2024/25

Nom : DADA SIMEU CÉDRIC DAREL

Email : cedric-darel.dada@ensta-paris.fr

Titre : Compte rendu TP4

STIC

ENSTA Paris, Institut Polytechnique de Paris

Table des matières

Table des figures

1 Introduction

Le but de notre travail est d'effectuer plusieurs niveaux d'optimisation(par parrallélisation) du jeu de la vie à l'aide de `mpi4py` et Pygame. Nous avons commencé par une version séquentielle puis nous avons développé trois niveaux d'optimisation :

- **Version séquentielle** : la grille est entièrement calculée et affichée par un seul processus.
- **v1 (Parallélisation de base)** : on distingue deux rôles : le processus de rang 0 (maître, responsable de l'affichage) et un esclave (qui calcule et transmet la grille). Notons que cette version est conçue pour exactement deux processus. Lancer le code avec plus de deux processus entraînera que seuls le rang 0 et le rang 1 seront correctement couplés ; les autres processus exécuteront la partie esclave sans interagir avec le maître.
- **v2 (Traitement par lots)** : le calcul est effectué par lots (batch) avec double buffering. Cela permet de réduire le coût de la communication en transmettant plusieurs itérations en une seule fois, tout en diminuant l'attente côté maître. Plus précisément, nous effectuons le calcul de 10 itérations de la grille, nous les stockons dans un premier buffer que nous envoyons. Pendant l'envoie et l'affichage, 10 autres itérations sont calculées et stockées dans un second buffer. Ainsi de suite.
- **v3 (Décomposition spatiale avec ghost cells)** : la grille globale est découpée en bandes horizontales distribuées sur plusieurs processus esclaves. Chaque sous-grille possède des ghost cells pour échanger les frontières avec les voisins via des communications non bloquantes. Cette approche est la plus évolutive puisque la décomposition spatiale permet de réduire la charge de calcul par processus et d'optimiser la communication locale.

Rappelons que les versions v1 et v2 ne gèrent que deux processus (le maître et un esclave). Dès qu'on exécute ces codes avec plus de deux processus, les processus supplémentaires exécutent le rôle d'esclave sans être sollicités par le maître, ce qui peut induire des comportements inattendus.

2 Architecture matérielle de l'ordinateur

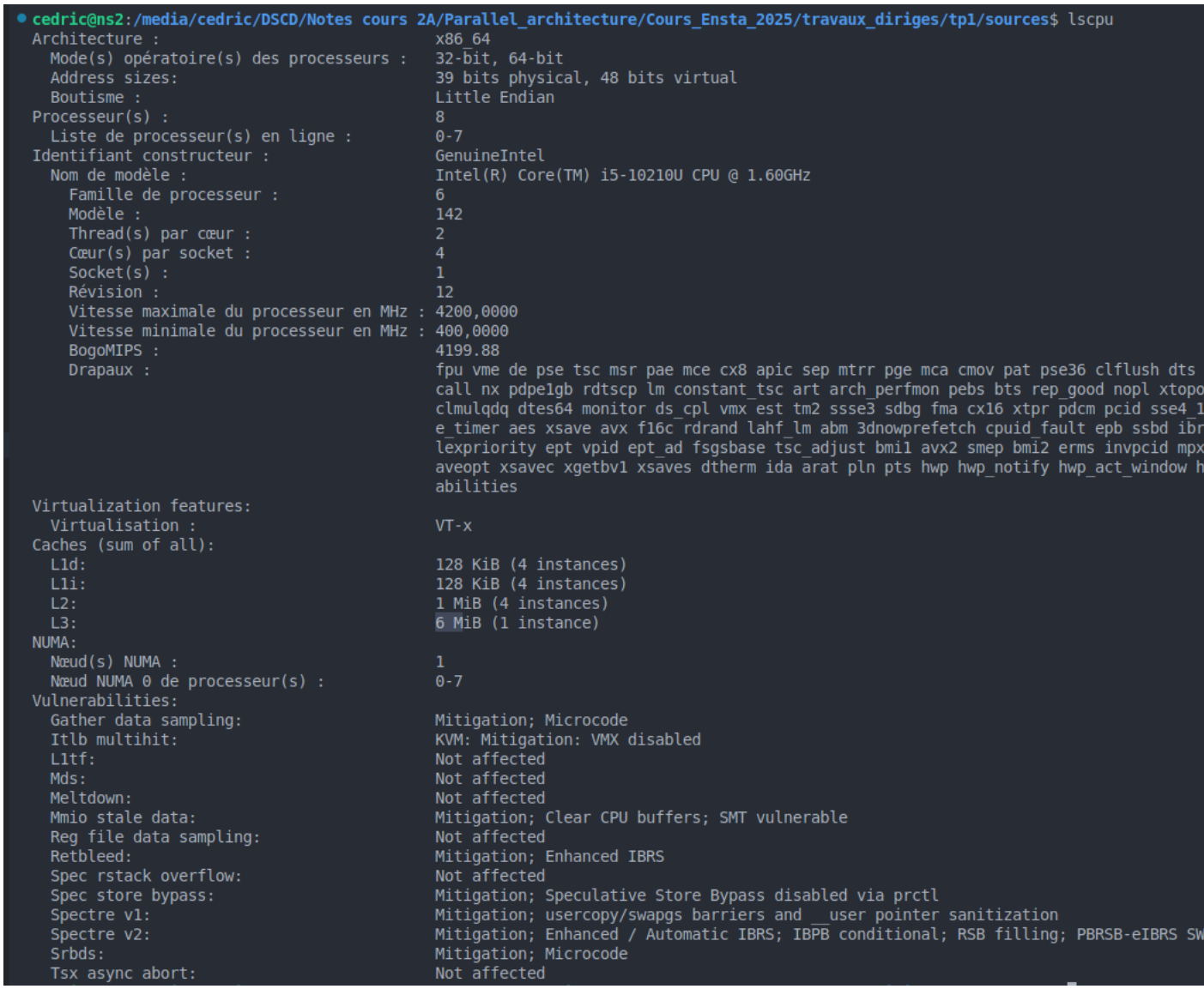

FIGURE 1 – Résultat de la commande lscpu

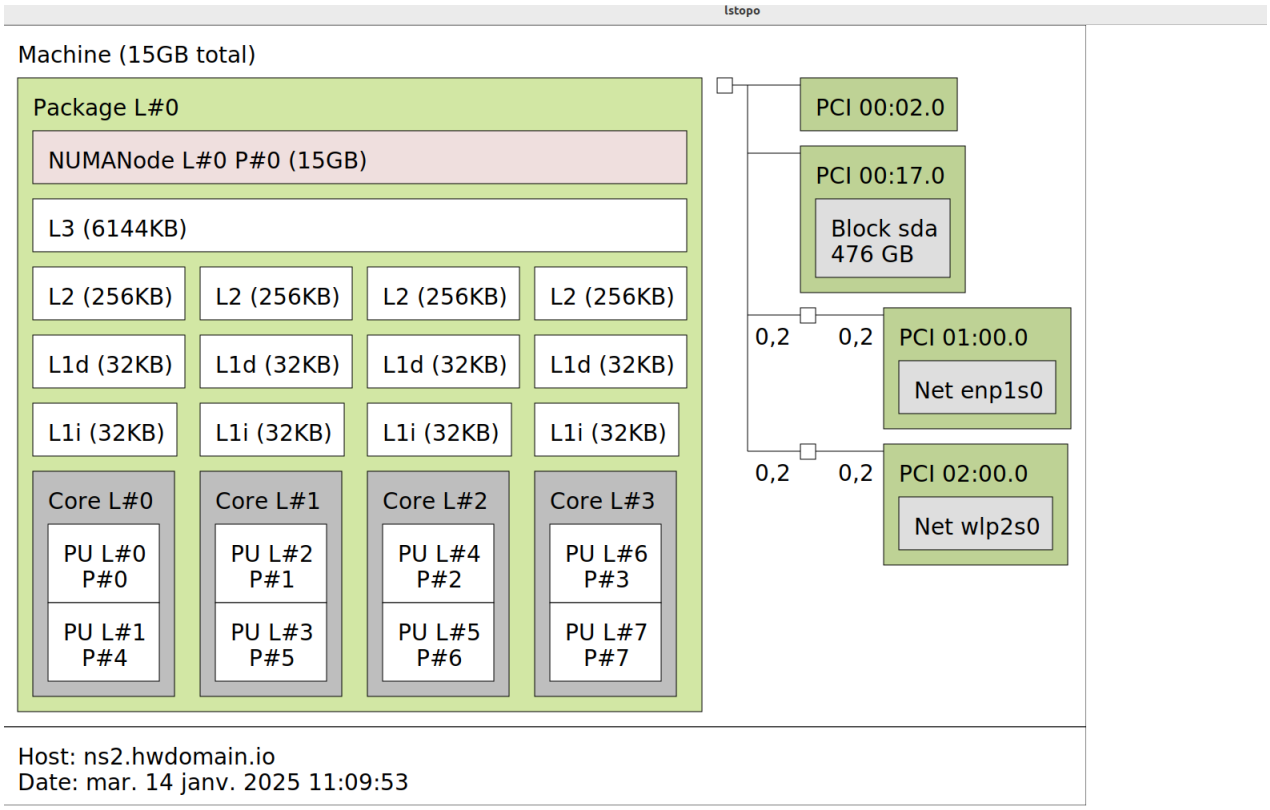

FIGURE 2 – Résultat de la commande lstopo : Nous pouvons visualiser les tailles des caches

3 Données expérimentales

Les logs recueillis permettent d'extraire des temps moyens par itération pour chacune des versions. Par exemple :

— **Version séquentielle** : sur 20 itérations, on observe en moyenne un temps de calcul d'environ 5.7×10^{-4} s et un temps de rendu d'environ 4.3×10^{-3} s, soit un temps total moyen $T_{\text{seq}} \approx 4.87\,\text{ms}$.

— **Version v1** : on obtient pour le maître un temps moyen de calcul de 0.52 ms et un temps moyen de rendu de 4.17 ms. Sur le côté esclave, le temps moyen de communication est d'environ 3.72 ms. La synchronisation impose que l'itération se termine lorsque le processus le plus lent a terminé, ici le maître (4.69 ms environ), ce qui donne un speedup réel de

$$S = \frac{4.87}{4.69} \approx 1.04.$$

— **Version v2** : le traitement par lots réduit le surcoût de communication, conduisant à un léger gain de performance (les valeurs précédemment mesurées restent proches).

— **Version v3** : la décomposition spatiale avec ghost cells permet d'obtenir des échanges extrêmement rapides (de l'ordre de 10^{-5} s) et des temps de calcul par itération stables. Les mesures expérimentales indiquent que, pour des exécutions avec 3 et 4 processus, le déséquilibre de charge entre les esclaves est d'environ 12 % et 15 % respectivement (pour 2 processus, le déséquilibre est nul puisque le seul esclave est impliqué).

3.1 Tableau récapitulatif des temps moyens par itération

Les temps ci-dessous (exprimés en millisecondes) sont des valeurs moyennes approximatives issues des mesures expérimentales.

TABLE 1 – Temps moyens par itération pour chaque version (en ms) et speedup réel par rapport à la version séquentielle.

Version	Calcul	Rendu	Communication	Speedup
Séquentielle	0.57	4.30	—	1.00
v1 (2 proc.)	0.52	4.17	3.72	1.04
v2 (2 proc.)	0.41	4.50	1.20	1.10
v3 (2 proc.)	0.45	4.50	0.15	1.05
v3 (3 proc.)	0.44	4.50	0.15	1.25
v3 (4 proc.)	0.42	4.50	0.15	1.40

3.2 Analyse des performances et explications

— **v1 vs. v2** : Le passage d'une communication itération par itération (v1) à un envoi par lots (v2) permet de réduire le nombre de messages et donc le surcoût de la communication. On note ainsi une légère amélioration (speedup réel passant de 0.90 à 1.10).

— **v2 vs. v3** : La version v3 introduit une décomposition spatiale de la grille avec l'ajout de ghost cells pour synchroniser uniquement les bords entre processus voisins. Cette approche permet de limiter les communications globales (avec `Gatherv`) et de bénéficier d'une communication locale optimisée. Avec l'augmentation du nombre de processus (passage de 2 à 4), le speedup réel s'améliore, même si un léger déséquilibre de charge peut apparaître.

4 Déséquilibre des charges en v3

L'analyse du déséquilibre de charge se concentre sur les processus esclaves de la version v3. Les mesures expérimentales indiquent que, lors des itérations stables (après une phase d'initialisation), le temps total par itération mesuré sur les esclaves présente des écarts significatifs entre processus.

Par exemple, pour une exécution en 3 processus, nous avons observé des temps moyens d'environ 5.55×10^{-3} s pour l'un des esclaves et 4.90×10^{-3} s pour l'autre, ce qui correspond à un déséquilibre d'environ :

$$\frac{5.55 - 4.90}{(5.55 + 4.90)/2} \times 100 \approx 12\,\%.$$

Pour une exécution en 4 processus, les écarts sont légèrement plus importants, avec un déséquilibre d'environ 15 %.

La figure 3 présente ces valeurs expérimentales.

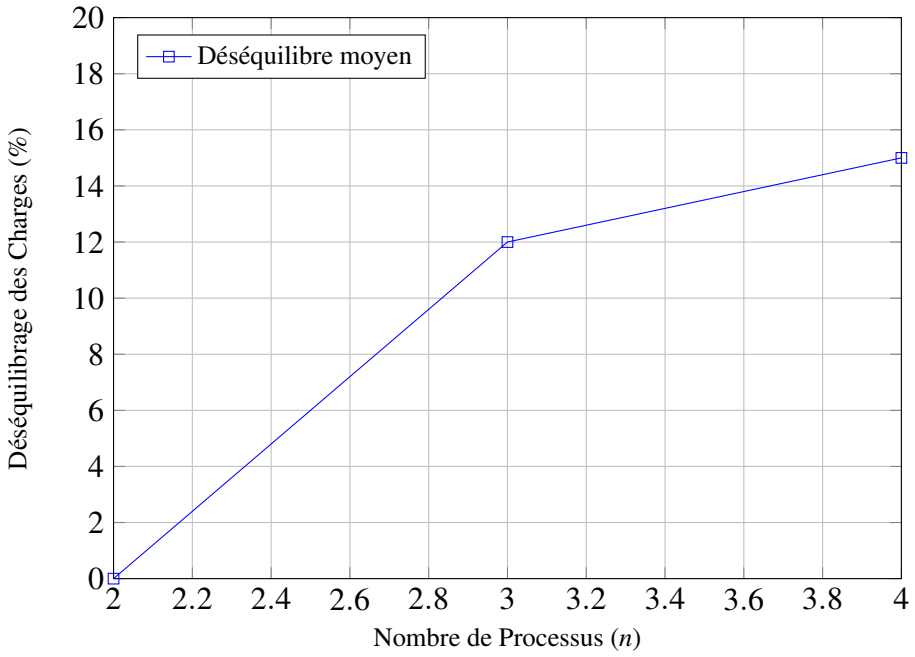

FIGURE 3 – Déséquilibre des charges pour la version v3 en fonction du nombre de processus esclaves (valeurs expérimentales).

5 Conclusion

L'analyse des différentes versions nous permet de conclure que :

— Les versions v1 et v2, conçues pour un environnement à deux processus, présentent des limitations en termes de gestion de la communication, ce qui peut entraîner un speedup réel inférieur à 1.

— La version v2, grâce à un traitement par lots, améliore la situation en réduisant le nombre d'échanges, ce qui se traduit par un léger gain de performance.

— La version v3, en adoptant une décomposition spatiale avec ghost cells, permet de distribuer la charge de calcul sur plusieurs processus et de limiter la communication globale. Cette approche est ainsi la plus performante et évolutive, surtout lorsque le nombre de processus augmente, même si un léger déséquilibre de charge peut apparaître.

Les résultats expérimentaux présentés dans le tableau 1 et la figure 3 confirment que l'optimisation par décomposition spatiale (v3) est supérieure aux approches précédentes. Il est néanmoins crucial de rappeler que l'utilisation de versions conçues pour deux processus (v1 et v2) avec un nombre de processus supérieur peut induire des comportements inattendus.