

# CMP Performance Analysis with gem5

**Javier Andres Tarazona Jimenez**

*Ingénieur Degree Programme  
STIC  
ENSTA Paris  
Paris, France  
javier-andres.tarazona@  
ensta-paris.fr*

**Jair Anderson Vasquez Torres**

*Ingénieur Degree Programme  
STIC  
ENSTA Paris  
Paris, France  
jair-anderson.vasquez@  
ensta-paris.fr*

**Maeva Noukoua**

*Ingénieur Degree Programme  
STIC  
ENSTA Paris  
Paris, France  
maeva-sandy.noukoua@  
ensta-paris.fr*

**Carlos adrian Meneses Gamboa**

*Ingénieur Degree Programme  
STIC  
ENSTA Paris  
Paris, France  
carlos-adrian.meneses@ensta.fr*

*Abstract—...*

*Index Terms—...*

## I. ARCHITECTURE MULTICOEURS AVEC DES PROCESSEURS SUPERSCALAIRES OUT-OF-ORDER (CORTEX A15)

### A. Stratégie adoptée pour traiter la Q9

Pour répondre à la Q9 (faire varier le nombre de threads et la largeur superscalaire, puis produire un graphe 3D des cycles), nous avons mis en place une chaîne reproductible en quatre scripts :

- un script d'orchestration des simulations,
- un script gem5 SE adapté au modèle A15/o3,
- un script de post-traitement pour extraire les cycles et générer la visualisation,
- un script dédié à l'extraction/calcul de l'IPC.

Cette organisation permet de lancer une campagne complète, reprendre après erreur, tracer précisément chaque exécution, et générer automatiquement le CSV et la figure 3D demandés.

### B. Script `run_q9_a15.sh`: orchestration de la campagne

#### Ce qu'il fait :

- lance toutes les combinaisons (`size`, `width`, `threads`) pour Q9 ;
- crée un répertoire de sortie par combinaison ;
- enregistre l'état d'avancement dans `state.tsv` (PENDING, DONE, FAILED) ;
- permet la reprise automatique après interruption/échec ;
- journalise chaque run dans un fichier de log dédié.

#### Comment il le fait :

- construit la commande `gem5` avec `-cpu-type=detailed`, `-o3-width`, `-num-cpus`, et les arguments du benchmark ;

- exécute les runs séquentiellement et s'arrête au premier échec pour conserver un diagnostic clair ;
- relit `state.tsv` au redémarrage pour ignorer les cas déjà DONE ;
- supporte un mode de mitigation OpenMP (`-omp-active-wait`) via un fichier d'environnement passé à `gem5`.

### C. Script `se_a15.py`: configuration gem5 pour A15/o3

#### Ce qu'il fait :

- instancie un système gem5 en mode syscall emulation (SE) ;
- configure des CPU de type `detailed` (modèle o3) ;
- applique la largeur superscalaire via `o3-width` ;
- exécute le binaire `test_omp` avec les paramètres `threads` et `size`.

#### Comment il le fait :

- s'appuie sur les options standard gem5 (`Options.addCommonOptions`, `Options.addSEOptions`) ;
- crée `num-cpus` cœurs simulés et fixe `issueWidth` pour chaque cœur ;
- configure hiérarchie mémoire/caches et lance la simulation avec `Simulation.run()` ;
- prend en charge un fichier `-env` pour injecter des variables OpenMP/libgomp si nécessaire.

### D. Script `plot_q9_cycles.py`: extraction et visualisation

#### Ce qu'il fait :

- lit `state.tsv` et sélectionne les runs DONE valides ;
- extrait les cycles à partir des `stats.txt` de gem5 ;
- génère un CSV consolidé ;
- produit le graphe 3D demandé (`threads`, `largeur`, `cycles`).

#### Comment il le fait :

- vérifie les colonnes attendues de `state.tsv` et la présence des fichiers `stats.txt` ;
- récupère la métrique `system.cpu*.numCycles` et conserve la valeur maximale par run ;
- écrit `q9_cycles.csv` puis `trace q9_cycles_3d.png` avec Matplotlib ;
- signale explicitement les combinaisons manquantes/invalides non incluses dans la figure.

#### E. Script `extract_q9_ipc.py`: extraction de l'IPC

##### Ce qu'il fait :

- extrait `sim_insts` et `numCycles` des runs DONE ;
- calcule l'IPC pour chaque configuration (`width, threads`) ;
- exporte les résultats détaillés et les maxima.

##### Comment il le fait :

- lit `state.tsv`, filtre les runs valides et ouvre chaque `stats.txt` ;
- utilise `max(system.cpu*.numCycles)` en multi-cœur, avec fallback `system.cpu.numCycles` en mono-cœur ;
- écrit `results/images/A15/q9_ipc.csv` et `results/images/A15/q9_ipc_max.csv`.

#### F. Expérimentation

L'expérimentation a été lancée avec le script `run_q9_a15.sh` en conservant les valeurs par défaut du script : `size=64`, `widths={2,4,8}`, `threads` en puissances de 2, caches activés, et sorties dans `results/A15`.

Nous visions initialement une exploration jusqu'à 64 threads. En pratique, les exécutions à forte concurrence ont présenté des SIGSEGV de `gem5` (même après le message Done du benchmark), conformément au diagnostic détaillé dans `docs/report/sections/A15_boundary.md`.

Il est important de préciser que ce Done est imprimé par `test_omp` et signifie uniquement que le calcul applicatif est terminé ; il ne garantit pas la fin correcte de toute l'exécution `gem5`, puisque l'état DONE n'est validé que si le processus `gem5` se termine avec `exit=0`.

La mitigation `-omp-active-wait` (réduction de l'usage de la voie `futex/mutex`) a été déterminante pour stabiliser les cas en largeur 4, notamment à partir de `threads=16` et au-delà ; sans cette mitigation, plusieurs combinaisons échouaient.

a) *Résultats numériques* (extraits de `results/images/q9_cycles.csv`):

Width	Threads	Cycles
2	2	1282419
2	4	768565
2	8	515053
2	16	391465
2	32	341535
4	2	801594
4	4	520290
4	8	381832
4	16	318392
4	32	302483
8	2	785008
8	4	510238
8	8	376396
8	16	315224
8	32	299612

TABLE I: Cycles d'exécution obtenus pour Q9 (size=64).

##### b) Visualisation 3D:

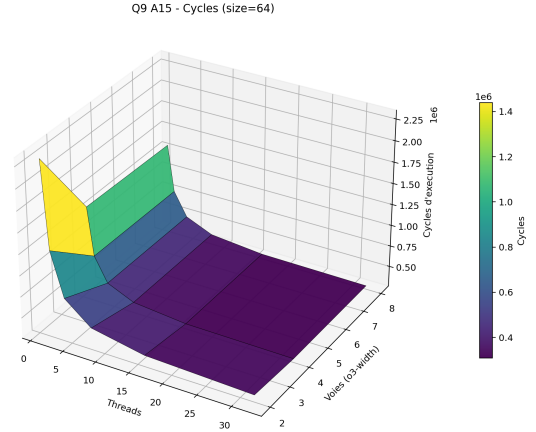


Fig. 1: Graphe 3D des cycles (X=threads, Y=voies/o3-width, Z=cycles).

c) *Cycles de référence à 1 thread* (extraits de `results/images/q9_speedup.csv`):

Width	Cycles (threads=1)
2	2308481
4	1365568
8	1334530

TABLE II: Cycles de référence utilisés pour le calcul du speedup.

d) *Calcul du speedup et résultats*: Le speedup est calculé, pour chaque largeur  $w$ , par rapport au cas `threads=1` de la même largeur :

$$S(w, t) = \frac{C_{w,1}}{C_{w,t}}$$

où  $C_{w,t}$  est le nombre de cycles de la configuration  $(w, t)$ . Dans notre cas, les valeurs obtenues sont celles de `results/images/q9_speedup.csv` :

Threads	Speedup (w=2)	Speedup (w=4)	Speedup (w=8)
1	1.000	1.000	1.000
2	1.800	1.704	1.700
4	3.004	2.625	2.616
8	4.482	3.576	3.546
16	5.897	4.289	4.234
32	6.759	4.515	4.454

TABLE III: Speedup obtenu pour Q9 (size=64), calculé à partir de `q9_speedup.csv`.

e) *IPC maximal par configuration*: Pour traiter la question sur l'IPC, nous avons créé le script `scripts/extract_q9_ipc.py`. Ce script lit `state.tsv` et les `stats.txt`, calcule :

$$IPC(w, t) = \frac{\text{sim_insts}(w, t)}{\text{cycles}(w, t)}$$

et écrit les résultats dans :

- `results/images/A15/q9_ipc.csv`,
- `results/images/A15/q9_ipc_max.csv`.

Width	Threads au max	IPC max
2	32	14.696
4	32	19.221
8	32	23.301

TABLE IV: IPC maximal pour chaque largeur (size=64).

Le maximum global observé est **IPC = 23.301**, obtenu pour width=8 et threads=32.

#### REFERENCES

- [1] TP5, "Analyse de performances de configurations de microprocesseurs multicœurs pour des applications parallèles," document de travaux pratiques du cours.
- [2] O. Hammami, *Introduction à l'Architecture des Microprocesseurs*, ENSTA ParisTech, 828 Bvd des Maréchaux 91762 Palaiseau cedex, <https://www.ensta-paristech.fr>, ENSTA PARIS - Cours ES201, Année 2022.