ES201 - Rapport de projet TP4

par

Bastien Hubert - Liam Kelley - Sylvain Largent - Thomas Raynaud

Professeur encadrant :
Nicolas Gac

Février 2022

Sommaire

1 Introduction

Dans ce TP, nous nous attachons à étudier deux coeurs ARM : le Cortex A7 et le Cortex A15. En effet, nous modifions les différents paramètres de l'architecture et évaluons l'impact sur les performances et l'efficacité surfacique des coeurs. On peut alors aussi déterminer l'efficacité énergétique. Ces études sont réalisées à partir des applications : Djikstra (Traitement de graphes) et Blowfish (Chiffrement). Grâce à l'ensemble de ces études, nous chercherons alors à déterminer la meilleure configuration du processeur big.LITTLE pour ces deux applications.

2 Partie 1

2.1 Question 1

2.1.1 Résultats pour BlowFish

On fait un make all pour compiler les exécutables.
On exécute alors le profiling avec **sim-profile -iclass true bf.ss input_small.asc**.

BlowFish		
Classe d'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 computation	0	0.0
trap	8	0.10

2.1.2 Résultats pour Dijkstra

On fait un make all pour compiler les exécutables.
On exécute alors le profiling avec **sim-profile -iclass true dijkstra_small.ss input.dat**.

Dijkstra		
Classe d'instructions	Nombre d'instructions	Pourcentage
load	14786552	26.94
store	4345374	7.92
uncond branch	437907	0.80
cond branch	9381122	17.09
int computation	25930417	47.25
fp computation	0	0.00
trap	239	0.0

2.2 Question 2

Dans le cas de **BlowFish**, les instructions les plus utilisés sont celles d'écriture (store) qui occupe 45.61% du code (Écriture de clef et les nombreux cryptages par BF_ENC). Et 32.62% des instructions sont utilisées pour réaliser des opérations sur les entiers (int computation). Il faudrait donc améliorer ces deux classes d'instructions au niveau des fichiers sources.

Dans le cas de **Dijkstra**, les instructions les plus utilisés sont pour les opérations d'entier (int computation) qui occupe 47.25% du code (Il y a 100 noeuds dans le graphe (4 boucles for, conditionnés par ce nombre)). Étant donné qu'il est nécessaire de parcourir le graphe à plusieurs reprises, il est naturel que 26.94% des instructions soient de lecture (load). Ces deux classes d'instructions sont donc à améliorer.

2.3 Question 3

Rappelons le profiling des différentes fonctions mentionnées dans la question.

Poly_Mul		
Classe d'instructions	Nombre d'instructions	Pourcentage
load	453057	23.07
store	168821	8.60
uncond branch	89481	4.56
cond branch	206344	10.51
int computation	1005753	51.21
fp computation	40397	2.06
trap	10	0.00

SSCA2-BCS		
Classe d'instructions	Nombre d'instructions	Pourcentage
load	580	7.94
store	3494	47.82
uncond branch	164	2.24
cond branch	813	11.13
int computation	2246	30.74
fp computation	0	0.00
trap	9	0.12

SHA-1		
Classe d'instructions	Nombre d'instructions	Pourcentage
load	2469983	16.72
store	1199384	8.12
uncond branch	11120	0.08
cond branch	900857	6.10
int computation	10192727	68.99
fp computation	0	0.00
trap	26	0.00

La fonction Poly_mul réalise un produit de polynômes. La fonction SSCA2-BCS est une fonction de traitement de graphes et la fonction SHA-1 est une fonction liée à la cryptographie.

On retrouve donc naturellement des similarités entre les profilings de SHA-1 et BlowFish. Cependant le premier utilise une méthode de chiffrement par hachage, tandis que le second créée une clef secrète. La classe d'instruction d'écriture (store) diffère car l'algorithme de BlowFish cherche à écrire des clefs secrètes dont la longueur est comprise entre 32 et 448 bits (Wikipédia), ce qui n'est pas le cas de SHA-1.

Les deux algorithmes liés au parcours de graphes (Dijkstra et SSCA2-BCS) ont des résultats similaires. Hormis la clase d'instruction d'écriture (load), les améliorations sont sur les classes d'instructions : de lecture, d'opérations sur les entiers et les branchements conditionnels. Cependant SSCA2-BCS semble réaliser davantage d'analyses, ce qui pourrait expliquer la différence au niveau des instructions d'écriture.

Les résultats entre le produit de polynômes et l'algorithme de Dijkstra, retrouvent leur forte similarité dans le fait qu'ils disposent un nombre similaire de boucles for. La principale différence entre les deux profilings, vient du fait

que l'algorithme de multiplication de polynômes manipule des flottants à la différence de l'algorithme de Dijkstra.

3 Partie 2

3.1 Commandes utilisées

Dans le cas d'A7 :
sim-outorder -bpred bimod -bpred :btb 256 4 -fetch :ifqsize 4 -fetch :mplat 8 -decode :width 2 -issue :width 4 -issue :inorder true -commit :width 2 -ruu :size 2 -lsq :size 8 -res :ialu 1 -res :fpalu 1 -res :imult 1 -res :fpmult 1 -cache :il1 il1 :32 :32 :2 :l -cache :dl1 dl1 :32 :32 :2 :l -cache :dl2 dl2 :512 :32 :8 :l -redir :sim sim/dijkstra_l_A7_32.out dijkstra_large.ss input.dat

Dans le cas de d'A15 :
sim-outorder -bpred 2lev -bpred :btb 256 4 -fetch :ifqsize 8 -fetch :mplat 15 -decode :width 4 -issue :width 8 -issue :inorder false -commit :width 4 -ruu :size 16 -lsq :size 16 -res :ialu 5 -res :imult 1 -res :fpalu 1 -res :fpmult 1 -cache :dl1 dl1 :32 :64 :2 :l -cache :il1 il1 :32 :64 :2 :l -cache :dl2 dl2 :512 :64 :16 :l -redir :sim sim/dijkstra_l_A15_32.out dijkstra_large.ss input.dat

3.1.1 Descriptif des différents paramètres

sim-outorder	
-bpred bimod	*Choix méchanisme de prédiction*
-bpred :btb 256 4	*BTB*
-fetch :ifqsize 4	*Fetch queue size*
-fetch :mplat 8	*Latence mis-pred*
-decode :width 2	*Decode*
-issue :width 4	*Issue*
-commit :width 2	*Commit*
-ruu :size 2	*RUU*
-lsq :size 8	*LSQ*
-res :ialu 1	*Nombre d'ALU entiers*
-res :fpalu 1	*Nombre d'ALU Flottants*
-res :imult 1	*Nombre de Multp. Entiers*
-res :fpmult 1	*Nombre de Multp. Flottants*
-cache :il1 il1 :32 :32 :2 :l	*Settings cache IL1*
-cache :dl1 dl1 :32 :32 :2 :l	*Settings cache DL1*
-cache :dl2 dl2 :512 :32 :8 :l	*Settings cache L2*

-redir :sim *Écrire résultat simul dans un fichier*
fichier de sortie script.ss input doc

3.2 Question 4

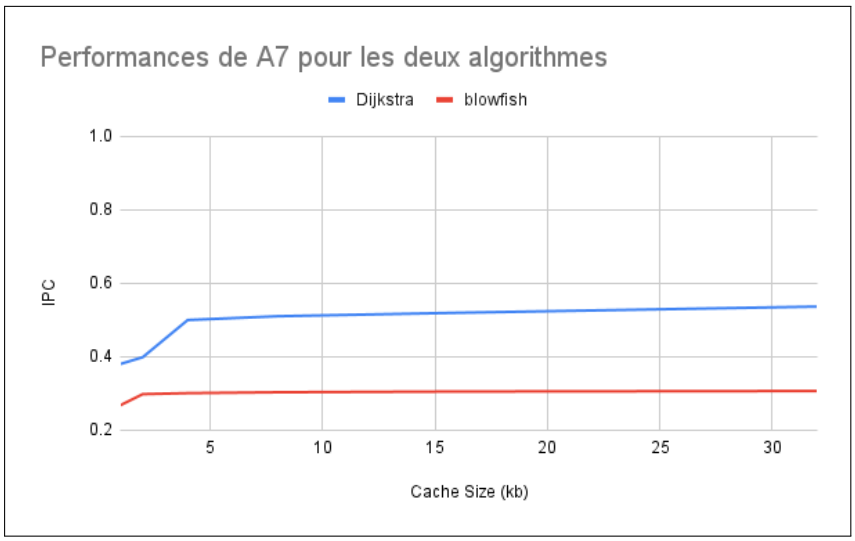


FIGURE 1 – Variation de l'IPC en fonction de la taille des caches pour le Cortex A7.

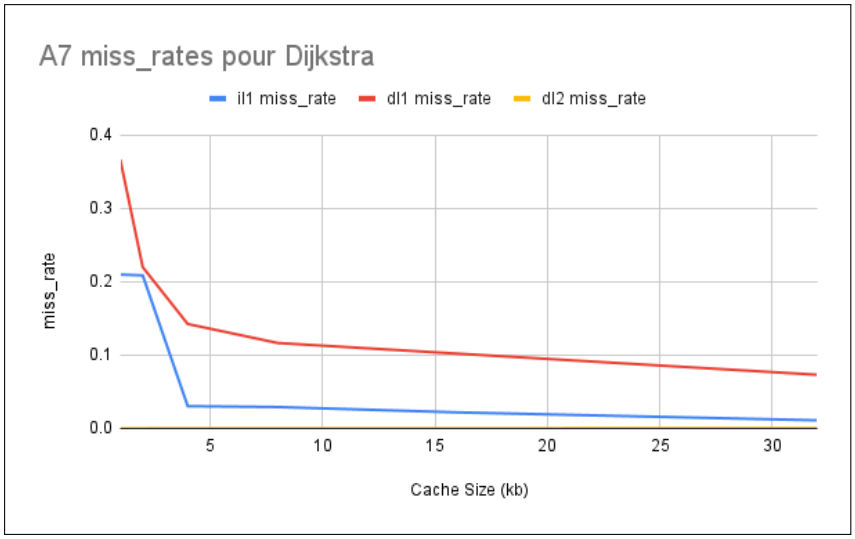


FIGURE 2 – Variation des miss rates en fonction de la taille des caches pour le Cortex A7.

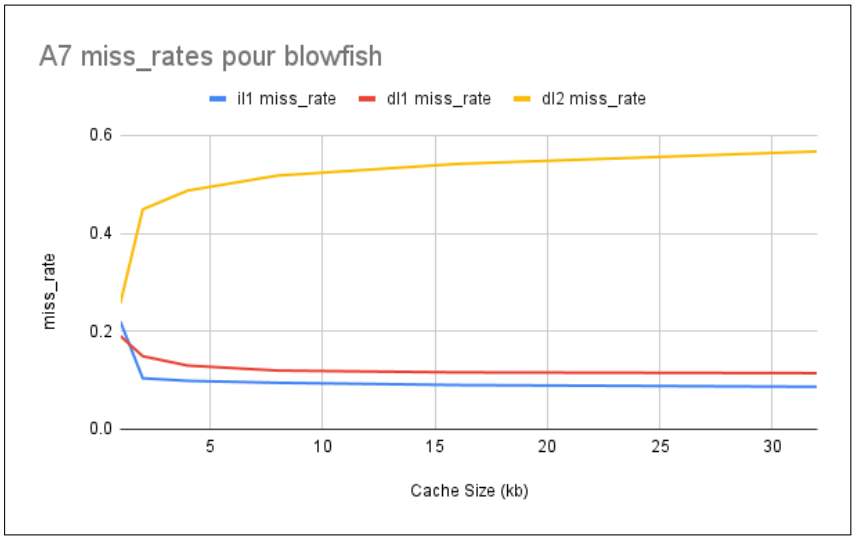


FIGURE 3 – Variation des miss rates en fonction de la taille des caches pour le Cortex A7.

Analyse des résultats

La performance du cortex A7 plafonne très rapidement pour les différentes tailles de cache. Le maximum est presque atteint pour un taille de cache d'environ 2kb pour nos simulations avec Blowfish et 4kb pour nos simulations avec Dijkstra. Les performances restent très marginalement croissantes passées ce stade.

Comme prévu, la miss_rate des caches 1 décroissent selon l'augmentation de la taille du cache. Pour Dijkstra, même si les valeurs sont très petites, on peut remarquer que dl2 miss_rate croit quadratiquement selon l'augmentation de la taille de cache 1. Cela s'explique par le fait que la taille de cache 2 reste constante pendant que la taille de cache 1 augmente. Pour Blowfish, la dl2 miss_rate continue de grandir pendant que celles du cache 1 stagnent à partir de 4kb. La conclusion pour le Cortex A7 est que la taille de L1 la plus performante est de 4kb pour les deux algorithmes.

3.3 Question 5

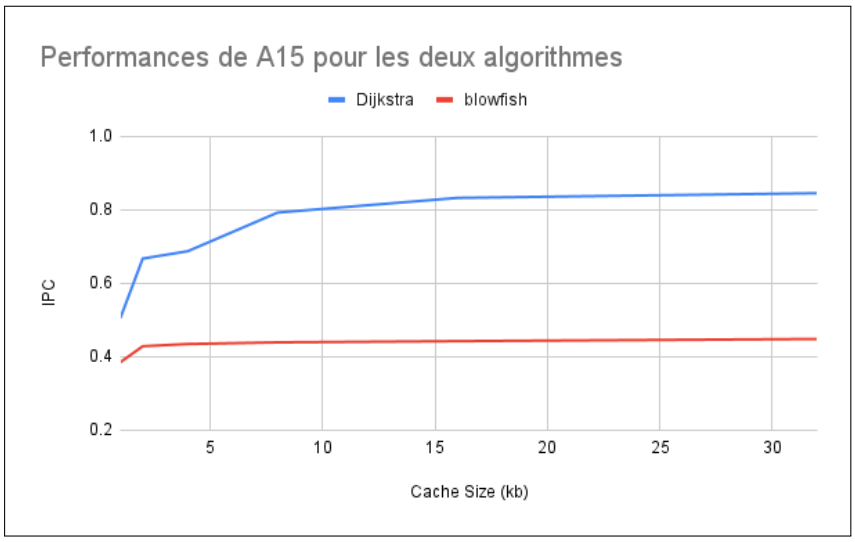


FIGURE 4 – Variation de l'IPC en fonction de la taille des caches pour le Cortex A7.

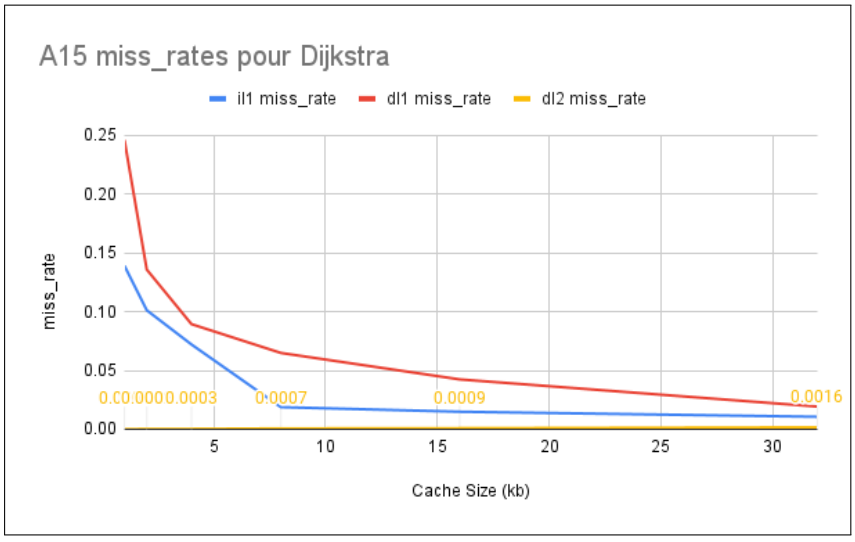


FIGURE 5 – Variation des miss rates en fonction de la taille des caches pour le Cortex A15.

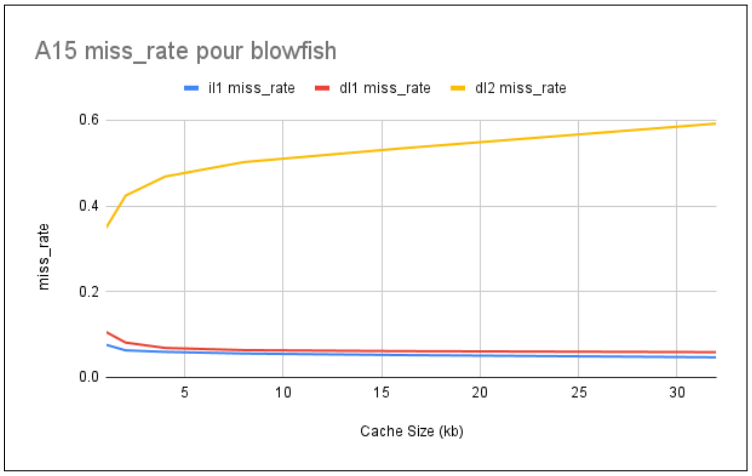


FIGURE 6 – Variation des miss rates en fonction de la taille des caches pour le Cortex A15.

Analyse des résultats

Les courbes sont similaires a celles du cortex A7. L'IPC stagne plus tard pour l'algorithme de Dijkstra, autour de 16kb. La variation des différentsparamètres a peu d'influence sur les le comportement de l'application BlowFish. Les miss_rate du cache 1 convergent toujours très rapidement à partir de 4kb pendant que le dl2 miss_rate continue de grimper.
On peut conclure que la taille de cache la plus efficace est certainement 8kb pour les deux algorithmes.

4 Partie 3

4.1 Question 6

D'après le fichier cache.cfg, les paramètres par défauts sont :

- taille de cache : 134217728 octets, soit 128 Mo
- taille de bloc : 64 octets
- associativité : 1
- technologie : 32 nm

4.2 Question 7

La version de CACTI utilisée ne supportant pas le 28nm, nous ferons l'hypothèse d'une technologie de 32nm.

Pour le Cortex A15, les caches I-L1 et D-L1 ont chacun les configurations suivantes :

— tableau de données : 0.0301238 mm^2
— tableau de tags : 0.00444115 mm^2

Soit **une surface totale de cache L1 de 0.069 mm^2 pour le Cortex A15.**

De même, pour Cortex A7, les caches I-L1 et D-L1 ont chacun les configurations suivantes :

— data array : 0.0301238 mm^2
— tag array : 0.008308 mm^2

Soit **une surface totale de cache L1 de 0.077 mm^2 pour le Cortex A7**.

Pour une technologie de 28nm, les Cortex A15 et A7 ont respectivement une surface de 2 mm^2 et de 0.45 mm^2, ce qui correspond (en appliquant un coefficient de proportionnalité à la longueur et la largeur) à des surfaces de 2.612 mm^2 et de 0.588 mm^2.

On en déduit que **le cache L1 représente 2.646% de la surface du cœur A15 et 13.077 de celle du cœur A7**.

Finalement, **les cœurs A15 et A7 (sans leur cache L1) occupent une surface de 2.543 mm^2 et de 0.511 mm^2 respectivement**.

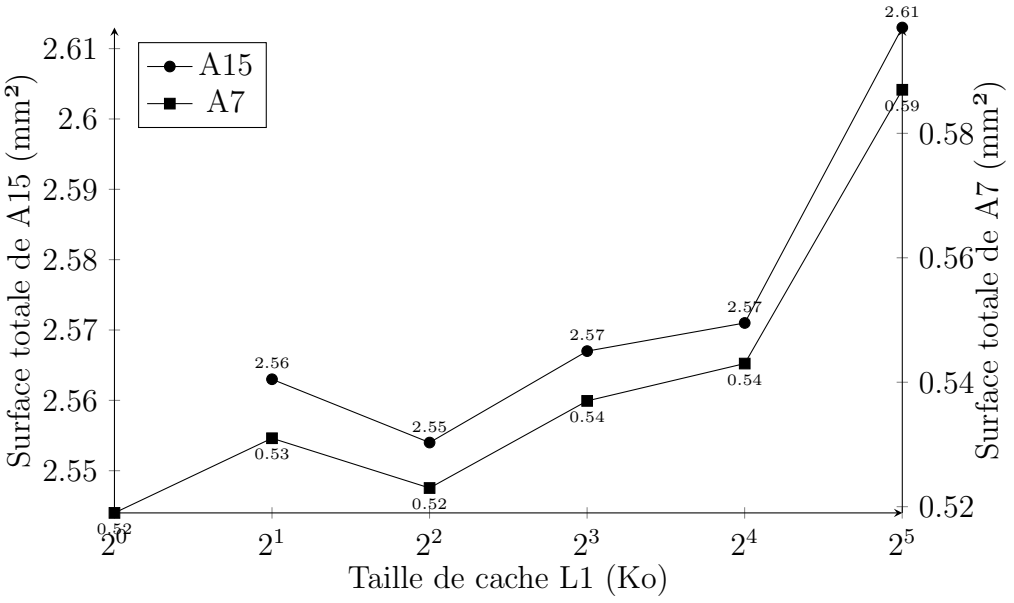
On constate que le Cortex A15 prend beaucoup plus de place que le A7, ce qui est logique car le A15 est plus performant que le A7. De plus, l'asociativité dans le Cortex A15 est supérieure à celle du A7, ce qui est cohérent avec le fait que la tag array y soit plus petite (et donc le cache L1 aussi puisqu'ils ont la même taille de data array).

4.3 Question 8

En faisant varier les tailles de cache L1 pour les Cortex A15 et A7, on obtient le tableau suivant :

Surface de cache L1 (mm^2)		
Taille de cache L1 (Ko)	A15	A7
1	-	0.008
2	0.019	0.020
4	0.010	0.012
8	0.024	0.026
16	0.028	0.032
32	0.070	0.076

Si à cela on ajoute les surfaces des cœurs avec leur cache L2, déterminées précédemment, on obtient le graphe récapitulatif ci-dessous :

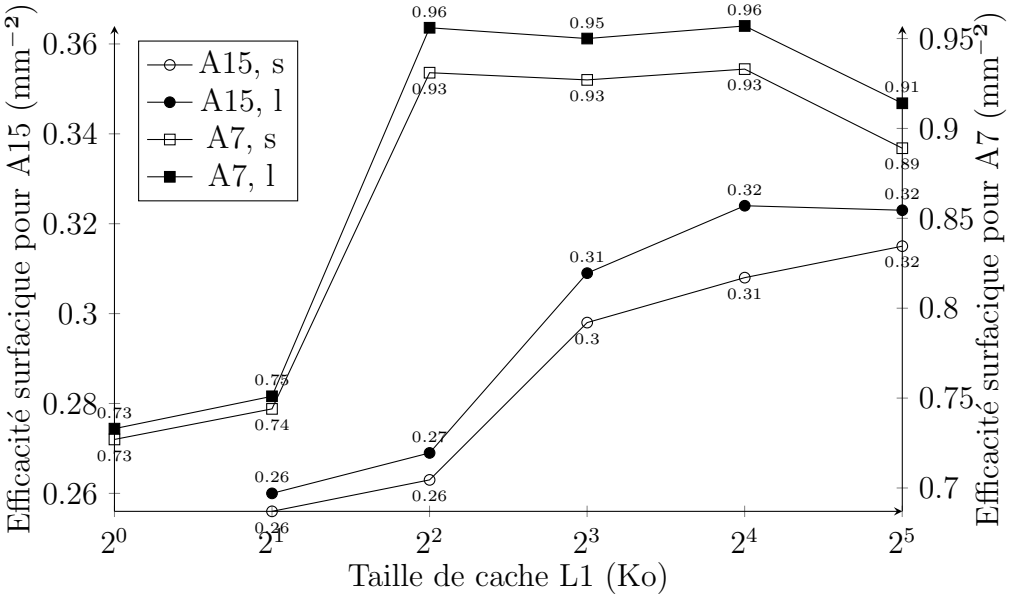


4.4 Question 9

4.4.1 Résultats pour Dijkstra

En récupérant les données fournies par sim-outorder, on trouve les efficacités surfaciques suivantes :

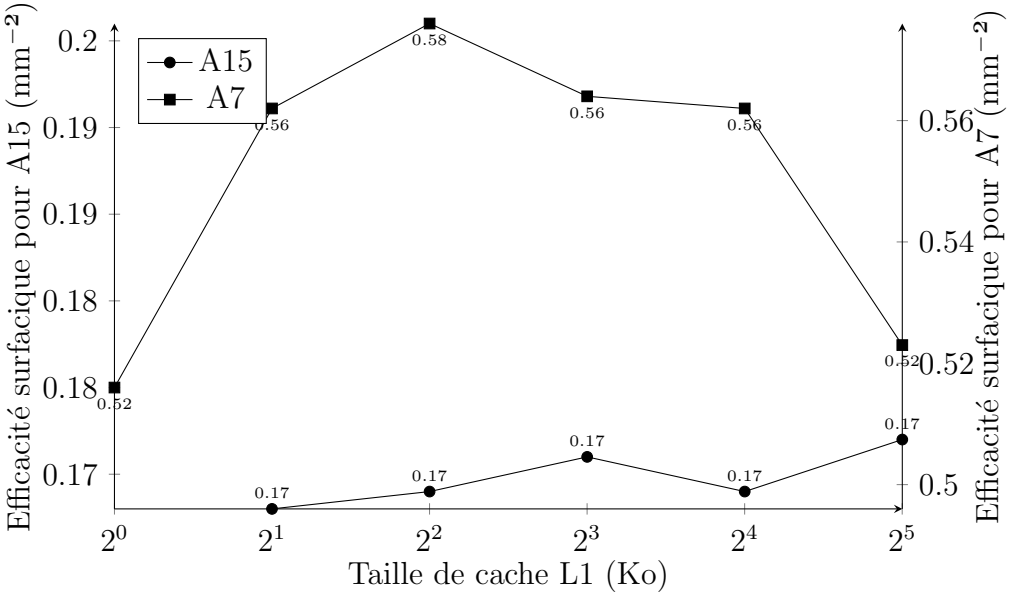
Efficacité surfacique pour Dijkstra (mm^{-2})				
Taille de cache L1 (Ko)	A15, s	A15, l	A7, s	A7, l
1	-	-	0.727	0.733
2	0.256	0.260	0.744	0.751
4	0.263	0.269	0.931	0.956
8	0.298	0.309	0.927	0.950
16	0.308	0.324	0.933	0.957
32	0.316	0.323	0.889	0.914



4.4.2 Résultats pour BlowFish

De même, les données de sim-outorder conduisent aux efficacités surfaciques suivantes :

Efficacité surfacique pour BlowFish (mm^{-2})		
Taille de cache L1 (Ko)	A15	A7
1	-	0.516
2	0.168	0.562
4	0.169	0.576
8	0.171	0.564
16	0.169	0.562
32	0.172	0.523



5 Partie 4

5.1 Question 10

Les consommations énergétiques respectives des cortex A7 et A15 sont de 0.10 mW/MHz et 0.20 mW/MHz. Ainsi, sachant que les fréquences maximales respectives des cortex A7 et A15 sont de 1.0 GHz et 2.5 GHz, on en déduit la puissance de chacun de ces cortex à la fréquence maximale :

$P(A7)_{Fmax} = Conso * Fmax = 0.10 * 1,0.10^3 = 100mW$
$P(A15)_{Fmax} = Conso * Fmax = 0.20 * 2,5.10^3 = 500mW$

5.2 Question 11

Pour suivre l'évolution de l'efficacité énergétique de chaque processeur en fonction de la taille du cache L1, on récupère l'évolution de l'IPC pour chaque processeur en fonction de la taille de cache L1 que l'on divise par la puissance des processeurs à la fréquence maximale. Cela permet d'aboutir aux deux tableaux suivants :

Algorithme : Dijkstra (input small)		
Taille de cache L1 (kB)	$Eff_{énergétique}(A7)$ (mW^{-1})	$Eff_{énergétique}(A15)$ (mW^{-1})
1	0.003774	-
2	0.003951	0.001305
4	0.004871	0.001348
8	0.004977	0.001534
16	0.005067	0.001607
32	-	0.001649

Algorithme : BlowFish		
Taille de cache L1 (kB)	$Eff_{énergétique}(A7)$ (mW^{-1})	$Eff_{énergétique}(A15)$ (mW^{-1})
1	0.002679	-
2	0.002983	0.000857
4	0.003010	0.000869
8	0.003034	0.000878
16	0.003054	0.000885
32	-	0.000897

6 Partie 5

6.1 Question 12

En ne prenant en compte que l'efficacité énergétique des processeurs, les résultats obtenus pour la question précédente nous permettent de conclure que cette efficacité croit avec la taille du cache L1, et ce, pour les deux processeurs et pour les deux algorithmes. De ce fait, sur le plan énergétique, il semble souhaitable de choisir une taille de cache L1 de **16KB pour A7 pour les deux algorithmes et une taille de cache L1 de 32KB pour A15 pour les deux algorithmes**. Notons tout de même une différence entre Dijkstra et BlowFish : une croissance de l'efficacité est observée dans les deux cas, mais la croissance n'est pas rigoureusement identique, les performances gagnées en augmentant la taille du cache ne sont plus si évidentes pour BlowFish. De plus, on semble tendre vers des limites asymptotiques assez rapidement dans les deux cas. Notre choix actuel se tourne vers l'utilisation de la plus grande taille de cache, cependant, et au vu de ces observations, il serait peut être judicieux d'apporter de nouveaux éléments dans l'analyse afin d'effectuer un meilleur choix.

7 Partie 6

7.1 Question 13

Les courbes obtenues dans les parties 2 et 3 pour les Cortex A15 et A7 présentent les mêmes tendances globales, même si les ordres de grandeurs sont différents. Aussi, il semble judicieux de lier les résultats et analyses effectués sur le plan énergétique avec ceux sur le plan surfacique. En effet, en prenant en compte cette dimension, de nouveaux choix peuvent être effectués. Pour A7, l'efficacité surfacique est la meilleure pour une taille de cache de 16KB pour Dijkstra et 4KB pour BlowFish. Étant donné la faible évolution de l'efficacité énergétique à partir de ces tailles de cache, ainsi que la faible différence d'efficacité surfacique entre les tailles de cache 16KB et 8KB pour Dijkstra, un compromis entre les deux dimensions nous dirigent vers le choix du taille de cache de 8KB pour Dijkstra et 4KB pour BlowFish pour le Cortex A7. Pour le Cortex A15, nous suivons la même logique et choisissons une taille de cache de 8KB pour l'algorithme BlowFish et de 16KB pour l'algorithme de Dijkstra car l'écart de performance entre les taille 16KB et 32KB sont trop faibles pour pencher vers le choix d'une plus grande

taille de cache qui implique des coûts supplémentaires.

Pour résumer, après analyse sur les plans surfaciques et énergétiques, nous optons pour les choix suivants :

- **Pour A7 : 8KB pour Dijkstra et 4KB pour BlowFish**
- **Pour A15 : 16KB pour Dijkstra et 8KB pour BlowFish**

On a vu que les efficacités surfaciques et énergétiques du Cortex A7 sont supérieures à celles du Cortex A15, bien que ses performances soient moins bonnes que celles de ce dernier. Les deux cœurs ne sont pas équivalents et on doit donc trouver un compromis entre utiliser le Cortex A15, très performant, et le Cortex A7, bien plus efficace. Ce compromis se fait en fonction des besoins en performances et en efficacité de l'utilisateur, et le cœur utilisé peut être déterminé au moment de l'exécution de son programme.