Architecture des microprocesseurs (ECE_4ES01_TA)

TD/TP4 : Analyse de configurations d'architectures

Microprocesseur superscalaire / mémoires caches

Auteurs (quadrinôme)

CHACÓN GÓMEZ José Daniel

NOM Prénom 2

NOM Prénom 3

NOM Prénom 4

Encadrant / Chargé de TD : _____

École Nationale des Techniques Avancées (ENSTA Paris)
Février 2026

Table des matières

1 Introduction

Dans ce TP, nous étudions l'impact de choix micro-architecturaux et de la hiérarchie mémoire sur les performances, la surface et l'énergie, en nous appuyant sur des simulations (gem5) et sur les applications `Dijkstra` et `BlowFish`.

2 Exercice 3 — A completer

2.1 Questions

2.1.1 Q1

Contenu de l'Exercice 3 – Q1 a completer.

2.1.2 Q2

Contenu de l'Exercice 3 – Q2 a completer.

2.1.3 Qi

Contenu de l'Exercice 3 – Qi a completer.

3 Exercice 4 — Architecture big.LITTLE : Dijkstra & BlowFish

3.1 1. Profiling de l'application

3.1.1 Q1 — Pourcentage par classe d'instructions

> **Énoncé (Q1).** Générez le pourcentage de chaque classe d'instructions de `dijkstra` et `BlowFish` (en utilisant `gem5`) et reportez les valeurs dans un tableau.

Objectif. Identifier la *répartition* des classes d'instructions (contrôle, entiers, mémoire, flottants, etc.) afin d'anticiper quelles unités fonctionnelles (ALU, LSU, unités de branchement, etc.) sont les plus sollicitées.

Méthodologie.

— Compilation RISC-V (conforme au sujet) :

```
REPO_ROOT=/path/to/ES201-TP
cd "$REPO_ROOT"
make -C TP4/Projet/dijkstra clean all
make -C TP4/Projet/blowfish clean all
```

— Simulation gem5 en mode SE (CPU OoO) — Architecture A7 :

```
GEM5=/path/to/gem5/build/RISCV/gem5.opt
"$GEM5" \
  -d TP4/Projet/q1_m5out/m5out_q1_a7_dijkstra TP4/se_A7.py \
  --cmd TP4/Projet/dijkstra/dijkstra_large.riscv \
  --options TP4/Projet/dijkstra/input.dat

"$GEM5" \
  -d TP4/Projet/q1_m5out/m5out_q1_a7_blowfish TP4/se_A7.py \
  --cmd TP4/Projet/blowfish/bf.riscv \
  --options e TP4/Projet/blowfish/input_large.asc \
  TP4/Projet/q1_m5out/output_q1_a7.enc 0123456789ABCDEF
```

— Extraction des pourcentages (tableau final Q1) :

— Extraction des compteurs par classe d'instructions depuis `m5out/stats.txt` (ou script de parsing), puis calcul :

$$\%(\text{classe } i) = 100 \times \frac{N_i}{\sum_j N_j}.$$

TABLE 1 – Pourcentage d'instructions par classe (gem5) — Dijkstra vs BlowFish (A7 et A15)

	A7		**A15**	
Classe	**Dijkstra**	**BlowFish**	**Dijkstra**	**BlowFish**
Entiers (IntAlu+IntMult+IntDiv)	68.225 357	64.424 663	68.225 357	64.424 663
Load (MemRead)	22.241 504	22.855 417	22.241 504	22.855 417
Store (MemWrite)	9.533 089	12.719 215	9.533 089	12.719 215
Contrôle (branch/jump)	0.000 000	0.000 000	0.000 000	0.000 000
Flottants (Float*)	0.000 006	0.000 091	0.000 006	0.000 091
Autres (No_OpClass, etc.)	0.000 045	0.000 614	0.000 045	0.000 614
Total	100.000 001	100.000 000	100.000 001	100.000 000

Résultats.

— **Dijkstra :** Majorité d'instructions **entiers** ($\sim$ 68%) + **mémoire** surtout en lecture (load $\sim$ 22%, store $\sim$ 9,5%).

— **BlowFish :** Entiers dominants ($\sim$ 64%) + **mémoire** plus marquée, avec **plus d'écritures** (store $\sim$ 12,7%) et load $\sim$ 22,9%.

— **Point clé :** Les deux sont dominés par **entiers** + **accès mémoire** ; BlowFish met davantage de pression sur la mémoire (stores).

3.1.2 Q2 — Catégorie d'instructions à optimiser

Énoncé (Q2). Quelle catégorie d'instructions nécessiterait une amélioration de performances ? Expliquez en quelques lignes (max 5 lignes).

Réponse. Même si les instructions **entières** dominent ($\sim$ 64–68%), une part très importante provient des **accès mémoire** (loads+stores $\approx$ 32–36%). La catégorie à optimiser en priorité est donc **Load/Store**, car elle dépend fortement de la **latence** et du **débit** des caches/RAM. **BlowFish** effectue davantage de **stores** ($\sim$ 12,7% vs $\sim$ 9,5%), ce qui accentue la pression sur la hiérarchie mémoire ; améliorer cache/bande passante/préfetch est le levier le plus rentable.

3.2 2. Evaluation des performances

3.2.1 Q4 — Cortex A7 : impact de la taille de L1 (L1I = L1D)

Énoncé (Q4). Générez les figures de performances détaillées (performance générale, IPC, hiérarchie mémoire, prédiction de branchement, etc.) en fonction de la taille du cache L1 pour les configurations testées. Analysez les résultats. Quelle configuration de L1 donne les meilleures performances pour le Cortex A7 pour `dijkstra` ? et pour `BlowFish` ?
N.B. : Mentionnez les paramètres d'exécution de Gem5 que vous avez utilisé.

Paramètres d'exécution (gem5).

— Gem5 (mode SE, ISA RISC-V) : `build/RISCV/gem5.opt`

— Script de configuration : `TP4/se_A7.py` (CPU OoO)

— Paramètres CLI utilisés : `-d <outdir>`, `-cmd <binary>`, `-options <args>`, `-l1i-size <NkB>`, `-l1d-size <NkB>`

— Balayage : `-l1i-size = -l1d-size` $\in \{1, 2, 4, 8, 16\}$ kB

— L2 fixé à 512kB.

TABLE 2 – Q4 (Cortex A7) — `dijkstra_large` : métriques vs taille L1 (L1I=L1D, L2=512kB)

L1 (kB)	IPC	CPI	Cycles (M)	I$ miss	D$ miss	L2 miss	Mispred
1	0.231858	4.312987	879.477	0.046418	0.235547	0.000280	0.010266
2	0.239406	4.177012	851.750	0.037714	0.196325	0.000333	0.010268
4	0.249901	4.001583	815.977	0.017283	0.152747	0.000445	0.010266
8	0.271426	3.684245	751.268	0.002980	0.079314	0.000903	0.010263
16	0.278733	3.587658	731.573	0.001091	0.056496	0.001294	0.010263

TABLE 3 – Q4 (Cortex A7) — `blowfish_large` : métriques vs taille L1 (L1I=L1D, L2=512kB)

L1 (kB)	IPC	CPI	Cycles (M)	I$ miss	D$ miss	L2 miss	Mispred
1	0.251957	3.968937	52.348	0.298339	0.181680	0.002570	0.024673
2	0.257932	3.876992	51.136	0.128580	0.147473	0.003898	0.024617
4	0.270185	3.701164	48.817	0.024657	0.096718	0.007233	0.024615
8	0.296596	3.371592	44.470	0.000666	0.018584	0.043697	0.024614
16	0.298072	3.354891	44.249	0.000537	0.014353	0.057827	0.024614

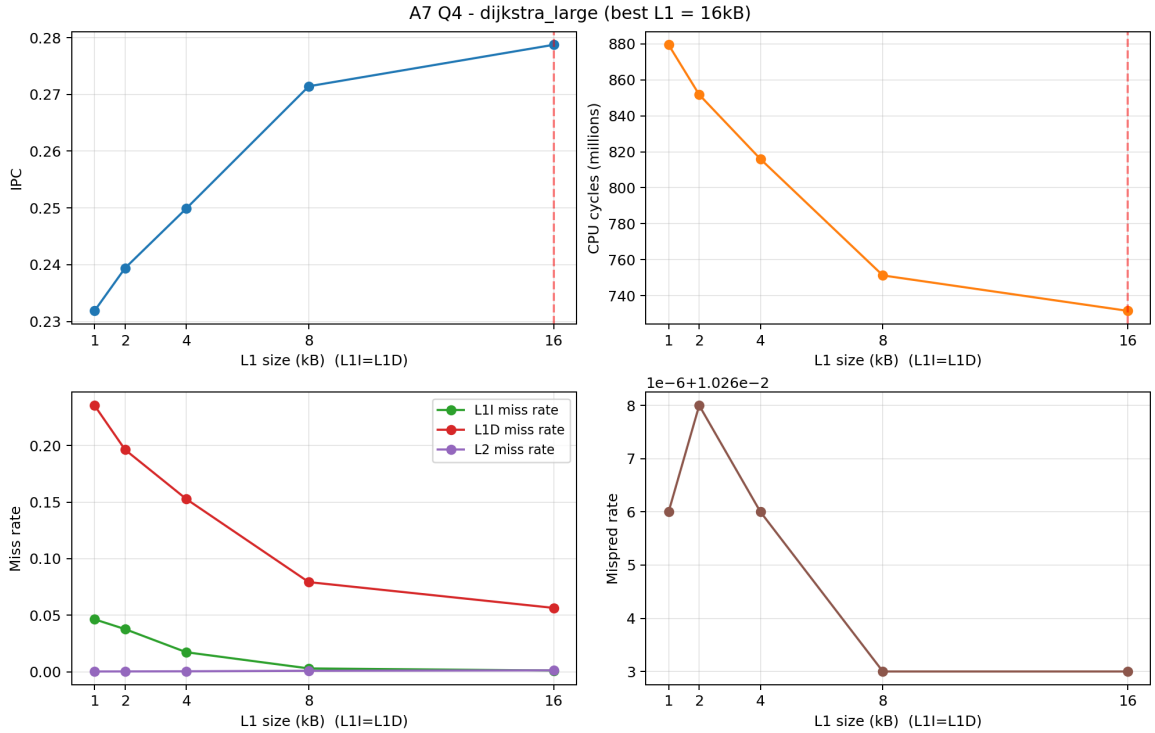

FIGURE 1 – Q4 (A7) — `dijkstra_large` : IPC, cycles, hiérarchie mémoire et prédiction de branchement vs taille L1.

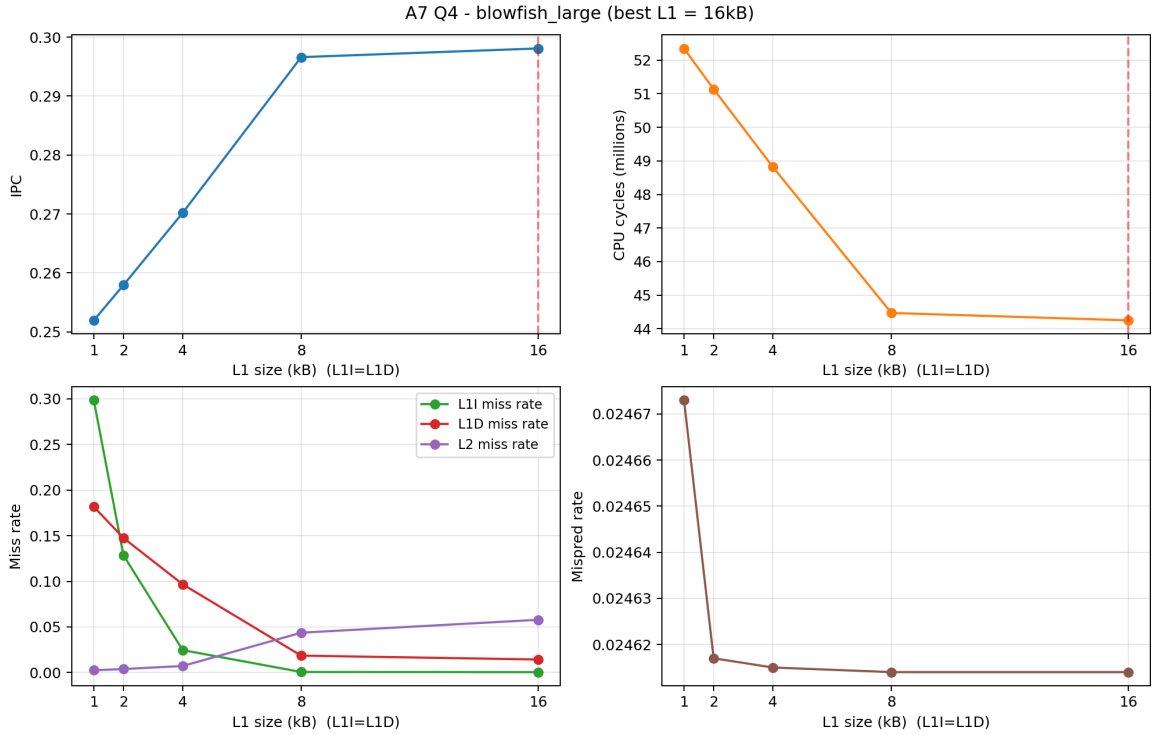

FIGURE 2 – Q4 (A7) — `blowfish_large` : IPC, cycles, hiérarchie mémoire et prédiction de branchement vs taille L1.

Analyse On observe une tendance très claire : quand on augmente la taille de L1 (avec **L1I=L1D**), les **misses I$ et D$ diminuent fortement** et, en parallèle, **l'IPC augmente**

tandis que **le nombre de cycles baisse**. Cela a du sens, car un miss de cache force le processeur à attendre (L2/mémoire), ce qui crée des *stalls* et réduit le débit global.

Pour `dijkstra_large`, on voit que la baisse des misses est progressive et continue jusqu'à 16kB, ce qui se traduit par une amélioration régulière des performances : plus de données et d'instructions tiennent en L1, donc moins d'allers-retours vers les niveaux inférieurs.

Pour `blowfish_large`, l'effet est encore plus spectaculaire sur le cache d'instructions : **I$ miss chute très rapidement** (dès 2–4 kB), et **D$ miss** baisse aussi fortement. C'est cohérent avec le fait que BlowFish exécute des boucles intensives : dès que le code et les tables de travail tiennent en cache, on réduit fortement les attentes mémoire.

Enfin, on constate que **le gain se tasse à partir de 8kB** : passer de 8 à 16 kB apporte peu de bénéfice supplémentaire. Cela suggère qu'à 8kB une grande partie du *working set* critique tient déjà dans L1, et que le reste des coûts provient d'autres goulots (accès L2/mémoire résiduels, limites du pipeline, etc.). De plus, **la misprediction reste quasi constante**, ce qui confirme que l'évolution est principalement pilotée par la hiérarchie mémoire.

Meilleure configuration (A7). Sur les points testés, la meilleure performance est obtenue avec **L1I=L1D=16kB** pour `dijkstra` et `BlowFish` (min cycles / max IPC), même si 8kB apparaît déjà comme un bon compromis au vu des gains marginaux.

3.2.2 Q5 — Cortex A15 : impact de la taille de L1 (L1I = L1D)

> **Énoncé (Q5).** Générez les figures de performances détaillées (performance générale, IPC, hiérarchie mémoire, prédiction de branchement, etc.) en fonction de la taille du cache L1 pour les configurations testées. Analysez les résultats. Quelle configuration de L1 donne les meilleures performances pour le Cortex A15 pour `dijkstra` ? et pour `BlowFish` ?
> **N.B. : Mentionnez les paramètres d'exécution de Gem5 que vous avez utilisé.**

Paramètres d'exécution (gem5).

— Gem5 (mode SE, ISA RISC-V) : `build/RISCV/gem5.opt`

— Script de configuration : `TP4/se_A15.py` (CPU OoO)

— Paramètres CLI utilisés : `-d <outdir>`, `-cmd <binary>`, `-options <args>`, `-l1i-size <NkB>`, `-l1d-size <NkB>`

— Balayage : `-l1i-size = -l1d-size` $\in \{2, 4, 8, 16, 32\}$ kB

— L2 fixé à 512kB dans le script.

TABLE 4 – Q5 (Cortex A15) — `dijkstra_large` : métriques vs taille L1 (L1I=L1D, L2=512kB)

L1 (kB)	IPC	CPI	Cycles (M)	I$ miss	D$ miss	L2 miss	Mispred
2	0.654236	1.528500	311.682	0.052157	0.185632	0.000220	0.011167
4	0.716546	1.395584	284.579	0.026767	0.143118	0.000290	0.011140
8	0.901382	1.109407	226.223	0.005627	0.077506	0.000574	0.011522
16	0.981770	1.018568	207.700	0.002244	0.051970	0.000871	0.011546
32	1.141754	0.875845	178.597	0.001183	0.017590	0.002893	0.011588

TABLE 5 – Q5 (Cortex A15) — `blowfish_large` : métriques vs taille L1 (L1I=L1D, L2=512kB)

L1 (kB)	IPC	CPI	Cycles (M)	I$ miss	D$ miss	L2 miss	Mispred
2	1.060149	0.943264	12.441	0.110876	0.176653	0.002642	0.125675
4	1.134343	0.881568	11.627	0.019981	0.124223	0.004438	0.124880
8	1.359661	0.735477	9.701	0.000680	0.034137	0.022683	0.124722
16	1.390119	0.719363	9.488	0.000507	0.029321	0.030426	0.124713
32	1.497467	0.667794	8.808	0.000448	0.001067	0.798194	0.124786

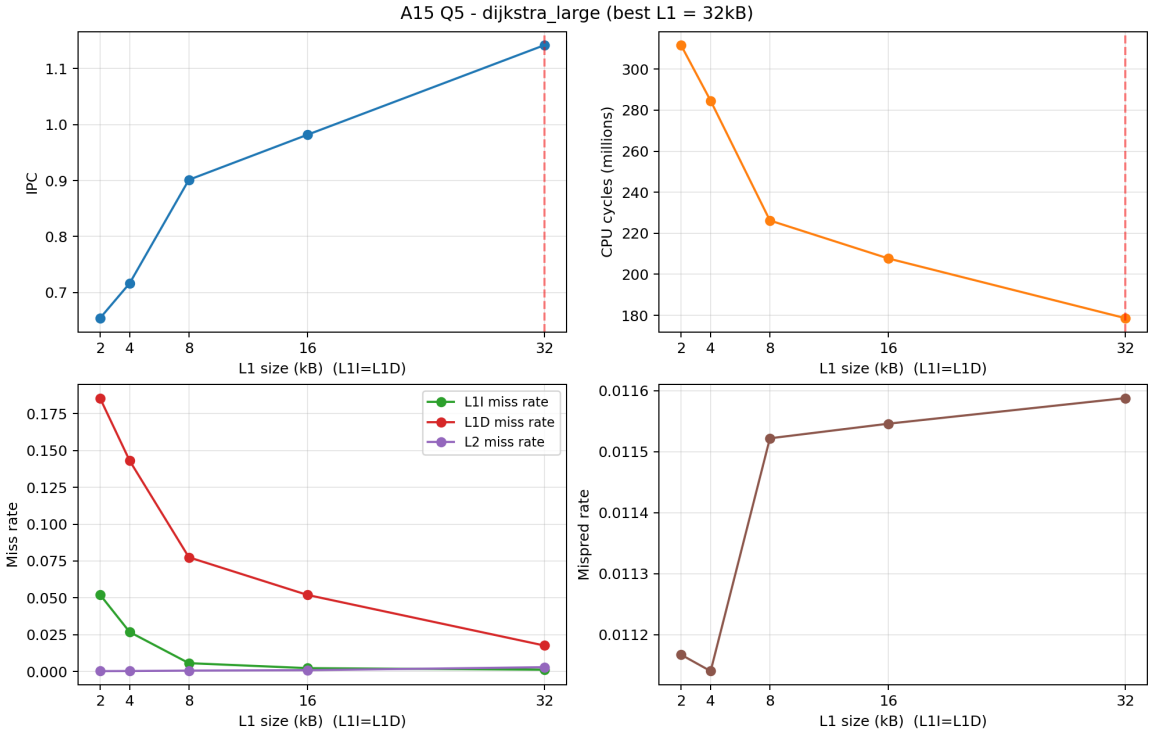

FIGURE 3 – Q5 (A15) — `dijkstra_large` : IPC, cycles, hiérarchie mémoire et prédiction de branchement vs taille L1.

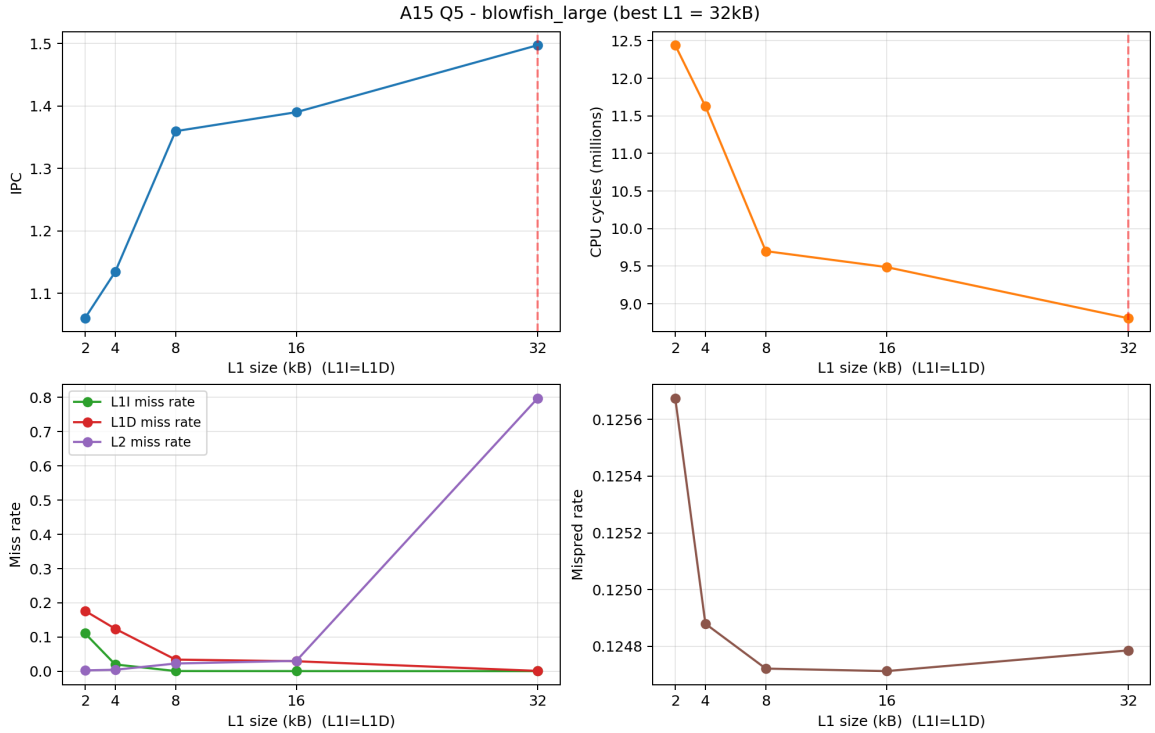

FIGURE 4 – Q5 (A15) — `blowfish_large` : IPC, cycles, hiérarchie mémoire et prédiction de branchement vs taille L1.

Analyse Les résultats des Tables 4 et 5, confirmés par les Figures 3 et 4, montrent une tendance nette : quand la taille de L1 augmente (**L1I=L1D**), **I$ miss** et **D$ miss** chutent, ce qui se traduit par **moins de cycles** et **un IPC plus élevé**. Cela a du sens, car réduire les misses réduit les attentes vers L2/mémoire et donc les *stalls*.

Pour `dijkstra_large` (Table 4, Figure 3), l'amélioration est régulière : on observe simultanément la baisse des misses (surtout **D$**) et la hausse de l'IPC, ce qui indique que l'application bénéficie directement d'un meilleur taux de hits en L1.

Pour `blowfish_large` (Table 5, Figure 4), le gain est surtout marqué entre 2kB et 8kB : les misses chutent très rapidement et l'IPC augmente fortement, ce qui correspond bien à une charge *bouclée* (code répétitif + tables) dont le *working set* tient progressivement en cache. Le taux de misprediction reste globalement stable (Figures 3 et 4), donc la performance est principalement pilotée par la hiérarchie mémoire.

Remarque (BlowFish, L1=32kB) : pic de miss L2. Sur la Figure 4 et la Table 5, le **miss L2** augmente fortement à 32kB. D'après `stats.txt`, cela s'explique surtout par un **effet de ratio** : seulement $m2547$ accès atteignent L2, et une grande partie sont des misses, alors même que D\$ miss devient quasi nul (accès majoritairement servis en L1). La tendance globale (cycles en baisse, IPC en hausse) confirme que l'effet dominant reste la chute des misses L1.

Meilleure configuration (A15). Sur les points testés, la meilleure performance est obtenue avec **L1I=L1D=32kB** pour `dijkstra` et `BlowFish` (min cycles / max IPC).