Université de Versailles
Saint-Quentin-en-Yvelines
Département de Informatique

Master 1 Calcul Haute Performance, Simulation

PPN- OpenMP

Réalisé par :
- Bouchelga Abdeljalil
- Mohamed Aitmhand

Encadré par :
- Dr. Hugo Bolloré

Année Universitaire :2021-2022

TABLE DES MATIÈRES

TABLE DES FIGURES

PARTIE 1

INTRODUCTION

1.1 OpenMP (Open Multiprocessing)

Les ordinateurs personnels modernes sont équipés de processeurs multicœurs, ce qui permet l'exécution simultanée de deux flux d'instructions indépendants. Grâce à cette fonctionnalité, vous pouvez doubler les performances sur un processeur double cœur, quatre fois sur un processeur quadricœur, etc.

Il existe plusieurs technologies pour implémenter des threads dans une application. Pour cela nous avons la bibliothéque **OpenMP** fournit un ensemble limité de directives pour programmation multi-thread :

- Création de sections parallèles.
- Répartition des itérations de boucle entre les threads.
- Répartition de charge.

OpenMP signifie Open Multiprocessing, c'est une bibliothèque de programmation parallèle implémentée en tant qu'extensions de compilateur. La bibliothèque comprend une description des directives, des fonctions et des variables d'environnement.

1.1.1 Utilisation d'OpenMP

Le travaille avec la bibliothèque OpenMP est comme suit : Nous utilisons des directives du compilateur aux endroits requis pour dire au compilateur quoi faire avec le programme séquentiel. L'vantage c'est que cette approche garantit l'unicité du code, c'est à dire le programme série et parallèle se ressemblent. Et quand un compilateur qui ne prend pas en charge OpenMP ignorera les directives inconnues.

OpenMP utilise le parallélisme "pulsé" **(fork-join)** : le nombre de threads exécutés en parallèle peut changer pendant l'exécution.

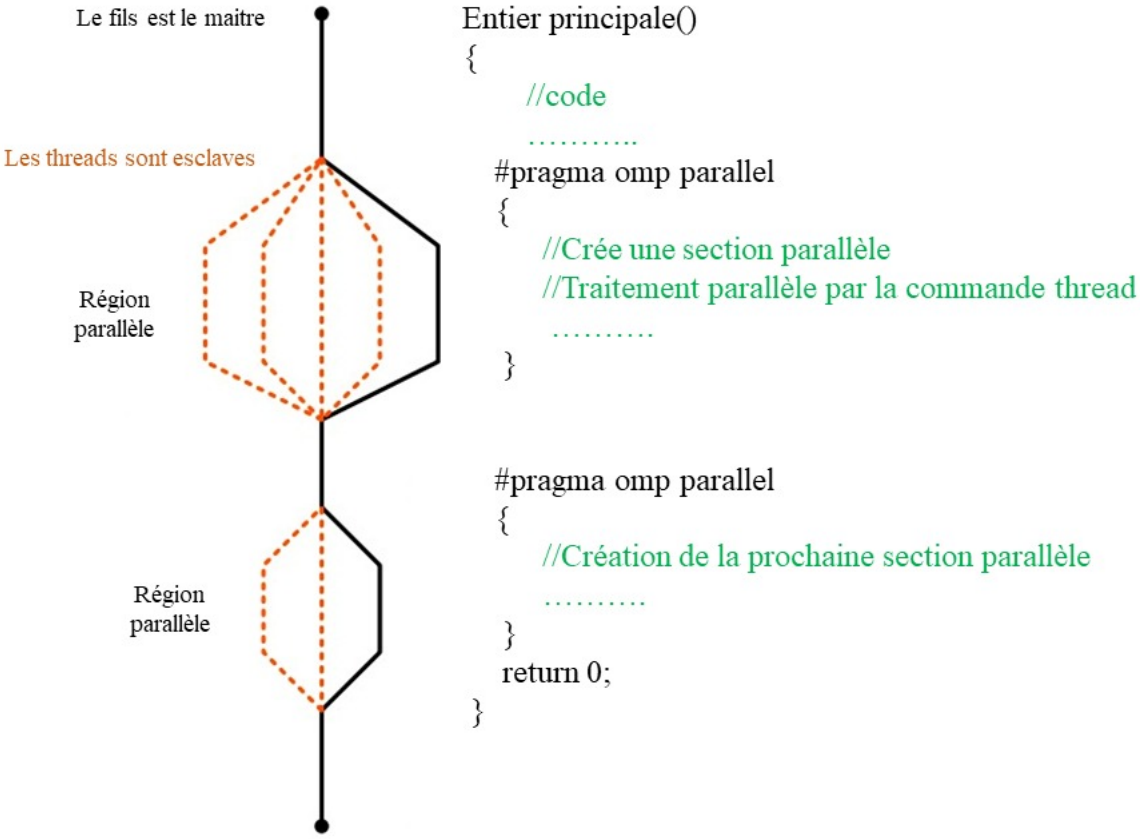

FIGURE 1.1 – Modèle fork-join

La plupart des fonctionnalités de la bibliothèque sont réalisées via des directives au compilateur.

Format directif :

```
# pragma omp commande [option [option] ... ]
{
      // bloc
      . . .
}
```

La directive s'applique uniquement à l'instruction ou au bloc suivant. Une directive peut avoir plusieurs options. Les directives sont sensibles au forme de lettre. Ils doivent être écrits en lettres minuscules.

La commande parallèle est la principale directive OpenMP. L'exécution parallèle du programme commence par lui. Le compilateur remplace la directive par un code spécial pour créer un groupe de threads. Le thread qui crée d'autres threads est appelé "maître (Master)", les threads créés sont appelés "esclaves (slave)". Chaque thread (y compris le thread maître

ainsi que les threads esclaves) exécute un bloc de code en parallèle. Les threads sont numérotés de 0 à N, où 0 est le thread maître.

Format directif :

```
# pragma omp parallèle [option[ [, ] option] ...]
{
      // bloc
          . . .
}
```

Les threads sont synchronisés en fin de bloc (synchronisation implicite de la barrière). Le programme ne poursuivra pas son exécution tant que tous les threads du bloc n'auront pas terminé leur travail. Toutes les variables sont partagées entre les threads.

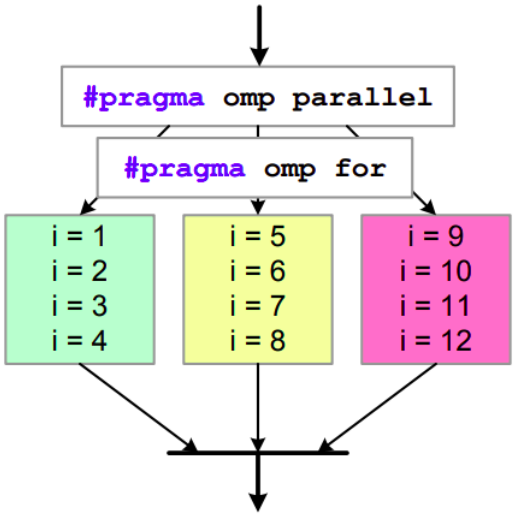

FIGURE 1.2 – Synchronisation des threads

Exemple :

```
#pragma omp parallel for
for (i = 0; i < 12; i++)
   c[i] = a[i] + b[i];
```

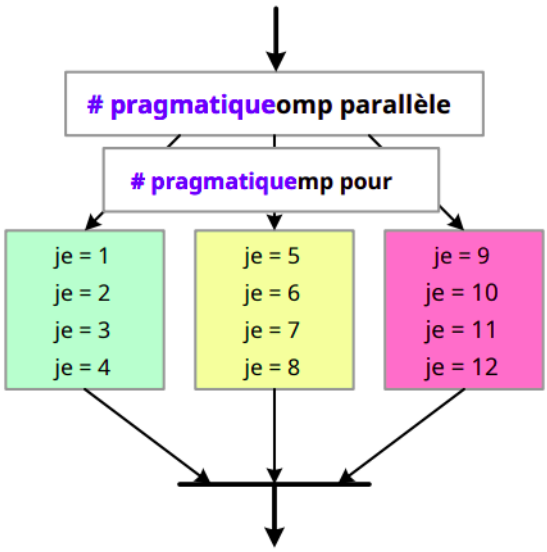

FIGURE 1.3 – Exemple de synchronisation des threads

1.1.2 Mode d'acces aux variables

Il est possible d'influencer la visibilité des variables à l'aide d'options privé et partagé :

- *Mode privé* - toutes les variables de la liste deviennent privées dans la région parallèle.
- *Mode partagé* - toutes les variables de la liste deviennent communes au thread.

1.1.3 Scheduling

OpenMP a la capacité de contrôler la planification des itérations entre les threads.

La première option est **statique**. Une distribution uniforme statique des itérations est effectuée. chaque thread décide indépendamment quel morceau de la boucle il traitera.

La deuxiéme option est **dynamique**, dans ce cas toutes les itérations sont divisées en blocs, mais l'affectation aux threads ne se produit pas. Au cours de son exécution, le thread demande la prochaine "portion" de travail.

Les pragmas que nous avons utilisé pour notre programme :

```
#pragma omp parallel for
#pragma omp for private(n)
#pragma omp for schedule(dynamic)
```

BIBLIOGRAPHIE

[1] "Guide into OpenMP", https://bisqwit.iki.fi/story/howto/openmp/

[2] "Optimisation et parallélisme avec OpenMP", https://www.martinjucker.com/documents/FlashInformatique_32007.pdf