Parallélisation d'une simulation de feu de forêt

Qizheng WANG Mingxuan WANG

14 mars 2025

1 Première étape : Analyse initiale et OpenMP

1.1 Nombre de coeurs et mémoire cache

— Nombre de cœurs physiques : 4
— Nombre total de CPU (cœurs logiques) : 4
— Mémoire cache : Cache L1d : 384 KiB (4 instances), Cache L2 : 10 MiB (4 instances), Cache L3 : 24 MiB (2 instances)

1.2 Modifications du code dans cette étape

Dans cette première étape, nous avons introduit la parallélisation avec OpenMP en définissant le nombre de threads via `omp_set_num_threads`. De plus, nous avons intégré une mise à l'échelle de l'affichage en introduisant un facteur de zoom ajustable.

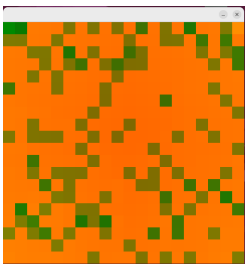

FIGURE 1 – Figure de la simulation

1.3 Résultats d'OpenMP

	nbp=1	**nbp=2**	**nbp=3**	**nbp=4**
Temps total (s)	110.254	96.0947	87.1381	90.952
Speedup pour le temps total	1.00	1.15	1.26	1.21
Temps moyen pour l'avancement (s)	0.034502	0.0161527	0.0128009	0.00979441
Speedup pour l'avancement	1.00	2.14	2.69	3.52

TABLE 1 – Résultat et Speedup

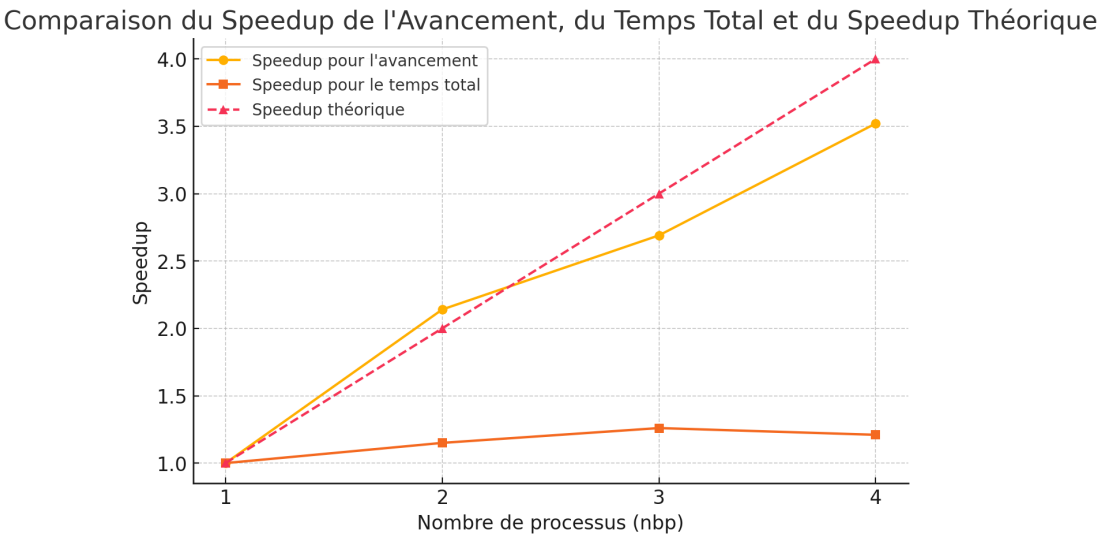

FIGURE 2 – Comparaison du Speedup de l'Avancement et du Speedup Théorique

1.4 Analyse des résultats d'OpenMP

1. Ces résultats démontrent que l'augmentation du nombre de processus améliore nettement l'efficacité du programme.

2. De plus, bien que le speedup de l'avancement soit relativement proche du speedup idéal, une analyse du temps total d'exécution montre que l'amélioration globale n'est pas aussi marquée, probablement en raison des coûts de communication et de synchronisation entre les processus.

2 Deuxième étape : MPI

2.1 Mise en place de l'environnement MPI

L'environnement MPI est intégré dans le code afin de distribuer la charge de calcul entre plusieurs processus.

2.2 Modification du code

Dans cette étape, nous avons intégré **MPI**. Le processus de rang 0 conserve la gestion de l'affichage, tandis que les autres processus exécutent uniquement les calculs.

2.3 Résultats de MPI avec 2 processus

	Origine	**2 Process**
Temps total (s)	69.2008	60.9467
Speedup pour le temps total	1	1.136
Temps moyen pour l'avancement (s)	0.000731274	0.000580028
Speedup pour l'avancement	1	1.260

TABLE 2 – Résultat et Speedup MPI

2.4 Interprétation des résultats

Les résultats montrent que l'utilisation de deux processus MPI n'entraîne pas d'amélioration significative du temps total d'exécution (*speedup* de 1.136). Cependant, l'avancement en temps bénéficie d'un léger gain (*speedup* de 1.260), ce qui suggère que la parallélisation réduit le temps

de calcul pour chaque étape. L'absence d'amélioration globale peut s'expliquer par la surcharge de communication entre les processus.

3 Troisième étape : OpenMP

3.1 Modification du code

Dans cette étape, nous avons combiné **MPI** et **OpenMP** pour paralléliser l'avancement en temps. Chaque processus MPI traite une partie du travail en fonction de son rang, tandis qu'OpenMP est utilisé pour paralléliser les calculs internes de chaque processus. Nous avons également ajouté des directives `#pragma omp atomic` afin de garantir la synchronisation des mises à jour des variables partagées, réduisant ainsi les conflits entre threads.

3.2 Accélération globale et celle de l'avancement en temps

	1 thread	**2 threads**	**3 threads**	**4 threads**
Temps total (s)	101.213	93.559	92.006	89.666
Speedup global	1.000	1.082	1.100	1.129
Temps moyen avancement (s)	0.024255	0.014243	0.011589	0.009319
Speedup avancement	1.000	1.703	2.093	2.603

TABLE 3 – Résultat et Speedup OpenMP

Note : Dans notre code, le processus de rang 0 participe également au calcul, ce qui permet d'exécuter le programme avec 1, 2, 3 ou 4 processus.

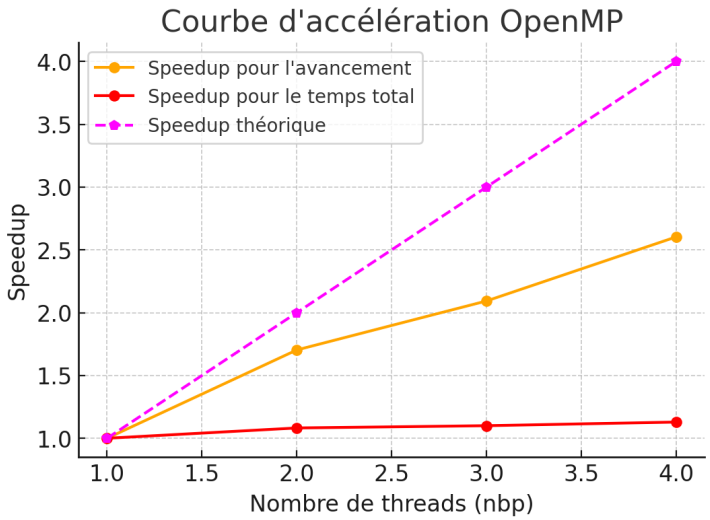

FIGURE 3 – Speedup du temps total et de l'Avancement de MPI

3.3 Interpretation des résultats

— **Pour l'accélération globale :** le speedup passe de 1.00 à 1.129 avec 4 threads, c'est une amélioration assez modérée. Cependant, cette accélération n'est pas linéaire. Ça peut être une surcharge liée à la synchronisation, ou bien l'affichage.

— **Pour l'avancement en temps :** le speedup atteint 2.603 avec 4 threads, soit une accélération plus marquée que celle du temps global. Cela indique que la parallélisation est plus efficace pour la phase de calcul que pour la gestion complète de la simulation,

suggérant que d'autres facteurs influencent le temps total, comme l'affichage ou les accès mémoire.

4 Quatrième étape

4.1 Expliaction du code

L'implémentation de l'étape 4 s'est avérée particulièrement complexe en raison de la gestion des communications entre processus MPI. Malgré nos efforts pour répartir efficacement les calculs et synchroniser les échanges via des cellules fantômes, les résultats obtenus sont incohérents, suggérant une mauvaise mise à jour des données entre processus.

Nous avons exploré plusieurs solutions sans avoir la bonne simulation. L'origine du problème pourrait résider dans une mauvaise synchronisation des frontières ou dans une mauvaise répartition des indices.

4.2 Résultats et comparaison

Note : Les presultats sont faux.

	1 process	**2**	**3**	**4**	**8**
Temps moyen d'avancement(s)	0.000651	0.000493	0.000369	0.000277	0.000463
Temps total simulation(s)	67.498	60.870	63.384	52.395	108.384

Table 4 – Résultats expérimentaux de l'étape 4

4.3 Suggestions d'amélioration

Une analyse plus approfondie des communications MPI et de la gestion des données distribuées est nécessaire.