TP HPC/Big Data

OpenMP

Supaéro
Module FITR35

Rédigé par
Gaëlle Usseglio / Annaïg Pedrono

2024/2025

Préparation du TP

Le code source se situe sur ce dépôt.
Vous pouvez le rapatrier en local via la commande *git clone*.

Vous trouverez une partie dédiée à OpenMP et une autre à MPI. Par défaut, la branche *main* contient le code à compléter et la branche *solution*, les solutions des excercices.
Le TP peut s'éxcuter sur le cluster TREX via l'URL https ://jupyterhub.cnes.fr. Plus d'explication dans la notice du JupyterHub CNES

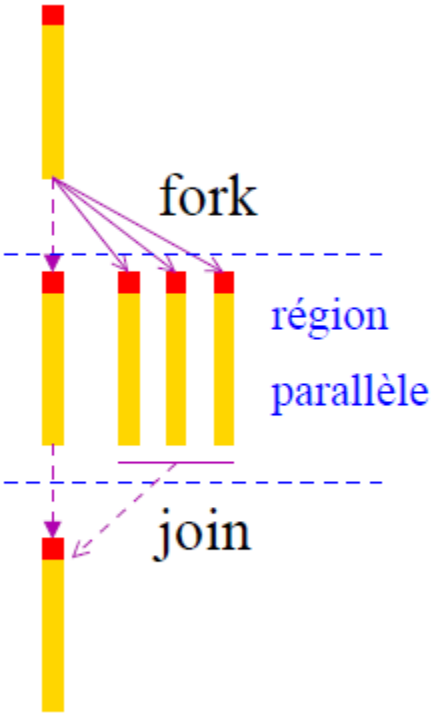
OpenMP

1 Rappel des concepts

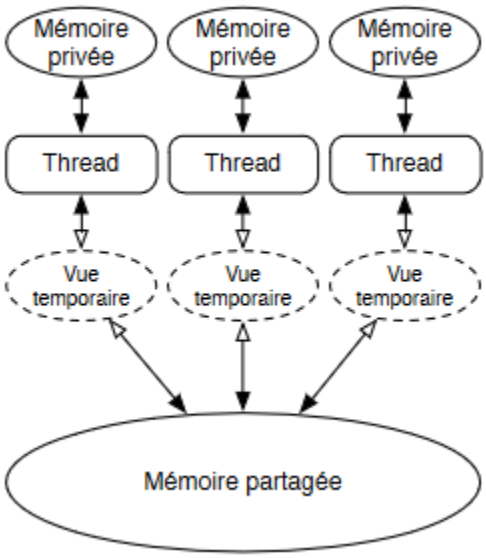
OpenMP manipule les concepts suivants :

- Thread : Entités d'exécution partageant un adressage (heap) et avec une mémoire locale (stack).
- Team : Un ensemble de un ou plusieurs threads qui participent à l'exécution d'une région parallèle.
- Task/Tâche : Une instance de code exécutable et ses données associées. Elles sont générées par les constructions PARALLEL ou TASK.
- Variable partagée : Une variable dont le nom permet d'accéder au même bloc de stockage au sein d'une région parallèle entre tâches.
- Variable privée : Une variable dont le nom permet d'accéder à différents blocs de stockage suivant les tâches, au sein d'une région parallèlle.

OpenMP est une alternance de régions séquentielles et parallèles : « fork and join » :



Une communication entre les threads implicite est faite grâce à la Mémoire partagée. :



2 Compilation - Run Time

Avant tout chose, il est important de vérifier son environnement et rendre la compilation "prête" pour OpenMP.

- Première condition, avoir un compilateur ! Regarder si le compulation GNU est présent.
- Seconde condition, activer OpenMP à la compilation. Le fichier *compil.sh* à la racine du répertoire openmp permet cette activation. Je vous laisse le regarder ....

Les principales APIs sont décrites dans le lien suivant **OpenMP cheat sheet**

3 Exemples

Deux exemples sont disponibles et permettent de mieux comprendre le fonctionnement des régions //, des scopes de variable et la notion de tâche :

- *just_parallel* :
  - La région parallèle permet-elle de différencier les actions réalisées dans les threads ?
  - Par défaut, quel est le scope des variables ? Qu'est ce que cela signifie ?
  - Quelles sont les différences entre les scopes ? Private vs shared ....
- *task* :
  - Quelle est la différence entre tâche et thread ?
  - Y-a-t-il besoin d'une synchronisation ?

4 Exercices

Maintenant, la question importante : Comment bénéficier de la parallélisation pour gagner en performance ? Essayer de paralléliser sur plusieurs threads les codes suivants :

- Estimation de **pi** via une intégrale :

$$pi = \int_0^1 f(x)\,\mathrm{d}x$$

avec

$$f(x) = \frac{4}{1+x^2}$$

  - Exemple de reduction
  - Exemple de boucle for avec protection en écriture de la variable d'accumulation
- Calcul **matriciel** : calculs arbitraires sur plusieurs tableaux avec une forte sollicitation mémoire :
  - Manipulation du heap
  - Utilisation de la stack