OS02 – Systèmes Parallèles et Distribués
TP : Parallélisation du Jeu de la Vie avec MPI

Noukoua Tatmfo Maëva Sandy

16 février 2026

1 Introduction

Ce TP a pour objectif d'étudier la parallélisation d'une simulation numérique à l'aide de MPI en Python (`mpi4py`). L'application choisie est le *Jeu de la Vie*, un automate cellulaire évoluant sur une grille bidimensionnelle torique.

Les objectifs sont :
— Séparer calcul et affichage dans des processus distincts.
— Mettre en œuvre un découpage de domaine 1D.
— Gérer des cellules fantômes (*ghost cells*).
— Comparer version synchrone et asynchrone.
— Mesurer et analyser les performances.

2 Le Jeu de la Vie

Le Jeu de la Vie est un automate cellulaire défini sur une grille de cellules pouvant être vivantes (1) ou mortes (0).

À chaque itération, l'état d'une cellule dépend de ses huit voisines :
— Une cellule vivante avec moins de 2 voisines vivantes meurt.
— Une cellule vivante avec 2 ou 3 voisines reste vivante.
— Une cellule vivante avec plus de 3 voisines meurt.
— Une cellule morte avec exactement 3 voisines devient vivante.

Dans ce TP, la grille est torique : les bords sont connectés entre eux.

3 Architecture parallèle

Le programme utilise MPI avec la configuration suivante :
— Rank 0 : affichage (pygame)
— Ranks 1 à P-1 : calcul parallèle

3.1 Découpage 1D

La grille globale de taille (N_y, N_x) est découpée par bandes horizontales. Chaque processus reçoit un sous-ensemble de lignes.

3.2 Cellules fantômes

Chaque processus possède :
— Ses lignes réelles
— Une ligne fantôme au-dessus

— Une ligne fantôme en-dessous

À chaque itération, les processus échangent leurs lignes frontières via `MPI Sendrecv`.

4 Vectorisation

Le calcul des voisins est effectué sans boucles Python, à l'aide de `numpy.roll`. La somme des huit voisins est calculée vectoriellement, puis les règles sont appliquées par masques booléens.

Cette approche est significativement plus rapide qu'une double boucle Python.

5 Expérimentations

Configuration utilisée :

— Pattern : `glider`

— Grille : 100×90

— Résolution fenêtre : 800×800

— Nombre total de processus : 5 (1 affichage + 4 calcul)

5.1 Mode synchrone

Extrait typique :

```
[rank2] calc=6.79e-05s  comm=1.91e-03s
[rank1] calc=9.01e-05s  comm=1.96e-03s
```

On observe :

— Temps de calcul $\approx 10^{-4}$ secondes

— Temps de communication $\approx 2 \times 10^{-3}$ secondes

Le temps de communication est environ 20 fois plus élevé que le temps de calcul.

5.2 Mode asynchrone

Extrait :

```
[rank4] calc=1.65e-04s  comm=7.74e-01s
```

Le mode asynchrone améliore la fluidité de l'affichage mais peut entraîner des comportements différents selon la gestion des messages MPI.

6 Analyse des performances

On observe que :

— Le calcul est très rapide grâce à la vectorisation.

— La communication MPI domine le temps total.

— L'affichage peut devenir un goulot d'étranglement.

— L'asynchronisme améliore la réactivité de l'interface.

Ainsi, pour une grille relativement petite (100×90), le parallélisme n'apporte pas un gain significatif car la communication domine le calcul.

7 Discussion

Le gain potentiel du parallélisme devient pertinent pour :
— Des grilles de grande taille (ex : 1000×1000)
— Des simulations longues
— Un coût de calcul supérieur au coût de communication
Ce projet illustre bien que :

$$T_{total} = T_{calcul} + T_{communication} + T_{affichage}$$

Et que l'optimisation doit porter sur la composante dominante.

8 Conclusion

Ce TP a permis de :
— Implémenter une simulation parallèle avec MPI en Python.
— Comprendre le découpage de domaine et les cellules fantômes.
— Comparer mode synchrone et asynchrone.
— Analyser l'impact des communications MPI.
Il met en évidence qu'un algorithme parallélisé ne garantit pas un gain de performance si la communication domine le coût total.