Clara Ducher
Alexandre Fournel

Rafael Gomes
Miguel Colmenares

Rapport TP3 exercice n°4 - ES201

1. Profiling de l'application

Q1 : On utilise la commande sim-profile -iclass dijkstra_small input.dat pour obtenir les pourcentages de chaque classe d'instruction. On obtient les résultats suivants :

Classe d'instruction	dijkstra_small	dijkstra_large	bf.ss
load	26.94	27.21	8.12
store	7.92	7.65	45.61
uncond branch	0.80	0.41	2.18
cond branch	17.09	16.81	11.37
int cmputation	47.25	47.91	32.62
fp computation	0.00	0.00	0.00
trap	0.00	0.00	0.10

Q2 : D'après les trois tests précédents, la classe d'instruction qui sollicite le plus de temps processeur est celle manipulant des entiers (int computation) à hauteur d'environ 50%. Pour dijsktra, on observe que les opérations sur la mémoire (chargement ou stockage) occupent aussi environ 30% du temps. Pour améliorer les performances du programme, il faudrait améliorer des performances de manipulation des entiers et d'accès mémoire.

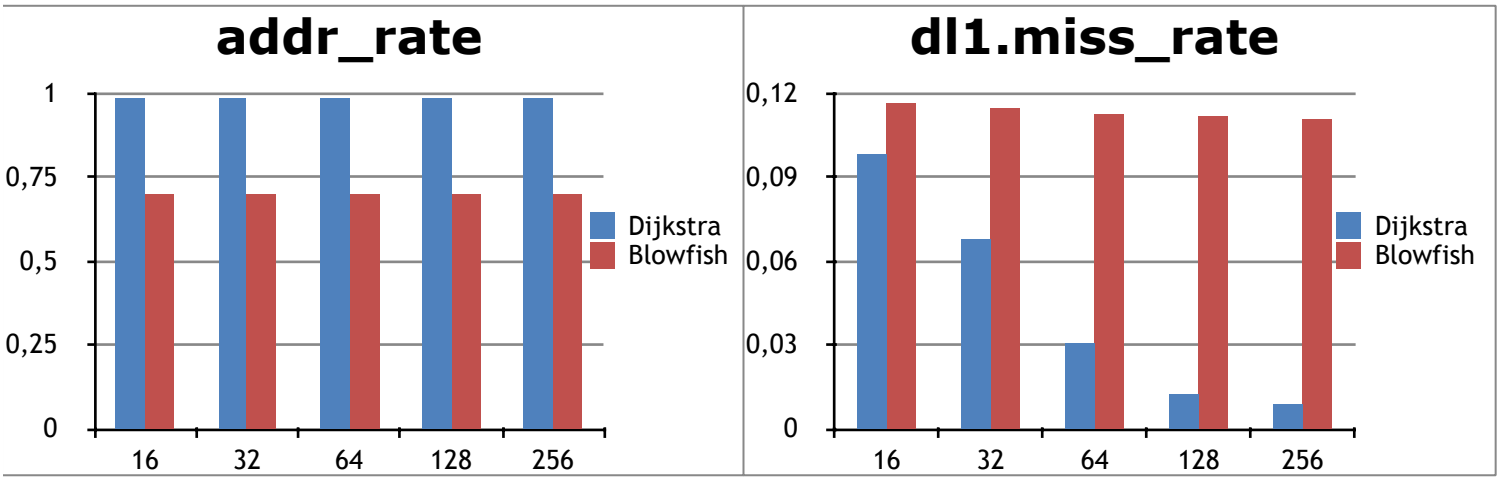
Q3 : Les résultats du TP2 sur les pourcentages de chaque instruction corroborent les résultats précédents sur les points à améliorer. Les instructions qui sollicitent le plus de temps processeur sont la manipulation d'entiers et les accès mémoire dans des pourcentages proches de 50%. En revanche, les applications précédentes ne faisaient pas de calculs avec des flottants (fp computation) tandis que les applications étudiées au TP2 avaient des pourcentages entre 3.44% et 10.56% ce qui représente une proportion non négligeable devant celle des autres instructions. Les applications divergent entre elles également au niveau des instructions de branchement inconditionnel, très peu sollicitées ici, contrairement aux applications du TP2 qui utilisent entre 5.95% et 7.96% d'instructions uncond branch.

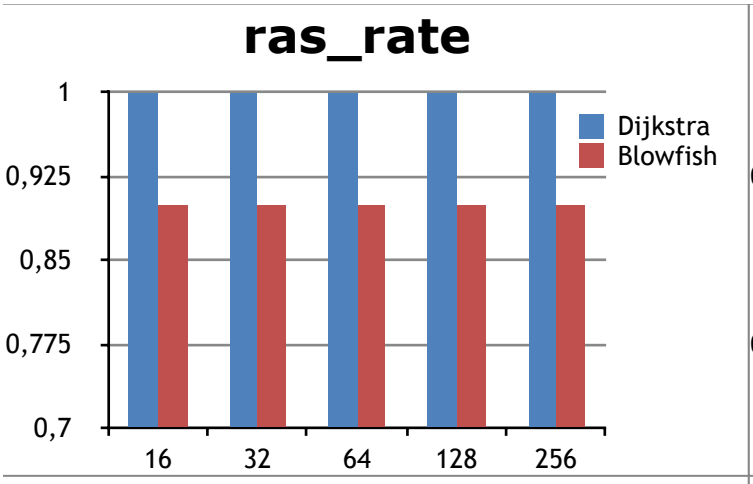
2. Evaluation des performances

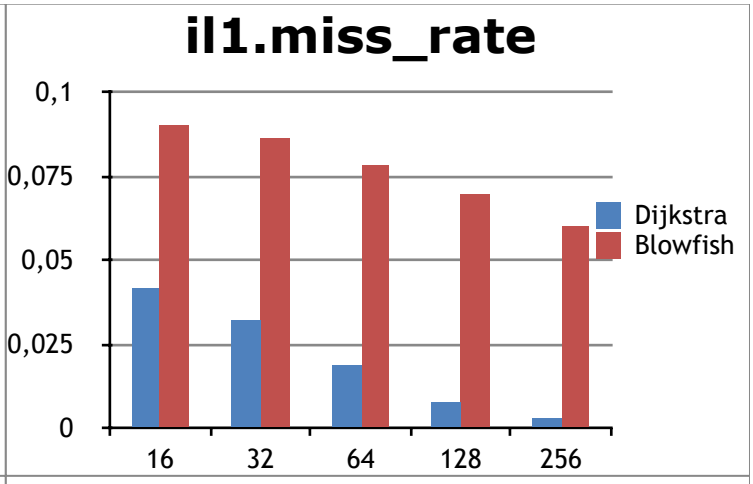
Q4 : On fixe la taille de cache L2 à 512KB et on fait varier la taille des caches L1. On utilise les commandes suivantes :

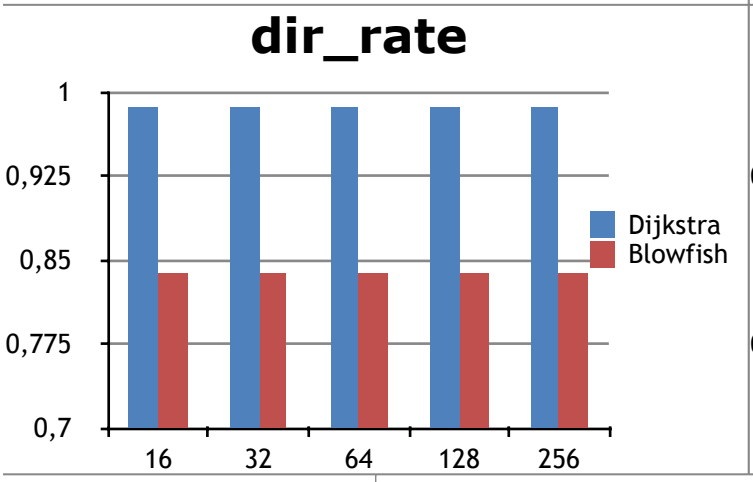
```
#!/bin/bash
export PATH=$PATH:/usr/ensta/pack/simplescalar-3v0d/bin/:/usr/ensta/pack/simplescalar-3v0d/simplesim-3.0
alias sslittle-na-sstrix-gcc='/auto/appy/ensta/pack/simplescalar-3v0d/bin/sslittle-na-sstrix-gcc'
alias sim-outorder='/auto/appy/ensta/pack/simplescalar-3v0d/simplesim-3.0/sim-outorder'
alias sim-profile='/auto/appy/ensta/pack/simplescalar-3v0d/simplesim-3.0/sim-profile'
alias sim-cache='/usr/ensta/pack/simplescalar-3v0d/simplesim-3.0/sim-cache'
clear
for taille in 16 32 64 128 256 512; do
    echo $taille
    for i in small; do
        sim-outorder -redir:sim data_dij_"$i"_"$taille".txt -issue:inorder false -bpred bimod -bpred:bimod 256 -fetch:mplat 8
        -fetch:ifqsize 4 -decode:width 2 -issue:width 4 -commit:width 2 -ruu:size 2 -lsq:size 8 -res:ialu 1 -res:fpalu 1 -res:imult 1
        -res:fpmult 1 -cache:il1 il1:"$taille":32:2:l -cache:dl1 dl1:"$taille":32:2:l -cache:il2 dl2 -cache:dl2 ul2:2048:32:8:l
        dijkstra_"$i" input.dat
    done
done
#performance
for donnee in sim_IPC sim_total_insn sim_cycle; do
grep "$donnee" data_dij_small_* >> sim_dij_small_"$donnee".txt
done

#memoire
for donnee in il1.miss_rate dl1.miss_rate ul2.miss_rate; do
grep "$donnee" data_dij_small_* >>sim_dij_small_"$donnee".txt
done

#branchement

foo='bpred_bimod.bpred_'
for data in addr_rate dir_rate jr_rate jr_non_ras_rate.PP; do
donnee=$pref$data
grep "$donnee" data_dij_small_* >>sim_dij_small_"$donnee".txt
done

donnee='bpred_bimod.ras_rate.PP'
grep "$donnee" data_dij_small_* >>sim_dij_small_"$donnee".txt
```

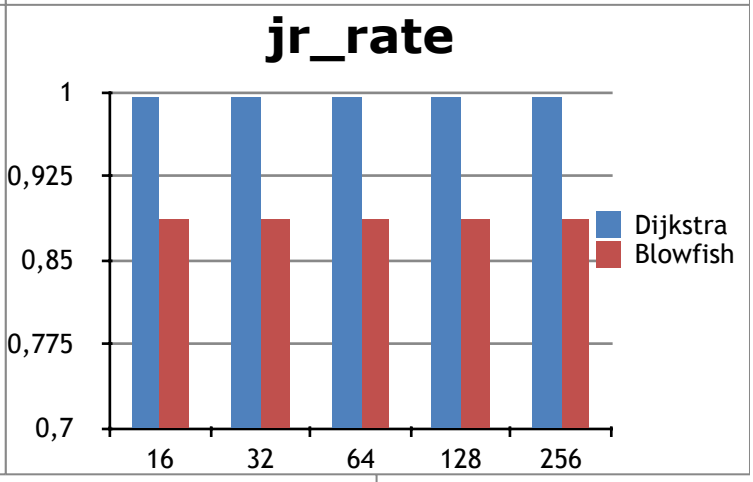
De même pour BlowFish en remplaçant l'exécutable par bf.ss et le fichier input.dat par input_small.asc. On obtient les figures de performances suivantes :

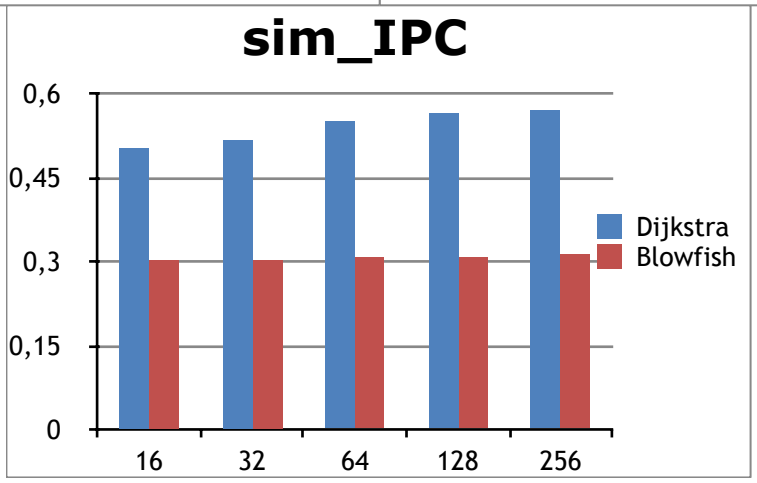






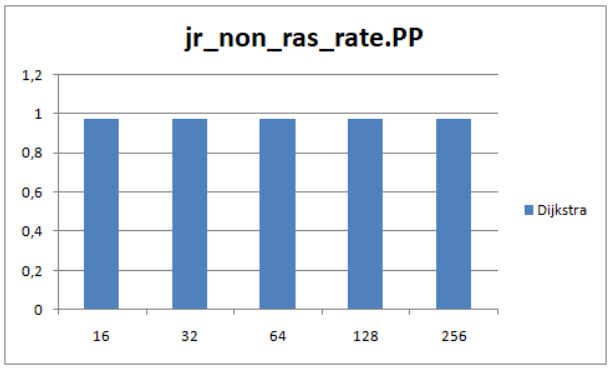


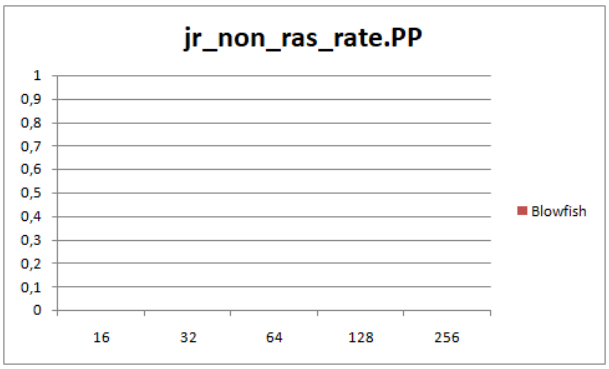


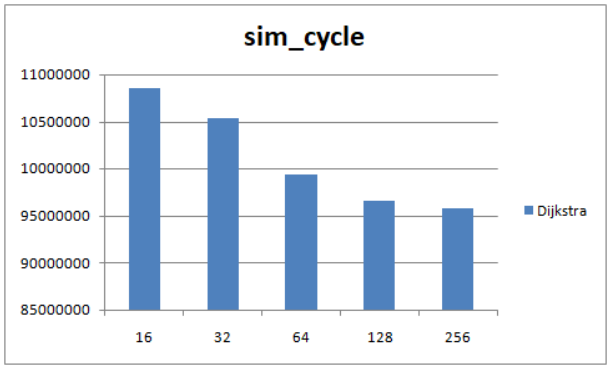


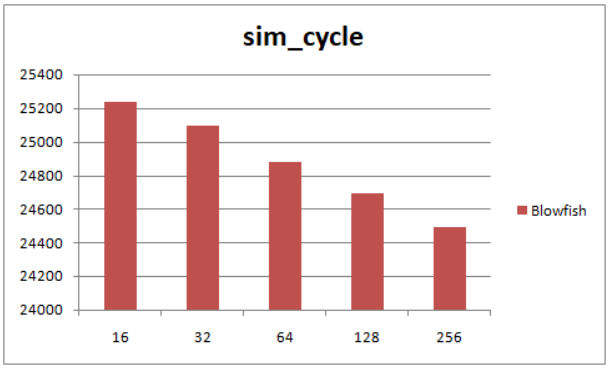
On remarque que pour les deux programmes, l'IPC augmente avec la taille des caches, ce qui implique que le nombre de cycles diminue. On observe donc un gain de performance en lien avec l'augmentation de la taille des caches. Cependant, si ce gain est significatif entre 16 KB et 64KB, il ne compense pas forcément le coût de l'architecture au-delà car il croît de moins en moins.

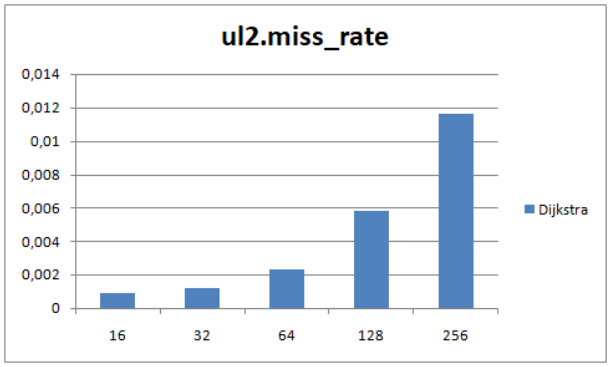
De même, le taux d'accès mémoire échoués miss_rate est inversement proportionnel à la taille des caches L1 (I et D), toutefois le gain de performance est plus prononcé dans le cas de l'application Dijkstra que Blowfish.

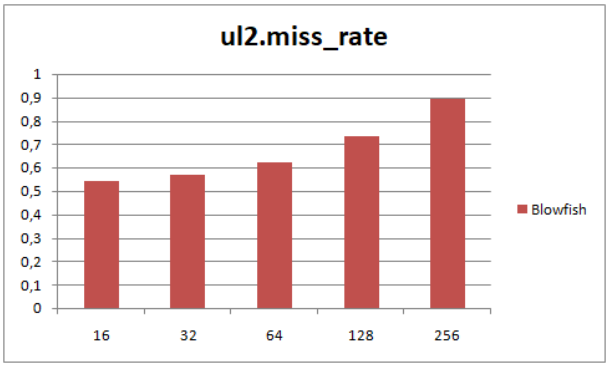


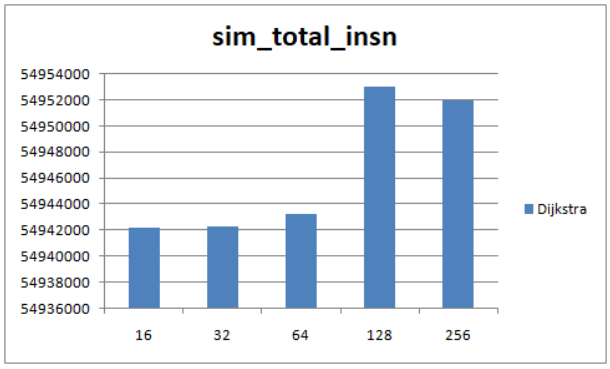


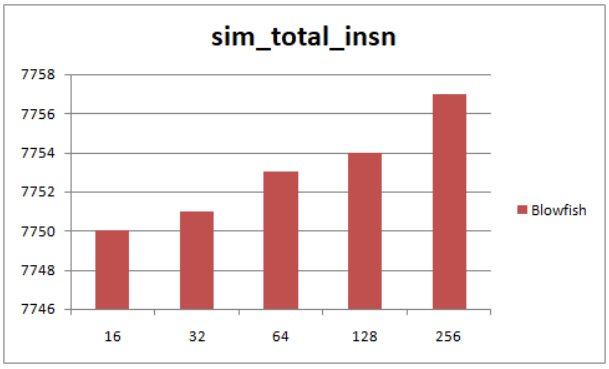










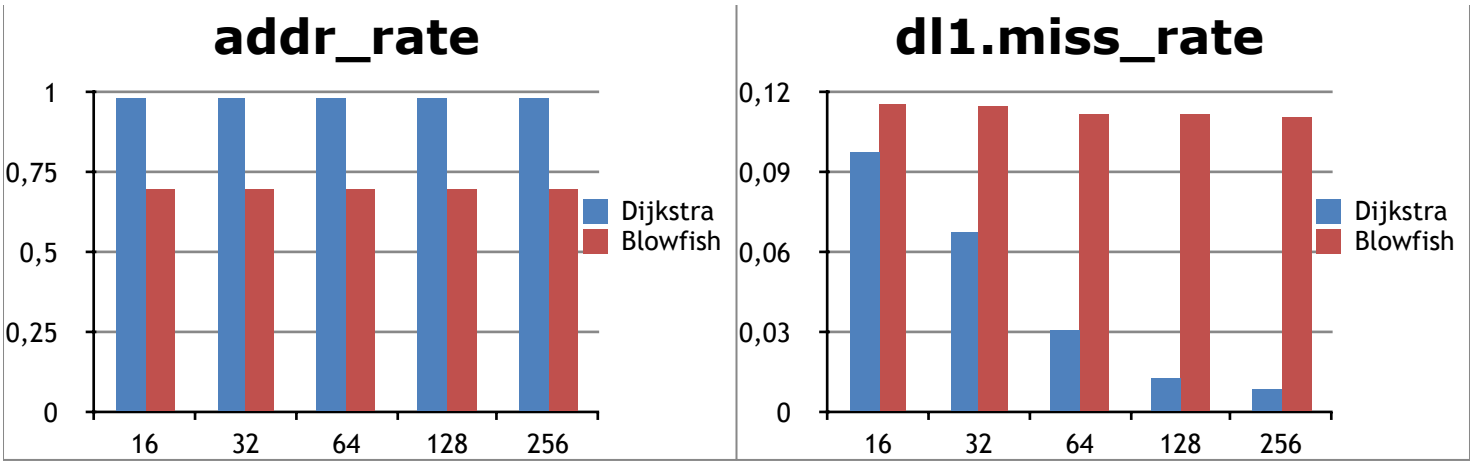


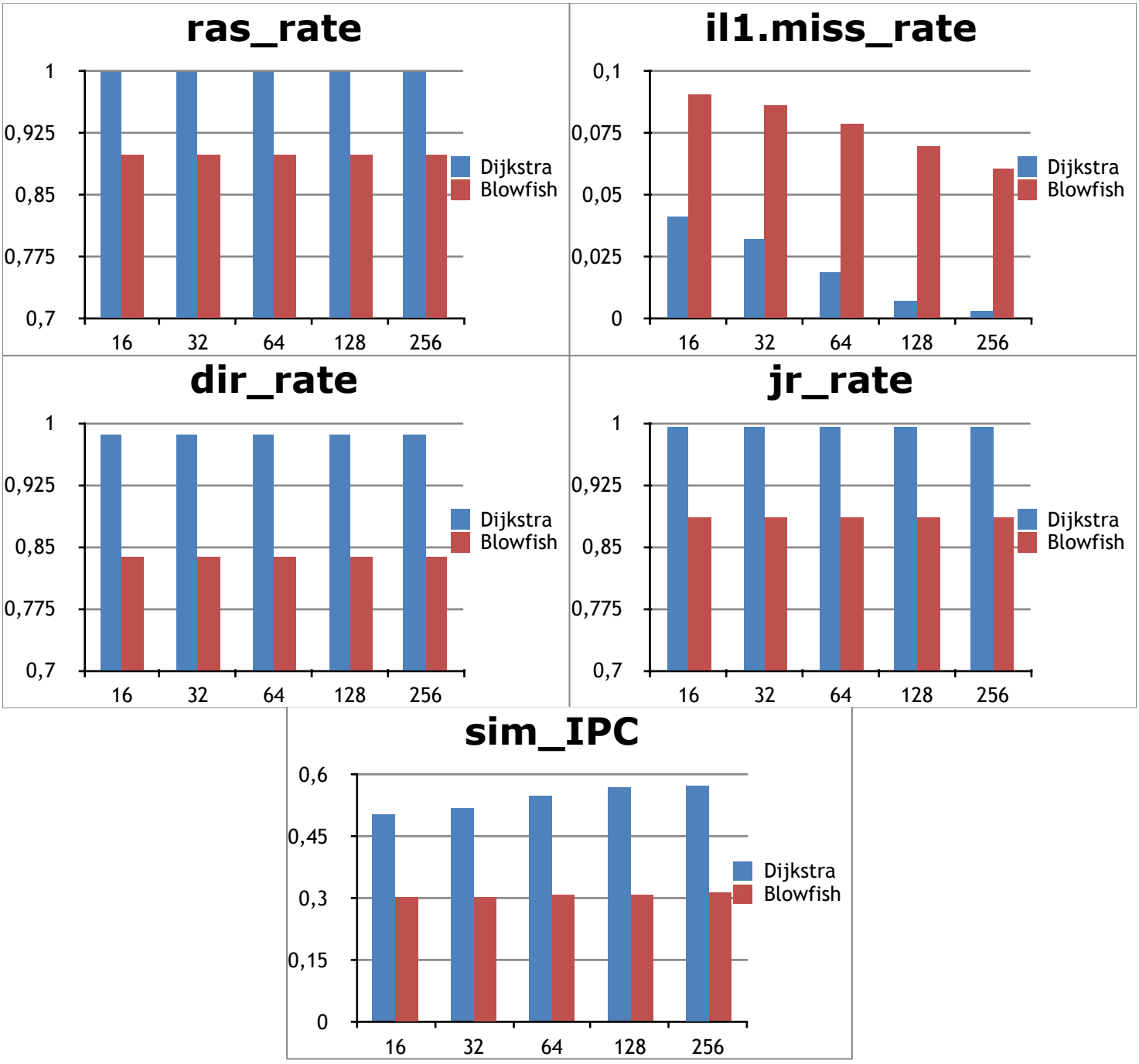
En revanche, le cache L2 tend à augmenter son taux d'échecs, ce qui peut être compensé par la baisse du taux d'erreur des caches L1 qui diminue le nombre de tentatives d'accès au cache L2. En conclusion, l'augmentation de la taille des caches L1 permet l'amélioration des performances du microprocesseur.

Q5 : On réalise les mêmes figures avec les paramètres du cortex A15 donnés dans l'énoncé, ce qui donne les commandes suivantes :

```
#!/bin/bash

export PATH=$PATH:/usr/ensta/pack/simplescalar-3v0d/bin/:/usr/ensta/pack/simplescalar-3v0d/simplesim-3.0
alias sslittle-na-sstrix-gcc='/auto/appy/ensta/pack/simplescalar-3v0d/bin/sslittle-na-sstrix-gcc'
alias sim-outorder='/auto/appy/ensta/pack/simplescalar-3v0d/simplesim-3.0/sim-outorder'
alias sim-profile='/auto/appy/ensta/pack/simplescalar-3v0d/simplesim-3.0/sim-profile'
alias sim-cache='/usr/ensta/pack/simplescalar-3v0d/simplesim-3.0/sim-cache'
clear
for taille in 16 32 64 128 256; do
    echo $taille
    for i in small; do
        sim-outorder -redir:sim data_dij_A15_"$i"_"$taille".txt -issue:inorder false -bpred bimod -bpred:2lev 256 -fetch:mplat 15
        -fetch:ifqsize 8 -decode:width 4 -issue:width 8 -commit:width 4 -ruu:size 16 -lsq:size 16 -res:ialu 5 -res:fpalu 1 -res:imult 1
        -res:fpmult 1 -cache:il1 il1:"$taille":64:2:l -cache:dl1 dl1:"$taille":64:2:l -cache:il2 dl2 -cache:dl2 ul2:512:64:16:l
        dijkstra_"$i" input.dat
    done
done
clear

#performance
for donnee in sim_IPC sim_total_insn sim_cycle; do
grep "$donnee" data_dij_A15_small_* >> sim_dij_A15_small_"$donnee".txt
done

#memoire
for donnee in il1.miss_rate dl1.miss_rate ul2.miss_rate; do
grep "$donnee" data_dij_A15_small_* >>sim_dij_A15_small_"$donnee".txt
done

#branchement

foo='bpred_bimod.bpred_'
for data in addr_rate dir_rate jr_rate jr_non_ras_rate.PP; do
donnee=$pref$data
grep "$donnee" data_dij_A15_small_* >>sim_dij_A15_small_"$donnee".txt
done

donnee='bpred_bimod.ras_rate.PP'
grep "$donnee" data_dij_A15_small_* >>sim_dij_A15_small_"$donnee".txt

```

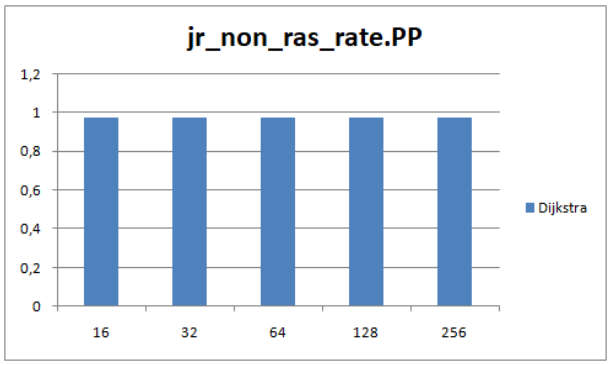
De même pour BlowFish en remplaçant l'exécutable par bf.ss et le fichier input.dat par input_small.asc. On obtient les figures de performances suivantes :

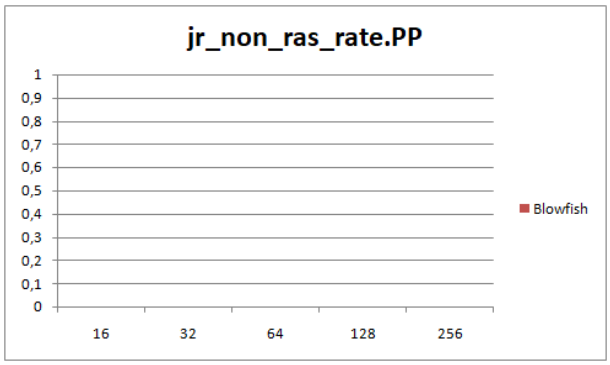


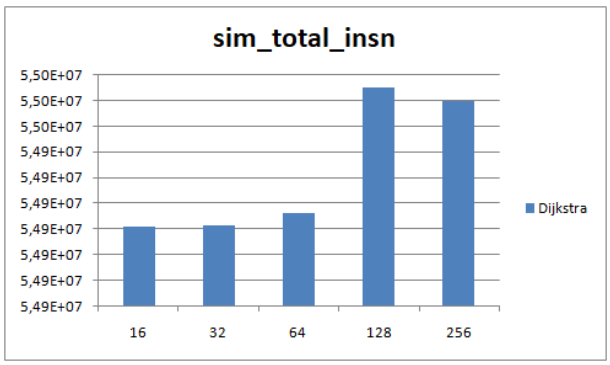


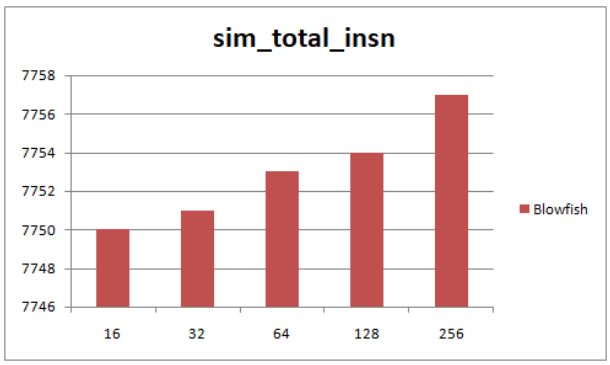
On observe dans le cas de l'application Blowfish que les performances sont peu liées à la taille des caches car l'IPC reste constant à partir de 8KB. De même, si les performances gardent la même tendance que pour le Cortex A7, notamment pour le programme Dijkstra, leur accrétion est moindre.

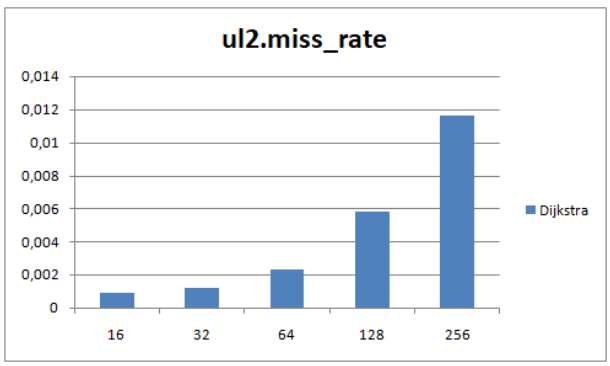
Pour ce qui est des taux addr, ras, dit et jr, on ne peut pas affirmer que l'augmentation de la taille des caches L1 se justifie. Toutefois, le miss_rate des dl1 et il1 diminue avec la taille de ces caches. L'augmentation de leur taille permet donc globalement d'améliorer les performances du processeur.

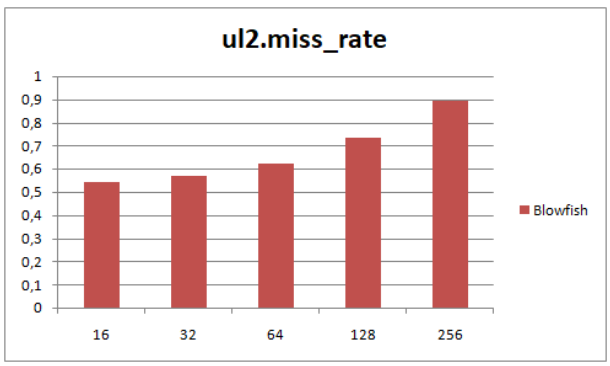


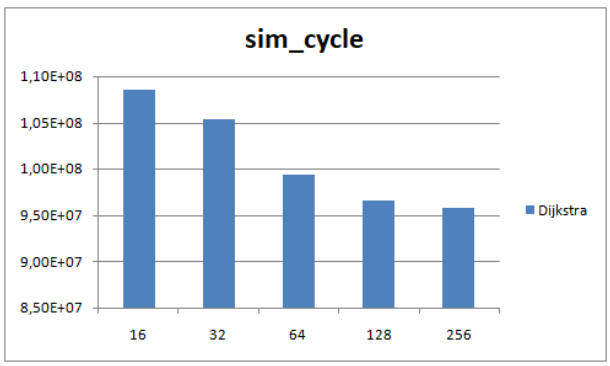


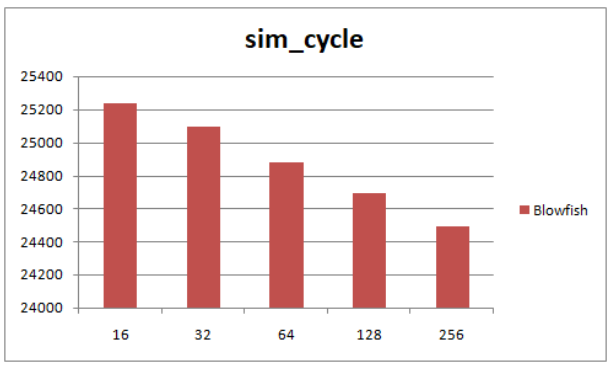












Cependant, pour Dijkstra comme pour Blowfish, on note qu'il y a peu de gain lorsqu'on passe de 16KB à 32KB pour ce qui est des taux d'erreur des dl1 et il1 alors que le taux d'erreur de l'ul2 s'envole. Ainsi on peut conclure de la même façon que pour le Cortex A7, à ceci près que pousser la taille des caches L1 jusqu'à 32KB ne se justifie plus en terme de gain de performance.

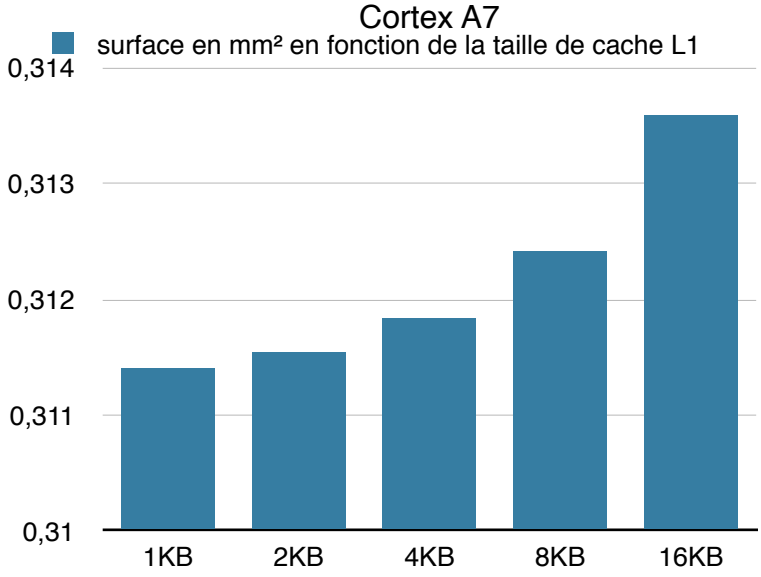
3. Efficacité surfacique

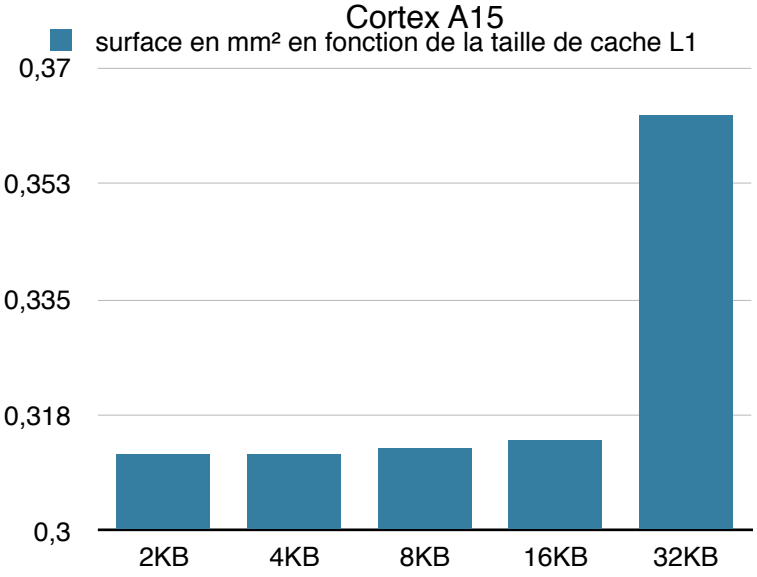
Q6 : Les paramètres de cache par défaut dans le fichier cache.cfg sont les suivants :

- taille : 134 217 728 octets
- taille de bloc : 64 octets
- associativité : 1
- technologie : 32 nm.

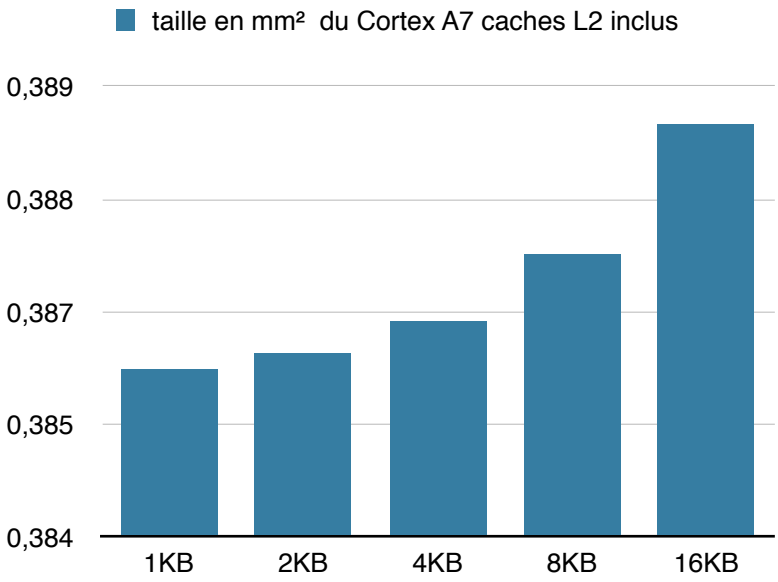
Q7 : Les cortex A7 et A15 ont la même taille de caches L1 (32KB) et la même associativité (2). Dans le fichier cache.cfg, nous ne pouvons pas utiliser la technologie 28nm, l'étude portera donc sur la technologie 32nm. On lance la commande ./cacti -infile cache.cfg pour le cortex A7 avec une taille de bloc à 32 bytes, 32KB de cache dl1, il1 et 512KB de cache l2. On obtient une surface de cache de 0.36293 mm², sachant que la surface du Cortex est de 0.45 mm², les caches L1 représentent 80.7% de la surface de Cortex. La taille du coeur, en excluant les caches L1, est de 0.08707 mm². Pour le Cortex A15 dont la taille des blocs est 64 bytes, les caches L1 ont une surface de 0.36293 mm² également soit 18.1% de la surface du Cortex. La taille du coeur sans cache L1 est de 1.63707. On observe que les caches occupent une place plus importante dans le Cortex A7 qui a une taille réduite.

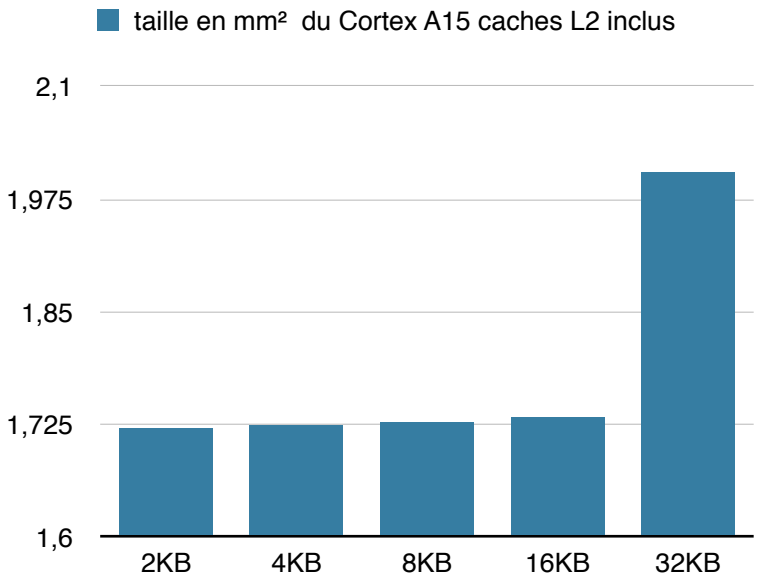
Q8 : En faisant varier la taille de cache L1 de 1KB à 16KB pour le Cortex A7 et de 2KB à 32KB pour le Cortex A15, on obtient les tailles de cache L1 suivantes :





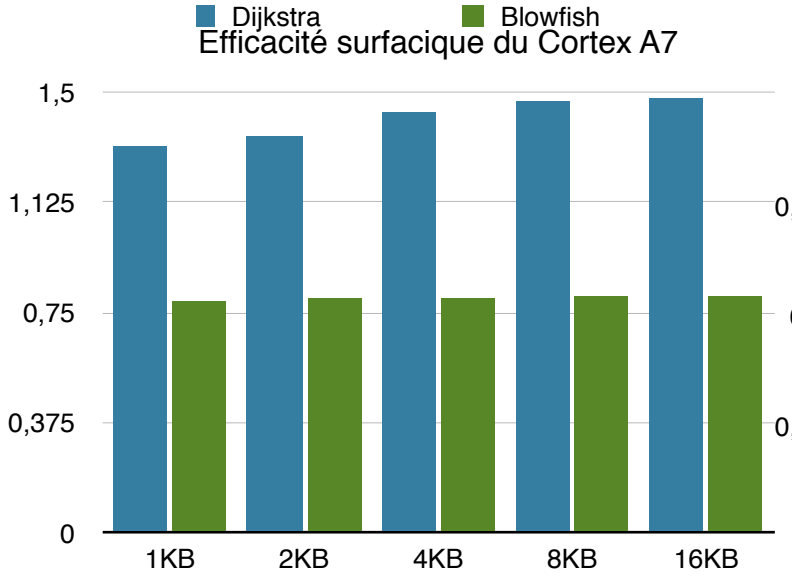
En conservant les pourcentages précédents, on obtient de nouvelles tailles pour les Cortex A7 et A15 :

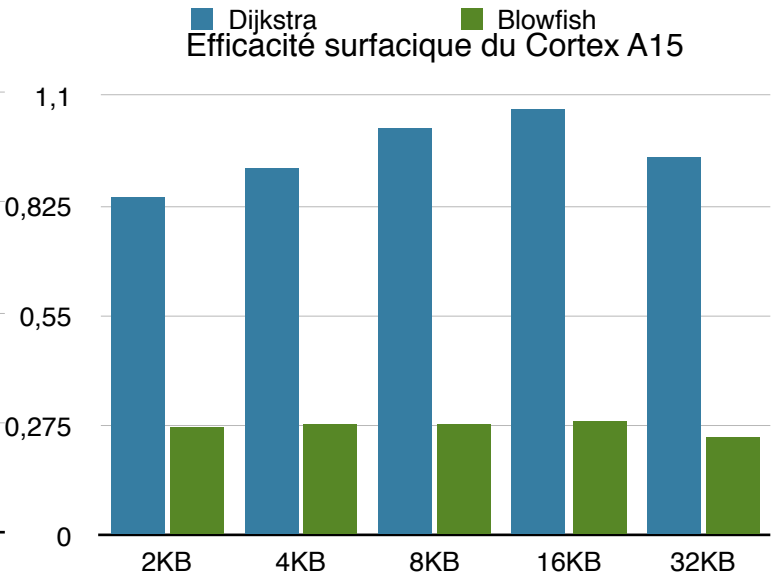




On constate que la taille du Cortex A7 croît davantage que celle du Cortex A15 sauf entre 16KB et 32KB. Ceci-dit, il est surprenant que pour des tailles de bloc différentes, on trouve des résultats semblables pour les deux Cortex.

Q9 : Grâce à la formule fournie par l'énoncé, on peut calculer l'efficacité surfacique de chaque processeur :





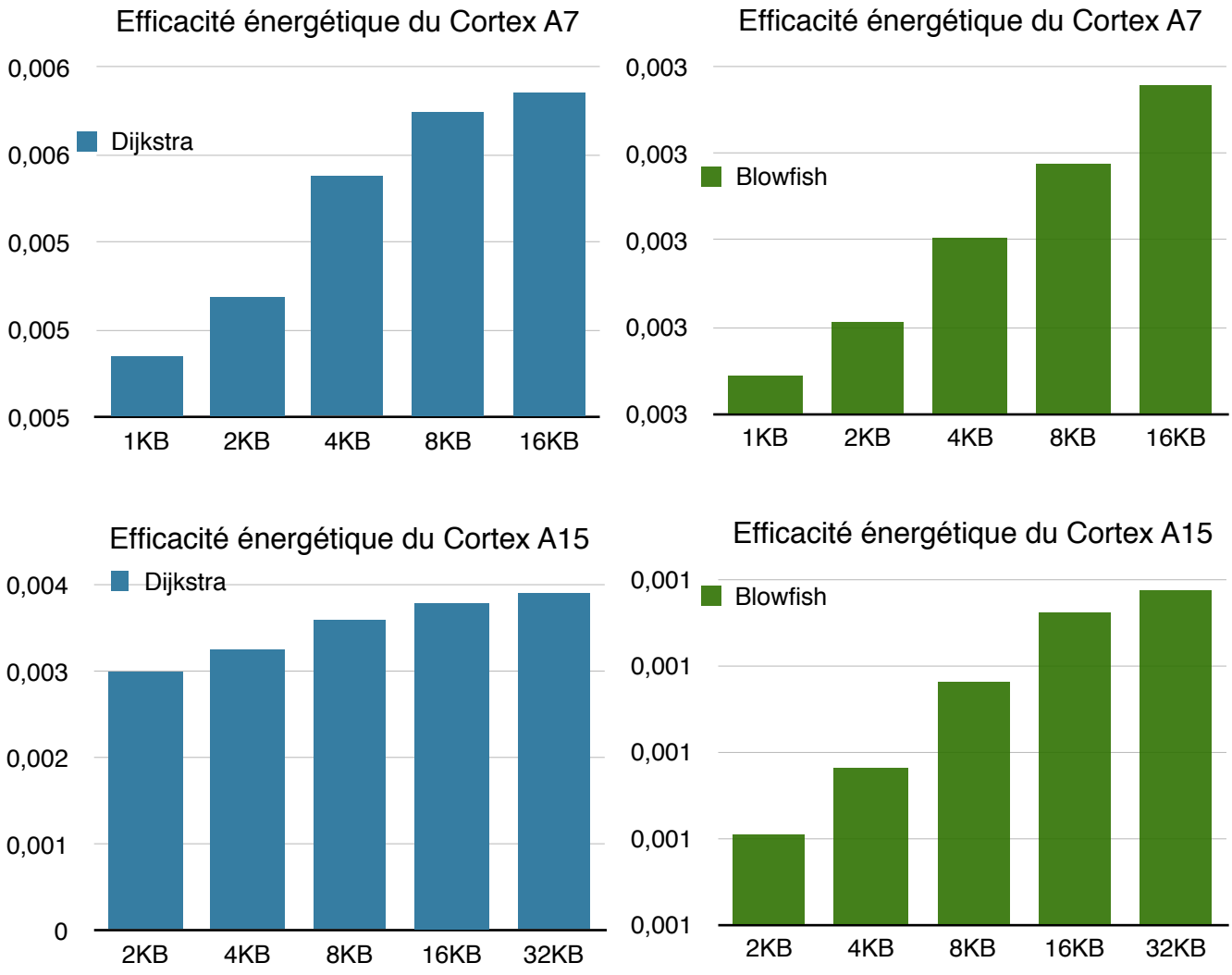
On observe la même tendance pour les deux Cortex et les deux applications, bien que l'efficacité surfacique soit plus importante pour Dijkstra que pour Blowfish. L'efficacité surfacique est la meilleure pour une taille de cache de 16KB, notamment pour le Cortex A15, ce qui

corrobore les observations précédentes sur la limitation pertinente de la taille des caches L1 du Cortex A15 à 16KB.

4. Efficacité énergétique

Q10 : D'après l'énoncé, la puissance consommée par le Cortex A7 à fréquence maximale est de 100mW et celle du Cortex A15 est de 500mW.

Q11 : Grâce à la formule fournie par l'énoncé, on peut calculer l'efficacité énergétique de chaque processeur :



Pour les deux applications, on observe une meilleure performance pour des tailles de caches de 16KB. L'amélioration des performances pour le Cortex A15 entre 16KB et 32KB n'est pas extrêmement significative et ne justifie pas, en regard de l'étude déjà menée sur les caches L1, de privilégier 32KB pour les tailles. L'aspect énergétique est très important dans le dimensionnement des microprocesseurs, il ne peut être négliger. Ici, il va dans le même sens que les conclusions précédentes.

5. Architecture système big.LITTLE

Q12 : Pour concevoir un système, il est nécessaire de limiter au maximum la consommation énergétique des composants. En effet, cette consommation d'énergie détermine l'autonomie de notre système. De ce point de vue, il est alors important de la minimiser, en essayant de maximiser son efficacité énergétique.

En regard du Dijkstra, nous avons remarqué que big.LITTLE gagne en IPC lorsque la taille du cache L1 augmente. Cette augmentation est beaucoup plus marquée sur le CORTEX A7, que sur le CORTEX A15.

Sur le CORTEX A7, il pourrait être intéressant de choisir une taille de cache de 8KB, car la différence entre le 8KB et le 16KB n'est pas significative, alors que la taille du cache est doublée, et donc la consommation d'énergie en est fortement impactée.

Tandis que pour le CORTEX A15, le 8KB serait également un bon choix, pour les mêmes raisons, à une échelle différente.

Concernant le Blowfish, les résultats sont tout à fait différents. La taille du cache influence grandement l'efficacité énergétique. Dans le cas du CORTEX A7, il serait intéressant de prendre une taille de cache égale à 16 KB pour maximiser son efficacité énergétique. Concernant le CORTEX A15, comme la différence n'est pas significative entre le 16 et le 32 KB, il serait plus bénéfique de choisir une taille de cache égale à 16 KB.