Projet Calcul Parallèle 2024/2025

Rapport de Projet

Auteur : [**Mohamad SAMMAN**]
Encadrant : [**S.ABIDE**]

March 14, 2025

1 Introduction

Ce projet vise à explorer le calcul parallèle à travers l'installation et l'utilisation des bibliothèques **HDF5** et **Decomp2d**. Après une première validation d'un code séquentiel utilisant la méthode des différences finies et le gradient conjugué, nous avons parallélisé les calculs avec **Decomp2d**.

2 Mise en place de l'environnement

L'installation des bibliothèques s'est faite en deux étapes :

- **HDF5** : Permet d'écrire des données en parallèle.
- **Decomp2d** : Utilisée pour distribuer des tableaux 3D sur une grille de processeurs 2D.

3 Validation du code séquentiel

Avant d'aborder le calcul parallèle, nous avons validé notre méthode sur un cas test basé sur une **solution fabriquée** (Méthode de la Solution Manufacturée - MMS) :

- Utilisation des **différences finies centrées** pour approximer les dérivées secondes. (ordre 2 grace à la symétrisation)
- Résolution du système par la **méthode du gradient conjugué**.

Les résultats sont illustrés par les graphes de la Figure (1). La solution FD est plus lisse. La différence de l'échelle pour les deux graph provient du fait que nous avons utilisé le nombre de point (nx et ny) pour la méthode CG et les coordonées (x et y) pour FD. Le domaine étant identique pour les deux méthodes: [0,1]x[0,1]

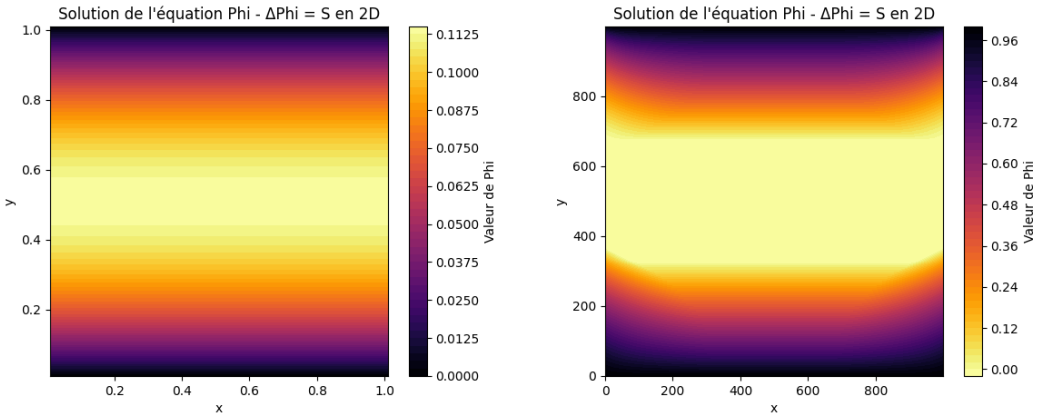

Figure 1: Résultats obtenus avec les différences finies (gauche) et le gradient conjugué (droite).

4 Concepts de Strong Scaling et Weak Scaling

Nous expliquons ces notions théoriquement et ensuite regardons les courbes de scalling.

4.1 Strong Scaling

Le **strong scaling** mesure l'accélération obtenue en augmentant le nombre de cœurs tout en conservant une taille de problème fixe :

$$S_s = \frac{T_1}{T_p} \tag{1}$$

où T_1 est le temps d'exécution en séquentiel et T_p en parallèle.

Allure attendue de la courbe :

- **Parallélisation idéale :** L'accélération est linéaire : en doublant le nombre de cœurs, le temps d'exécution est divisé par 2.
- **Parallélisation sous-optimale :** La courbe commence par suivre la tendance linéaire mais atteint une saturation due aux **communications entre processeurs** et aux **surcharges parallèles**.
- **Mauvaise parallélisation :** Peu ou pas d'amélioration, voire un ralentissement si la surcharge de communication dépasse le gain du calcul parallèle.

Nous pouvons observer une allure typique du strong scalling Figure (2) provenant de [1]

4.2 Weak Scaling

Le **weak scaling** analyse l'évolution du temps d'exécution lorsqu'on augmente la taille du problème proportionnellement au nombre de cœurs :

$$S_w = \frac{T_p(N)}{T_p(N/P)} \tag{2}$$

où N est la taille du problème et P le nombre de cœurs.

Allure attendue de la courbe :

- **Parallélisation efficace :** Le temps d'exécution reste constant, car la charge de travail par processeur reste la même.
- **Parallélisation inefficace :** Le temps d'exécution augmente progressivement à cause des **latences de communication** et de la **gestion de la mémoire partagée**.

Nous pouvons observer une allure typique du weak scalling Figure (3) provenant de [1]

5 Conclusion et perspectives

Nous avons réussi à valider l'algorithme séquentiel et à coder la paraléllisation avec la librarie **Decomp2d**. Pour la suite, il serait pertinent de :

- Étudier les communications entre processus pour optimiser le code parallèle.
- Implémenter le code pour d'autres types de problèmes.

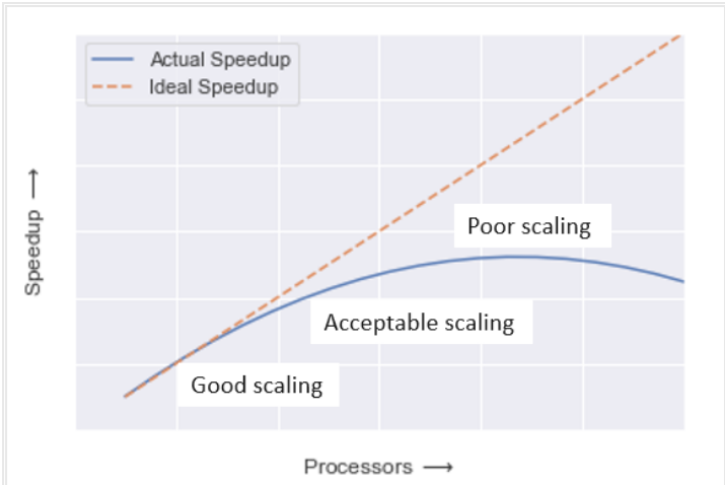

Figure 2: Allure typique d'une courbe de strong scaling.

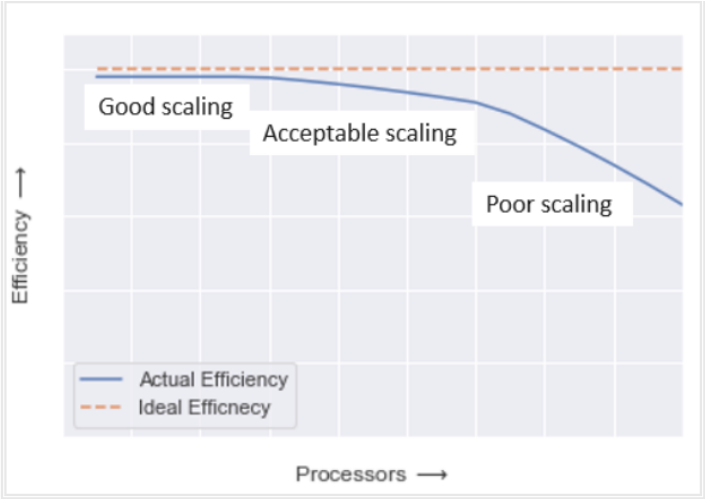

Figure 3: Allure typique d'une courbe de weak scaling.

References

[1] Scaling experiments: How to measure the performance of parallel code on HPC systems. Water Programming. https://waterprogramming.wordpress.com/2021/06/07/scaling-experiments-how-to-measure-the-performance-of-parallel-code-on-hpc-systems/