# Modèles et techniques en programmation parallèle hybride et multi-cœurs - TP1

Pierrick Guichard

Novembre 2020

## 1 Introduction

On se propose dans ce projet de résoudre le problème de la chaleur en 3 dimensions :

$$\begin{aligned} &\textit{Trouver } u : (x,t) \mapsto u(x,t) \textit{ avec } x \in \mathbb{R}^3 \textit{ et } t \geq 0 \textit{ tel que} \\ &\qquad \frac{\partial u}{\partial t} = \Delta u + f(x,t) \qquad \textit{dans } \Omega \\ &\qquad u(x,0) = u_0(x) \qquad \textit{dans } \Omega \\ &\qquad u(x,t) = g(x) \qquad \textit{sur } \partial\Omega,\ t \geq 0 \end{aligned}$$

Un code parallélisé avec MPI est fourni (parallélisme multi-noeuds) et nous nous proposons de réaliser à la fois un parallélisme multi-noeuds et multi-coeurs, ce qui sera plus adapté aux structures de calculs tels que les clusters (comme gin à l'ENSTA). Pour cela on choisit une stratégie hybride MPI/OpenMP.

- Le parallélisme MPI est déjà réalisé avec une décomposition 1D du domaine (selon l'axe des $x$).
- Le parallélisme OpenMP sera réalisé avec deux méthodes différentes : la méthode dite de "grain fin" (parallélisation des boucles de l'algorithme itérant sur chaque sous-domaine du code MPI, sans modifications importantes de l'algorithme MPI) et de "grain grossier" (découpage de chaque sous-domaine de calcul MPI en plusieurs sous-sous-domaines de calculs pour OpenMP, chaque sous-sous-domaine étant affecté à un thread OpenMP).

## 2 Parallélisme de type "grain fin"

Pour réaliser cette stratégie de parallélisme on procède de la manière suivante.

- On repère les boucles comptant un nombre d'itérations suffisantes. Lorsque plusieurs boucles sont imbriquées, on privilégie la parallélisation de la boucle la plus externe. Dans ce TP, contrairement au TP1, on choisit de ne pas paralléliser les boucles de l'initialisation en raison du faible gain de temps que cela procure proportionnellement au gain induit par la parallélisation de la boucle en temps.
- On repère les variables privées de ces boucles.
- On parallélise en ajoutant des pragmas aux bons endroits.

Cette méthode a pour avantage qu'elle est facile à mettre en oeuvre. Il suffit de bien prêter attention aux variables qui doivent rester privées et aux éventuelles réductions. Dans ce cas précis, on a utilisé une réduction dans le fichier *scheme.cxx* pour s'assurer que la variable qui permet de stocker la somme est incrémentée par un thread unique à la fois. Il a également fallut entourer la réduction MPI d'une région OpenMP "master" de manière à ce qu'elle soit effectuée une unique fois.

Néanmoins, cette méthode a pour désavantage qu'elle multiplie les régions parallèles ce qui diminue le gain de temps en exécutant sur plusieurs threads. En effet, le temps d'activation/ désactivation des threads n'est pas négligeable donc il est important de réduire autant que possible le nombre de régions parallèles et d'augmenter la taille de chaque région parallèle.

# 3 Parallélisme de type "grain grossier"

Pour mettre en oeuvre cette méthode il suffit de subdiviser le sous-domaine de calcul MPI en sous-sous-domaines de calcul OpenMP que l'on répartie entre les différents threads disponibles. On procède de la manière suivante :

- Création d'une région parallèle autour de la boucle en temps. Ce sera la seule région parallèle du code.
- Dans la boucle temporelle, la mise à jour de la solution ($C.iteration()$) se parallélise. Les autres instructions ne s'éxécutente pas en parallèle et nécessitent d'être prises en charge par un seul thread ($\#pragma\ omp\ single$).
- Il faut ensuite veiller à répartir correctement les points du sous domaine en fonction du numéro du thread courant obtenu avec l'instruction $omp\_get\_thread\_num()$.
- Il faut enfin mettre en oeuvre des barrières et des régions critiques de manière à ce que d'une part, tous les threads mettent à jour la solution nouvelle à partir de l'ancienne solution (et non de la nouvelle) et que d'autre part la variation soit calculée de manière correcte.

**Remarque.** *Pour les deux méthodes on a veillé à ce que les fonctions MPI appelées depuis une région parallèle OpenMP le soit depuis un environnement "pragma omp master". Ainsi il est possible d'initialiser l'environnement MPI avec l'option "FUNNELED" qui est moins restrictive que l'option "SERIALIZED". En contrepartie il faut veiller à ne pas utiliser de fonction MPI dans les environnements "single" de OpenMP et de mettre des barrières à la fin des environnements "master".*

# 4 Conclusion

Ce projet nous a permis de comparer 2 stratégies de parallélisation multi-noeuds et multi-coeurs. La méthode dite "de grain fin" est plus facile à implémenter que la méthode "de grain grossier", qui lui est cependant préférable d'un point de vue du temps de calcul.