Analyse de performances de configurations de microprocesseurs multicoeurs pour des applications parallèles

Rapport de Travaux Pratiques

Luiz Gariglio Dos Santos
Helena Guachalla De Andrade
Santiago Florido Gomez
Franck Ulrich Kenfack Noumedem

Programme : Ingénieur STIC

Ecole Nationale des Techniques Avancees
ENSTA Paris
Fevrier 2026

Table des matières

1 Resume

Ce rapport analyse l'influence de la hiérarchie mémoire et des contraintes ènergétiques sur les performances des processeurs. Dans un premier temps, nous èvaluons l'impact des caches L1/L2 sur des calculs matriciels, en mettant en èvidence le rôle clé de la localité et de l'organisation des données. Dans un second temps, nous menons une exploration de l'espace de conception d'une architecture *big.LITTLE* (Cortex-A7/A15) afin d'optimiser les configurations pour Dijkstra et Blowfish. Les performances sont mesurées avec Gem5 (IPC, taux de *miss*) et les coûts physiques des caches sont estimés avec CACTI, compilé en version 7.0 pour fiabiliser les résultats en 32 nm.

2 Introduction

Dans le contexte actuel des systèmes embarqués et du calcul haute performance, la conception de processeurs est confrontée à deux contraintes majeures : le *mur mémoire* (*Memory Wall*), qui limite les gains de performance lorsque la hiérarchie mémoire ne suit pas le rythme du calcul, et l'efficacité ènergétique, devenue déterminante pour les plateformes modernes. Ce rapport de travaux pratiques s'articule autour de deux ètudes complémentaires visant à caractériser et quantifier ces problématiques à travers des expérimentations reproductibles.

La première partie (Exercice 3) analyse l'impact de la hiérarchie mémoire (caches L1/L2) sur des algorithmes de calcul matriciel. L'objectif est de montrer que les performances ne dépendent pas uniquement de l'algorithme, mais aussi de la manière dont les données sont organisées et accédées en mémoire, notamment via la localité spatiale et temporelle.

La seconde partie (Exercice 4) porte sur une exploration de l'espace de conception (*Design Space Exploration*, DSE) appliquée à une architecture hétérogène *big.LITTLE*. Nous dimensionnons un cluster *LITTLE* (Cortex-A7) et un cluster *big* (Cortex-A15) pour des charges applicatives distinctes (Dijkstra et Blowfish), en mettant en èvidence les compromis entre performance, coût en surface et consommation.

Pour mener ces analyses, nous nous appuyons sur deux outils principaux : Gem5, utilisé en mode *Syscall Emulation* pour simuler l'exécution et extraire des indicateurs tels que l'IPC et les taux de *miss*, et CACTI, utilisé pour estimer la surface et l'énergie des caches. Enfin, afin de garantir la fiabilité des résultats de l'Exercice 4, nous avons automatisé les campagnes de simulation et pris l'initiative de compiler CACTI 7.0, ce qui a permis de corriger des erreurs critiques observées avec la version standard lors des estimations en technologies fines (notamment 32 nm).

3 Analyse théorique de cohérence de cache

On considère que chaque thread s'exécute sur un processeur dans une architecture de type multicoeurs à base de bus et 1 niveau de cache pour l'algorithme de multiplication de matrices. Les caches privés stockent les données locales, pour réduire le temps d'accès à la mémoire, et toutes les transactions passent par le bus. Pour la cohérence de cache, le Snooping est donc utilisé pour surveiller ce bus et effectuer les mises-à-jour correspondantes.

Dans la phase d'initialisation, le thread principal remplit A et B sur le processeur 1, et les données montent dans son cache. Lorsque les autres processeurs commencent le calcul, ils émettent des requêtes de lecture sur le bus, et les lignes de cache contenant A et B passent à l'état partagé dans tous les caches.

Dans la phase d'écriture, chaque thread calcule un élément $C[i][j]$ et tente de l'écrire. Pour réussir, il doit obtenir l'exclusivité, et s'il possède la ligne en état partagé, il doit envoyer un signal d'invalidation sur le bus. De cette manière, tous les autres caches marquent la ligne comme invalide et le processeur écrivain la passe en état modifié. À la fin, le thread principal devra relire la matrice C pour valider le résultat.

Ce système pose des problèmes lorsque le nombre de processeurs augmente. Le bus est de plus en plus sollicité pour les lectures et les signaux d'invalidation, ce qui peut saturer le bus et limiter le gain de performance.

4 Paramètres de l'architecture multicœurs

En plus de la configuration des caches d'instructions (I-cache) et du cache L2 unifié (présents dans le fichier d'options), le fichier de configuration du microprocesseur permet aussi d'identifier d'autres éléments micro-architecturaux importants. Par exemple, le prédicteur de branchement indiqué est un *Tournament Branch Predictor*, c'est-à-dire un prédicteur « composé » qui combine plusieurs prédicteurs internes (par exemple un prédicteur local et un prédicteur global) et s'appuie sur un méta-prédicteur pour décider, selon le contexte, lequel des deux doit être privilégié. En pratique, ce mécanisme fournit généralement une meilleure précision qu'un prédicteur plus simple, au prix d'une complexité matérielle et d'un coût énergétique potentiellement plus élevés. On trouve également une *fetch queue*, située entre les étages *Fetch* et *Decode* (ou très proche de cette frontière), dont le rôle est de stocker les instructions ou micro-opérations déjà fetch mais pas encore décodées ; elle permet de poursuivre le fetch tant que la file n'est pas pleine lorsque le decode est ralenti par backpressure, et inversement de maintenir le decode actif en consommant les μops déjà présentes lorsque le fetch est bloqué, par exemple à cause d'un *miss* du cache d'instructions. Sa capacité est de 32 μops/thread. Le paramètre *DecodeWidth* correspond au nombre maximal de micro-opérations décodées par cycle, ici 8 μops/cycle, tandis que *IssueWidth* représente le nombre maximal de μops émises par cycle depuis la file de scheduling vers les unités fonctionnelles, également 8 μops/cycle. De même, *CommitWidth* (ou *RetireWidth*) fixe le nombre maximal d'instructions validées par cycle, soit 8 instructions/cycle. Le *Reorder Buffer* (ROB) conserve les instructions en vol afin de garantir un commit en ordre (et la gestion des exceptions précises) et comporte ici 192 entrées. L'*Instruction Queue* (IQ), aussi appelée *Issue Queue*, est la structure où attendent les instructions renommées jusqu'à ce que les opérandes soient prêts et qu'une unité fonctionnelle soit disponible, avec une capacité de 64 entrées. Enfin, la *Load Queue* (LQ) et la *Store Queue* (SQ) suivent respectivement les loads et les stores en vol pour gérer la spéculation mémoire, détecter les violations de dépendances et respecter les contraintes d'ordre interne ; chacune dispose de 32 entrées.

Les cinq paramètres configurables de référence pour la description de l'architecture du microprocesseur sont présentés dans le Tableau 1.

Table 1 – Paramètres micro-architecturaux configurables (extrait du fichier de configuration).

Paramètre	**Valeur**
`branchPred`	`TournamentBP(numThreads = Parent.numThreads)`
`fetchQueueSize`	32
`decodeWidth`	8
`issueWidth`	8
`numROBEntries`	192

Références

[1] M. J. P. (University of York), "Profiling," *Lecture Notes (4th Year HPC), University of York.* [Online]. Available : https ://www-users.york.ac.uk/~mijp1/teaching/4th_year_HPC/lecture_notes/Profiling.pdf. Accessed : Feb. 9, 2026.