Implémentations parallélisées du Jeu de la vie

Nicolas Lopez Nieto

March 3, 2025

1 Introduction

Dans ce rapport, nous présentons trois implémentations du *Jeu de la vie* de Conway, chacune utilisant une stratégie différente de parallélisation :

- La première implémentation divise le travail en deux processus, l'un chargé du calcul de la génération suivante et l'autre de la visualisation.
- La deuxième implémentation divise l'écran en deux parties, en attribuant à chaque processus la responsabilité de calculer et de dessiner sa propre moitié.
- La troisième implémentation suit un modèle maître-travailleurs, dans lequel un processus maître gère la visualisation et plusieurs processus travailleurs calculent l'évolution de la grille.

Nous décrivons ensuite en détail les différences entre ces implémentations et analysons leur efficacité en termes théoriques.

2 Implémentations

2.1 Implémentation 1 : Calcul et Visualisation séparés

Dans cette version, deux processus travaillent de manière distincte : le premier calcule la génération suivante de cellules, tandis que le second reçoit l'état mis à jour et l'affiche à l'écran.

La communication entre les processus s'effectue via `comm.Send()` et `comm.Recv()`, ce qui facilite une synchronisation simple, bien qu'elle introduise des temps d'attente pour les deux processus. La principale limitation de ce schéma est que le processus de visualisation reste inactif pendant que l'autre effectue les calculs, réduisant théoriquement l'efficacité du parallélisme. Cependant, en pratique, cette latence est presque imperceptible grâce à l'optimisation du rendu, minimisant ainsi son impact sur les performances globales du programme.

2.2 Implémentation 2 : Division de l'écran entre les processus

Cette implémentation divise l'écran en deux régions, dans lesquelles chaque processus est responsable d'une partie : ainsi, chaque processus calcule et dessine sa propre moitié de la grille. Les informations échangées entre les processus concernent les cellules fantômes situées sur les bords communs, afin d'assurer la continuité de la simulation.

La communication s'effectue en utilisant `Irecv()` et `Send()`, ce qui permet d'effectuer des calculs pendant la réception des données, améliorant ainsi l'efficacité par rapport à l'implémentation 1. Toutefois, ce schéma peut introduire des problèmes de synchronisation si l'un des processus met plus de temps à terminer ses calculs ou à recevoir les informations.

2.3 Implémentation 3 : Un processus maître et plusieurs travailleurs

Dans cette version, les processus sont répartis en deux catégories : un processus maître, chargé exclusivement de la visualisation de la grille, et plusieurs processus travailleurs, qui calculent les générations futures. La charge de travail est répartie équitablement entre les travailleurs en fonction du nombre total de processus disponibles, permettant ainsi un traitement parallèle plus efficace.

Cette implémentation offre une meilleure évolutivité, puisque plusieurs processus calculent de manière indépendante l'évolution du jeu. La communication se fait via `Gatherv()`, ce qui implique que les données générées par les travailleurs doivent être rassemblées par le maître avant la visualisation.

Un problème potentiel de cette approche est que le processus maître peut devenir un goulot d'étranglement si la quantité de données à collecter est très importante. De plus, la synchronisation entre les processus est cruciale pour éviter des temps d'attente prolongés lors de la collecte des données.

3 Analyse Théorique

Option 1 : Calcul et Visualisation séparés
Dans ce cas, un processus est uniquement affecté au calcul tandis qu'un autre se contente d'afficher les résultats. Des fonctions telles que `comm.Send()` et `comm.Recv()` sont utilisées pour échanger les données, assurant ainsi une bonne organisation. Toutefois, le problème est que pendant qu'un processus travaille, l'autre reste inactif, attendant son tour. Cela peut ne pas être très perceptible dans de petites simulations, mais dans des scénarios plus complexes, cela peut entraîner une perte de temps.

Option 2 : Division de l'écran
Ici, l'écran est divisé en deux sections, et chaque processus est responsable à la fois des calculs et de la partie visuelle de sa zone. Cela signifie que les

deux processus peuvent être actifs simultanément, tirant ainsi mieux parti du parallélisme. Cependant, pour que la simulation reste cohérente, il est nécessaire qu'ils communiquent sur les bords qu'ils partagent (en utilisant `Irecv()` et `Send()`). Si l'un termine avant l'autre, de petits retards peuvent survenir.

Option 3 : Modèle Maître-Travailleurs
C'est l'option pensée pour les systèmes multi-cœurs. Un processus maître se consacre exclusivement à la visualisation, tandis que plusieurs processus travailleurs s'occupent des calculs, répartissant la charge de manière équitable. Bien que cette approche permette une meilleure évolutivité et une exploitation optimale des ressources, le maître doit recueillir les informations de tous les travailleurs (en utilisant `Gatherv()`), ce qui, dans de très grandes simulations, peut se transformer en un goulot d'étranglement.

4 Conclusion

- **Option 1 :** C'est la plus simple à implémenter, mais son principal inconvénient est qu'un des processus reste en attente, ce qui peut ralentir la simulation.
- **Option 2 :** Elle permet aux deux processus de fonctionner simultanément, bien que l'échange de données sur les bords puisse occasionner de légers retards.
- **Option 3 :** C'est l'option la plus évolutive et efficace sur des systèmes puissants, mais la tâche supplémentaire du maître pour rassembler les données peut devenir problématique dans des simulations de grande envergure.

En définitive, si vous disposez d'un système multi-cœurs, la troisième option est la plus indiquée pour tirer pleinement parti du parallélisme. Dans des environnements avec des ressources plus limitées, la deuxième option offre un bon compromis entre facilité d'implémentation et performance. Et, bien que la première option soit la plus simple à programmer, elle n'est pas idéale si vous cherchez à exploiter au maximum la capacité de traitement parallèle.