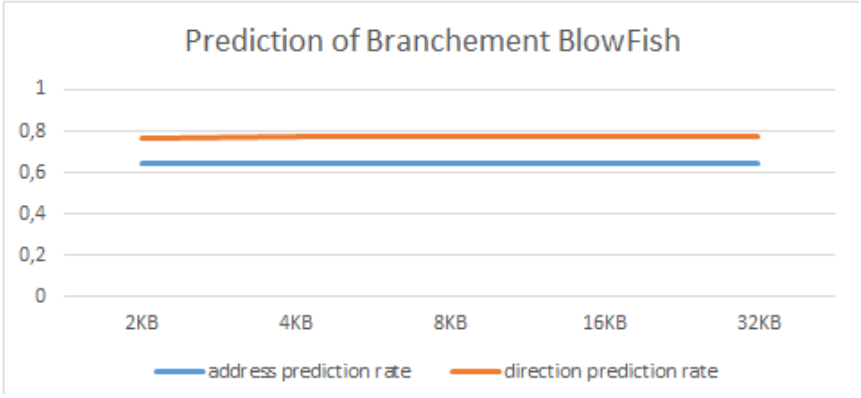

Number of cycles Dijkstra_large
240000000
220000000
200000000
180000000
160000000
2KB
4KB
8KB
16KB
32KB


Number of cycles BlowFish
17500
17000
16500
16000
2KB
4KB
8KB
16KB
32KB


Memory miss rate Dijsktra_small
0,05
0,04
0,03
0,02
0,01
0
2KB
4KB
8KB
16KB
32KB
il1
dl1
ul2
Memory miss rate Dijkstra_large
1,2
1
0,8
0,6
0,4
0,2
0
2KB
4KB
8KB
16KB
32KB
il1
dl1
ul2
Memory miss rate BlowFish
1,2
1
0,8
0,6
0,4
0,2
0
2KB
4KB
8KB
16KB
32KB
il1
dl1
ul2


Prediction of Branchement Dijkstra_small
0,984
0,982
0,98
0,978
0,976
2KB
4KB
8KB
16KB
32KB
address prediction rate
direction prediction rate


Prediction of Branchement Dijkstra_large
0,989
0,9885
0,988
0,9875
0,987
0,9865
2KB
4KB
8KB
16KB
32KB
address prediction rate
direction prediction rate


Prediction of Branchement BlowFish
1
0,8
0,6
0,4
0,2
0
2KB
4KB
8KB
16KB
32KB
address prediction rate
direction prediction rate

Conclusions Pour le **Cortex15** nous pouvons conclure que les performances sont améliorées quand la taille de la mémoire augmente. Nous pouvons également remarquer que lorsque la taille des caches L1 augmente, le **miss rate** des caches L1 diminue, mais que le **miss rate** du cache L2 augmente. En effet les données vont plutôt être stockées dans L1, et non dans le cache L2 : ainsi bien que les accès mémoire vers L2 diminuent, le miss rate de cache L2 va augmenter. Pour ce que concerne le *branchement*, on peut observer dans le programme *Dijsktra*présente une petite réduction du prédiction rate. Elle n'est pas trop évidente et donc nous la négligeons. Entre un cache de 16KB et un cache de 32KB, nous n'observons pas de remarquables améliorations des performances, nous pensons que la meilleure taille est de 16KB pour une surface plus petite.

4 Efficacité Surfacique

4.1 Q6

Par défaut on trouve pour le cache les paramètres par défaut suivants pour une technologie de 32nm :

- size (bytes) 134217728
- block size (bytes) 64
- associativité 1
- technology (nm) 32

4.2 Q7

Les caches IL1 et DL1 ont la même surface pour les deux processeurs parce qu'ils ont la même taille de cache et de bloc et la même associativité. Pour le cortex A7 on a :

- Taille de cache :32768
- Taille de bloc :32
- Associativité :2
- Surface de DL1 : 0.0384318 mm^2
- la surface de caches IL1+DL1=0.0768636mm^2soit environ 17.08%.
- La taille du coeur hors caches L1 : 0.45-0.0602476=0.3731364mm^2 soit environ 82.92%.

Pour le cortex A15 on a :

- Taille de cache :32768
- Taille de bloc :64
- Associativité :2
- Surface de DL1 : 0.03456495mm^2
- la surface de caches IL1+DL1=0.0691299 mm^2 soit environ 3.456%.
- La taille du coeur hors caches L1 : 2-0.0602476=1.9308701mm^2 soit environ 96.544%.

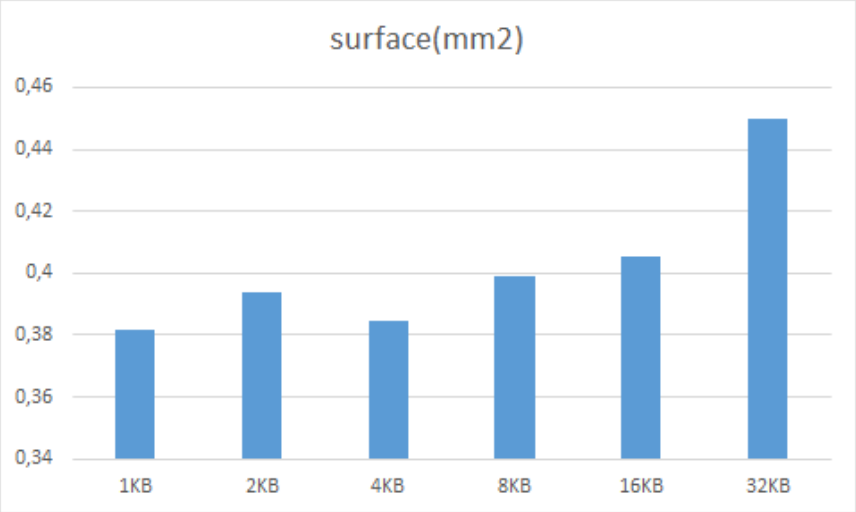
Conclusion : D'après les résultats obtenus à partir de CACTI, la mémoire cache L1 du cortex A7 occupe plus de surface relativement à la surface totale que la mémoire cache L1 du cortex A15. De plus les deux mémoires caches ont la même taille et la même associativité mais A7 possède une taille de bloc plus petite.

4.3 Q8

Cortex A7

D'après la question 7, la surface hors cache L1 est de 0.3731364mm^2

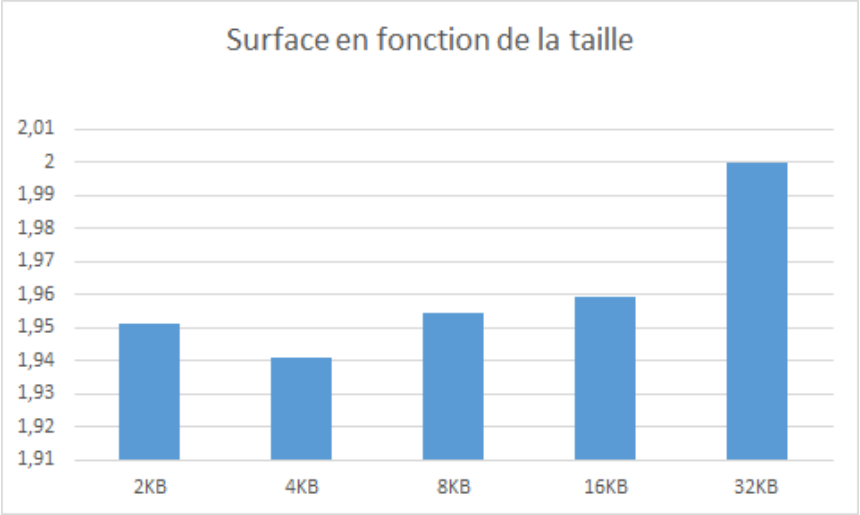
Taille de cache	Surface des caches L1(mm^2)	Surface totale du cortex(mm^2)
1KB	0.008513184	0.381649584
2KB	0.02085274	0.39398914
4KB	0.0113599	0.3844963
8KB	0.0256834	0.3988198
16KB	0.0324069	0.4055433
32KB	0.0768636	0.45



Cortex A15

D'après la question 6 la surface hors cache L1 est 1.9308701mm^2

Taille de cache	Surface des caches L1(mm^2)	Surface totale du cortex(mm^2)
2KB	0.020228124	1.951098224
4KB	0.01023186	1.94110196
8KB	0.02356934	1.95443944
16KB	0.028196	1.9590661
32KB	0.0691299	2

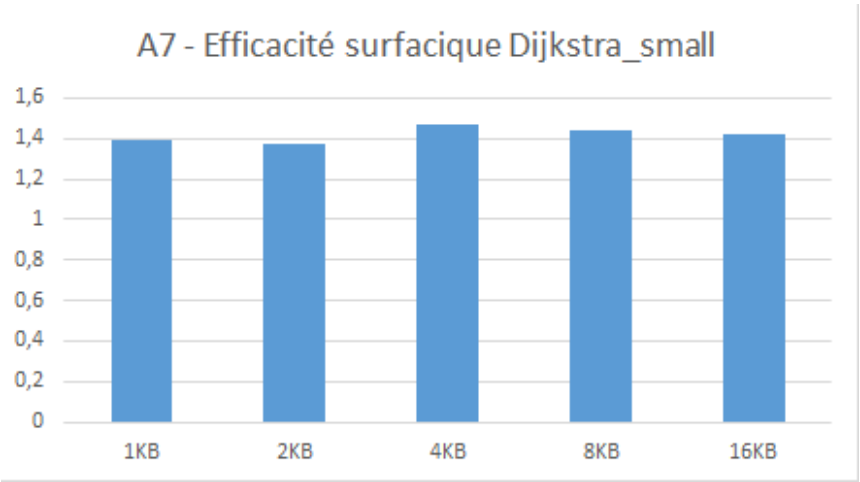


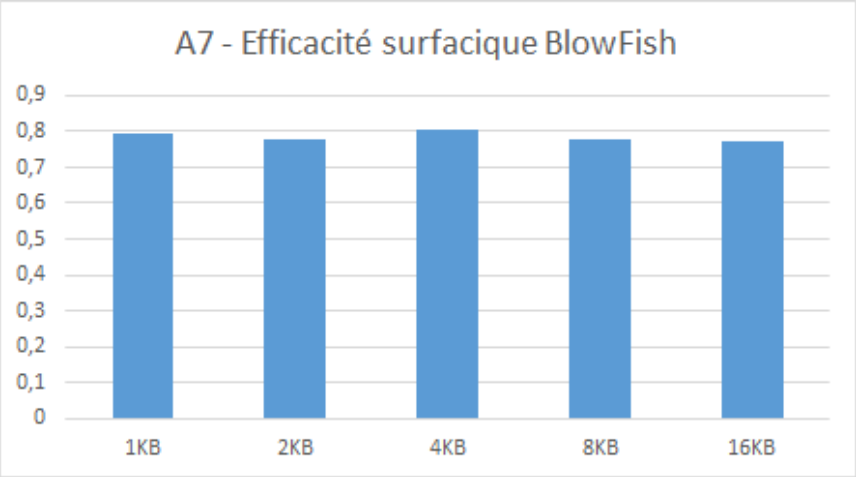
4.4 Q9

On calcule d'efficacité surfacique telle que : Efficacité surfacique=IPC/surface

Cortex A7

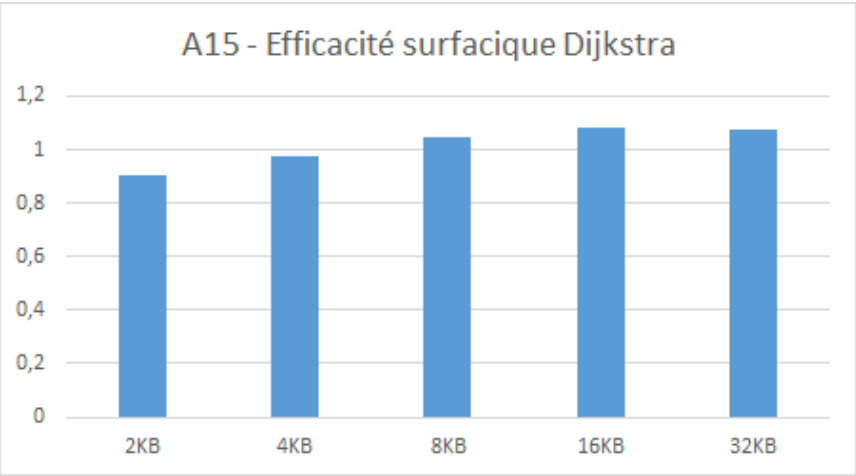
Taille de cache	Efficacité surfacique(Dijkstra)	Efficacité surfacique(BlowFish)
1KB	1.394734915	0.7970662428
2KB	1.376687692	0.7764173398
4KB	1.465813845	0.8026085036
8KB	1.443258334	0.7795500625
16KB	1.425988298	0.7727904764



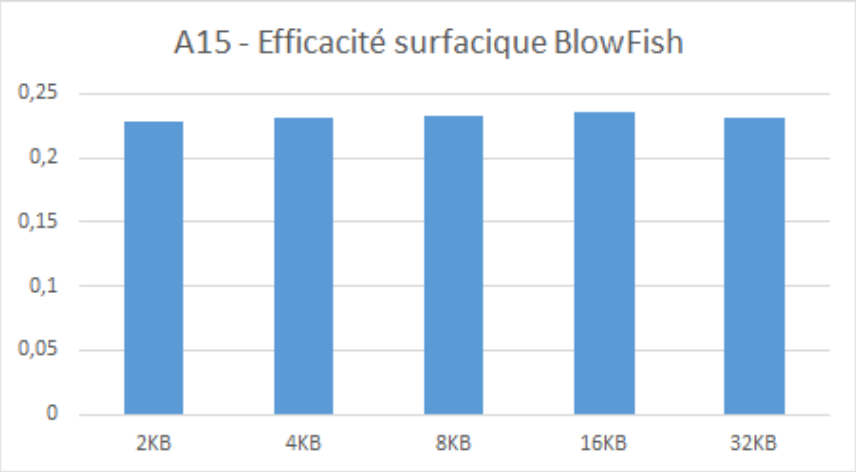


Cortex A15

Taille de cache	Efficacité surfacique(Dijkstra)	Efficacité surfacique(BlowFish)
2KB	0.9033528563	0.2279632957
4KB	0.9781557276	0.2310028063
8KB	1.047359134	0.2334684773
16KB	1.080463799	0.2361329207
32KB	1.07725	0.2318



Conclusion Nous trouvons donc que le maximum d'efficacité surfacique pour Dijkstra et pour BlowFIsh est obtenu pour un cache de 4KB pour A7 et 16KB pour A15.



5 Efficacité Énergétique

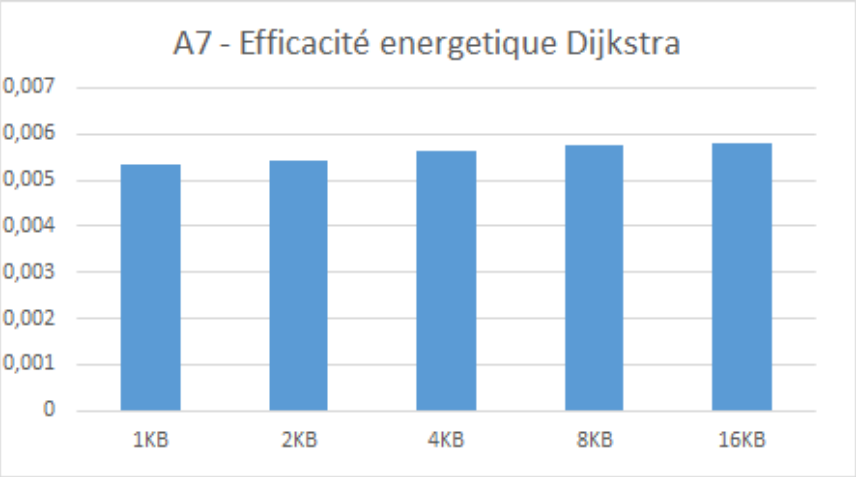
5.1 Q10

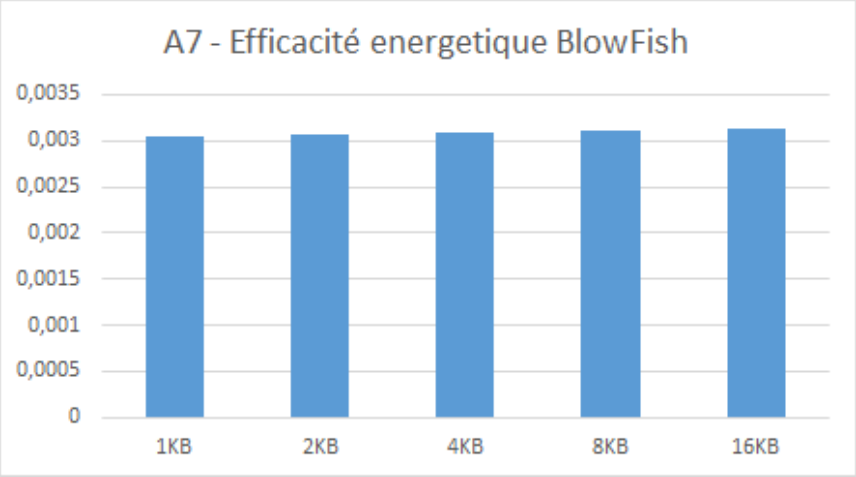
La puissance consommée à fréquence maximale(mW) est le produit de la consommation (mW/MHz) et de la fréquence maximale(MHz).On obtient donc les résultats suivants :
pour Cortex A7 la puissance consommée est de 100 mW(0.10x1000).
pour Cortex A15 la puissance consommée est de 500 mW(0.20x2500).

5.2 Q11

L'efficacité énergique est le rapport de nombre d'instructions par cycles (IPC) et de la consommation énergétiques précédemment calculée.
Cortex A7

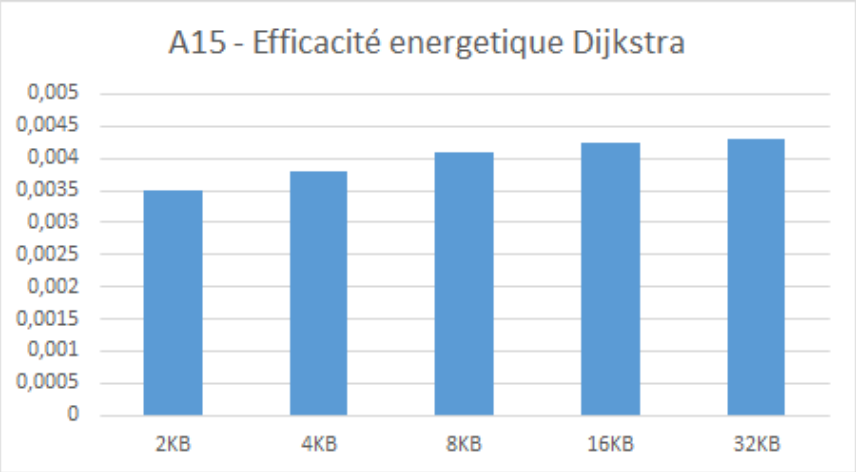
Taille de cache	Efficacité énergique(Dijkstra)	Efficacité énergique(BlowFish)
1KB	0.005323	0.003042
2KB	0.005424	0.003059
4KB	0.005636	0.003086
8KB	0.005756	0.003109
16KB	0.005783	0.003134

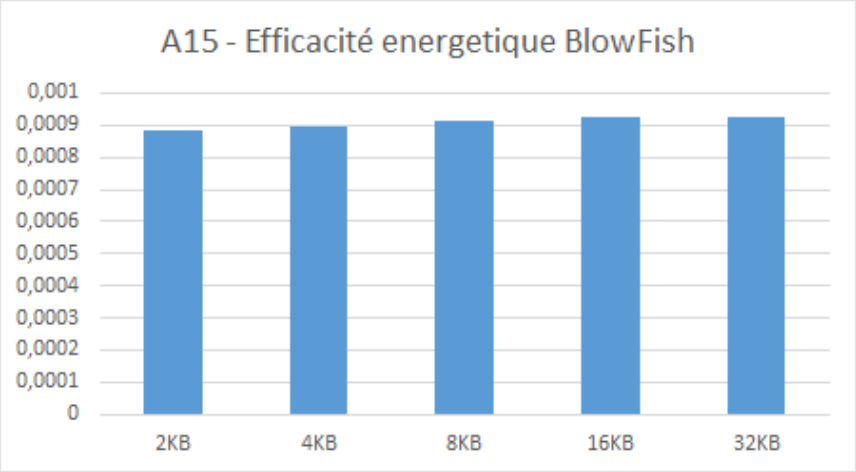




Cortex A15

Taille de cache	Efficacité énergique(Dijkstra)	Efficacité énergique(BlowFish)
2KB	0.003507	0.000885
4KB	0.0037974	0.0008968
8KB	0.004094	0.0009126
16KB	0.0042334	0.0009252
32KB	0.004309	0.0009272





6 Architecture système big.LITTLE

6.1 Q12

La recherche de l'efficacité énergétique est l'objectif commun à tous les fabricants de processeurs. Nous pouvons donc essayer de raisonner de cette manière pour trouver la meilleur configuration de cache possible. On rappelle que dans l'architecture sélectionnée, nous avons deux coeurs qui travaillent à tour de rôle. Cortex A7 est optimisé énergiquement, et A15 est optimisé pour ces performances.
Nous devons cherché la meilleure configuration possible pour les applications Dijkstra et BlowFish, cependant si le seul et unique critère retenu est l'efficacité énergétique, il semble évident qu'il faut prendre la taille de cache maximal car ce dernier critère augmente quand on augmente le cache (cf Q11). Cependant, si l'on considère d'autres facteurs et que l'on compare les deux applications l'une à l'autre, la réponse peut être nuancée. Tout d'abord, il faut noter que la taille du cache L1 double entre 16KB et 32KB. Si notre système

embarqué à besoin d'optimiser la surface, cela doit être pris en compte, sachant que les gains en performance/efficacité sont minimes entre ces deux configurations. Par ailleurs, si l'on compare une à une les deux applications et leurs performances, il s'avère que les gains ne sont pas les mêmes en fonctions de l'augmentation de la taille du cache. Par exemple, on peut noter qu'entre Dijkstra et BlowFish, on augmente l'efficacité respectivement de 7% et 20% et de 3% et 5% (A7 15). En gain relatif, l'optimisation entraînée par l'ajout de cache est 4 et 20 fois supérieure pour A7 et A15 entre Dijkstra et BlowFish. Sachant aussi que la consommation énergétique du coeur A15 est 5 fois supérieure à celle du coeur A7. Au vue de ces analyses, on peut se demander si cette ajout est vraiment nécessaire. En conclusion : Si l'on veut uniquement optimiser énergétiquement, on mettra un cache de taille maximal d'après nos graphiques, mais si l'on veut être raisonnable, on utilisera un cache de 16KB plutôt que 32KB pour minimiser la taille, et peut être que dans un gain de matériel, on préférera un cache de 8KB pour Blowfish (l'augmentation est assez lente après cela, et très faible en comparaison de l'amélioration chez Dijkstra).

7 Facultatif

7.1 Q13

Les configurations sont équivalentes. Si l'on se base sur le choix fait avec les questions 11/12.Par ailleurs, les compromis ont déjà été discutés dans la question 12, et on adaptera donc notre choix (performance, taille, autonomie énergétique) en fonction du besoin. A noter aussi que notre analyse ne se base que sur deux applications et sans réel contexte, cela limite forcément les compromis que l'on peut imaginer ou anticiper.

7.2 Q14

Tout d'abord, il faut spécifier le contexte d'utilisation : milieu contraignant en terme d'autonomie (équipement embarqué devant minimiser sa consommation énergétique) ou simplement cherchant à minimiser son impacte énergétique ; miniaturisation nécessaire (contrainte de taille) ; ou peut être un contexte où uniquement la performance et la rapidité sont recherché (super-calculateur...). De manière très approximative, on obtient ainsi une première ligne de conduite pour notre architecture. Une fois que l'on a défini notre besoin, on peut chercher à optimiser cela. Pour ce faire, on analysera l'ensemble des applications les plus couramment utilisées sur la plateforme (profiling). Enfin, on notera que de manière général, se focaliser sur la taille des caches est une très bonne option, puisqu'augmenter le cache (jusqu'à une taille raisonnable à déterminer) permet de diminuer les miss rates des caches L1 en diminuant les temps d'accès aux ressources mémoire, sans ajouter un grand coût énergétique dû au fait que les caches sont des unités de sauvegarde et non pas de calcul.