



Module OS202 : systèmes Parallèles et Distribués

**Projet final**

---

# Parallélisation d'une simulation de feu de forêt

---

*Elaborée par :*  
CHENXI YANG

Année universitaire : 2024/2025

# Table des matières

<b>1</b>	<b>Analyse des performances et parallélisation</b>	<b>2</b>
1.1	L'information de base de l'ordinateur . . . . .	2
1.2	Le temps moyen de l'exécution . . . . .	2
1.3	Parallélisation avec OpenMP . . . . .	3
<b>2</b>	<b>Parallélisation avec MPI</b>	<b>4</b>
2.1	Description de l'implimentation . . . . .	4
2.2	Analyse de résultat . . . . .	4
<b>3</b>	<b>Optimisation de la parallélisation</b>	<b>5</b>

# Chapitre 1

## Analyse des performances et parallélisation

### 1.1 L'information de base de l'ordinateur

Nous pouvons utiliser la commande `wmic cpu get` pour obtenir le nombre de cœurs physiques et les tailles des caches, ou bien consulter directement le Gestionnaire des tâches dans l'onglet Performance, où l'on peut voir les informations matérielles de l'appareil, y compris le nombre de cœurs physiques, le nombre de processeurs logiques et la taille des caches L1/L2/L3.

Information de base de l'ordinateur	
Nombre de cœurs physiques	12
Cache L1	1.1 MB
Cache L2	9.0 MB
Cache L3	18.0 MB

TABLE 1.1 – Information de base de l'ordinateur

### 1.2 Le temps moyen de l'exécution

En effectuant quelques modifications du programme `Model::update()` dans le fichier `model.cpp`, nous pouvons obtenir le résultat suivant, où le temps d'exécution moyen par time step est d'environ **782 $\mu$ s**. Ici, le temps total de la simulation du incendie avec  $n=100$  de cases par direction pour la discrétisation et la position du foyer initial  $s = (50,50)$ .

```

Temps pour une étape : 2.31e-05 secondes
avg step time: 0.000782278
Total simulation time: 74.5012 seconds

```

FIGURE 1.1 – Résultat séquentiel

### 1.3 Parallélisation avec OpenMP

On inclut `<omp.h>` et implémente la parallélisation dans `Model::update()`.

	n thread = 1	n thread = 2	n thread = 4	n thread = 8
Temps moyen de chaque avancement (ms)	121.9	85	37.8	29.7
Temps total (s)	159.5	134.2	104.6	88.9
speedup de temps moyen	1.00	1.43	3.22	4.10
speedup de temps total	1.00	1.19	1.52	1.79

TABLE 1.2 – Résultat de la parallélisation avec OpenMP

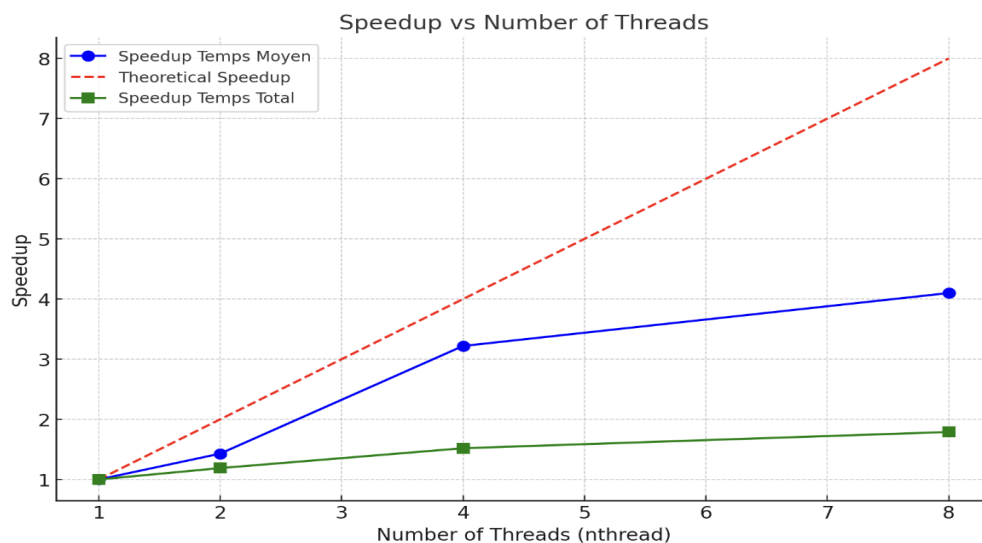


FIGURE 1.2 – Speedup de l'avancement

On peut voir que la parallélisation avec OpenMP améliore grandement les performances du programme. Mais il faut noter que le speedup n'est pas toujours linéaire avec le nombre de threads, l'amélioration de la parallélisation n'est pas significative par rapport au code original exécuté séquentiellement, ce qui peut être dû aux coûts de communication et de synchronisation entre les processus.

# Chapitre 2

## Parallélisation avec MPI

### 2.1 Description de l'implimentation

Sur la base de la première étape(multi-threading), nous utilisons MPI pour paralléliser plusieurs processus.

Nous avons principalement modifié la fonction principale du fichier `simulation.cpp`. Ici, le processus 0 est responsable de la visualisation. Il effectue d'abord `MPI_Recv`, puis met à jour l'affichage, reçoit la carte d'incendie des autres processus via une boucle, puis utilise `Displayer` pour le rendu. Tandis que les autres processus se contentent de calculer, `simu.update()` calcule le nouvel état d'incendie et `MPI_Send` envoie la carte d'incendie au rang 0.

### 2.2 Analyse de résultat

Avec commande `mpiexec -n 2 .\simulation -n -s`, nous obtenons les résultats suivants :

	n processus = 1	n processus = 2
Temps moyen de chaque avancement (ms)	37.8	38.3
Temps total (s)	104.6	104.1
speedup de temps moyen	1.00	0.98
speedup de temps total	1.00	1.005

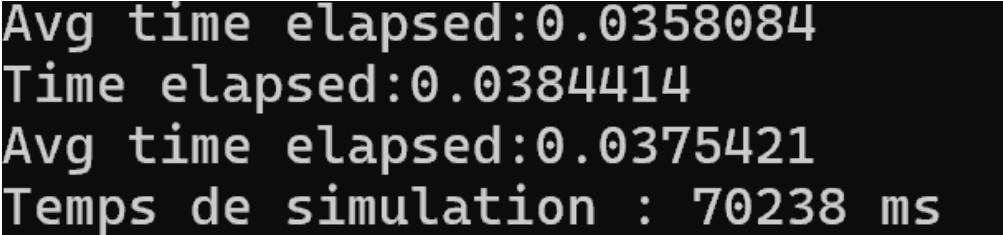
TABLE 2.1 – Résultat de la parallélisation avec MPI en cas de 4 threads

Nous constatons que le multiprocessus MPI optimise le temps global, mais le temps d'avancement moyen ne s'améliore pas significativement avec l'augmentation du nombre de processus. Cela peut s'expliquer par le coût de communication entre les processus.

## Chapitre 3

# Optimisation de la parallélisation

Malheureusement, nous n'avons pas implémenté les cellules fantômes, mais pour l'optimisation parallèle, nous avons changé le mode d'utilisation de `MPI_Send` et `MPI_Recv` (envoi et réception) dans la deuxième question en utilisant `MPI_Bcast` (mode de diffusion) et avons obtenu une amélioration des performances.



```
Avg time elapsed:0.0358084
Time elapsed:0.0384414
Avg time elapsed:0.0375421
Temps de simulation : 70238 ms
```

FIGURE 3.1 – MPI broadcast

C'est parce que le mode de diffusion garantit que tous les processus peuvent continuer à exécuter l'étape suivante après avoir reçu les données de diffusion. Tous les processus seront traités de manière synchrone une fois la diffusion des données terminée, évitant ainsi les situations asynchrones. Mais dans les modes `MPI_Send` et `MPI_Recv`, la gestion de la synchronisation entre les processus nécessite du code supplémentaire pour garantir que chaque processus reçoit les données à temps, ce qui peut entraîner des problèmes de synchronisation plus complexes, en particulier lorsque le nombre de processus est important.