TP HPC/Big Data

MPI

Supaéro
Module FITR35

Rédigé par
Gaëlle Usseglio / Annaïg Pedrono

2025/2026

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.
Pour une utilisation sur le cluster TREX, la récupération du dépôt se fait via *git clone file :///work/C3/formation-isae/repositories/TP_OpenMP_MPI.git*. Une exception vous demandera, peut-être, d'ajouter ce répertoire comme "safe".

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
Chaque exécution sur le cluster des codes MPI devront se faire avec une soumission Slurm pour adresser des ressources choisies sur les noeuds de calcul.
Les principales commandes pour Slurm sont :

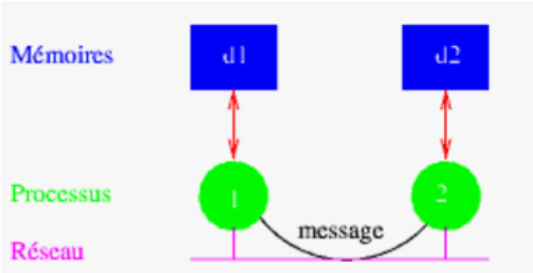
- Soumettre un job Slurm défini par un script shell : *sbatch <script sh>*.
- Voir ses jobs Slurm en cours : *squeue -u <login>*
- Supprimer un job en cours : *scancel <job id>*

MPI

1 Rappel des concepts

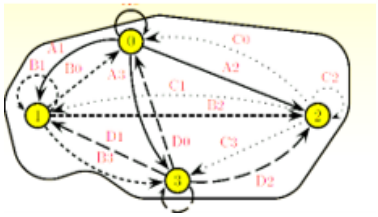
MPI manipule les concepts suivants :

- Processus : Entité totalement indépendantes et isolées les unes des autres ne partageant pas d'adressage mémoire.
- Communication : Messages échangés entre processus via le réseau de manière explicite (visible dans le code).

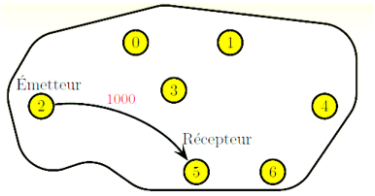


Deux types de communications sont à distinguer :

- Collectives :



- Point à Point



2 Compilation - Run Time

Pour vérifier son environnement et rendre la compilation/exécution "prête" pour MPI.

- Première condition, avoir un compilateur dédié ! Par exemple openMPI via les commandes *mpic++* et *mpirun*.
- Seconde condition, utiliser ce compilateur à la compilation et à l'exécution. Le fichier *compil.sh* à la racine du répertoire mpi/examples permet d'utiliser ce compilateur au moment de la compilation. Je vous laisse le regarder ....

Sur le cluster TREX, l'implémentation OpenMPI est directement disponible via la commande *module load openmpi*

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

3 Exemples

Les deux types de communications sont illustrés via des exemples :

- Collectives : APIs **Bcast** et **Scatter/Gather**
- Point à Point : APIs **Send/BSend** et **Recv**

4 Exercices

Maintenant à vous de jouer !

4.1 Exercice 1 : Les deadlocks

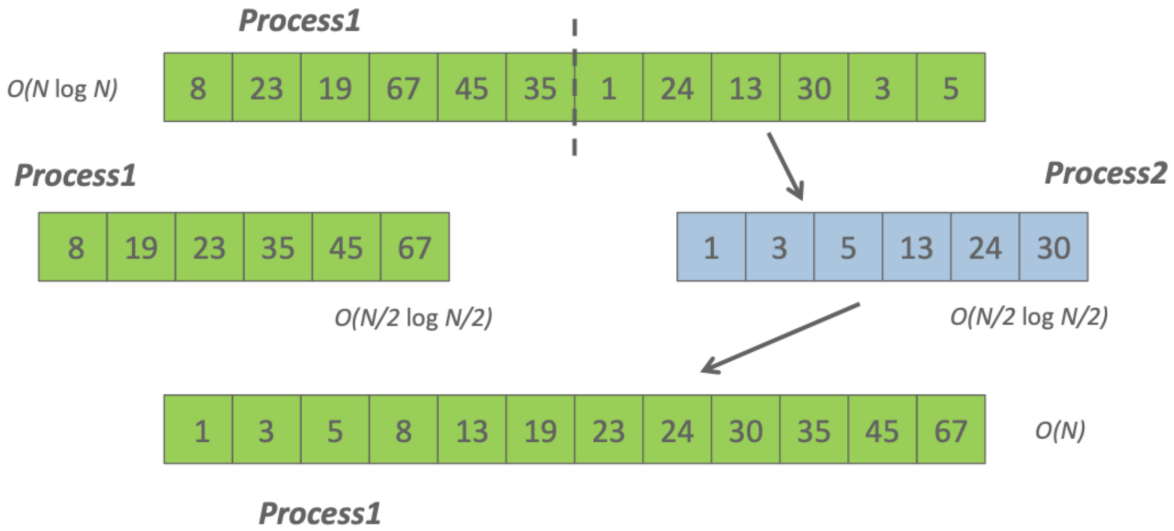
Il arrive que les communications MPI de par leurs natures bloquantes pour certains APIs, bloquent l'exécution d'un programme. Cela est par exemple le cas quand deux processus MPI veulent envoyer leur message respectif et ne sont pas prêtes à recevoir. Il est important d'identifier les cas ou cela se présente et trouver des alternatives.

Le programme *deadlock.cpp* illustre un cas de blocage.
Comment y remédier ? Il y a la possibilité d'utiliser d'autres APIs comme ISend et IRecv. Les commandes Wait et Test sont des compléments pour l'envoi de messages non bloquants.

4.2 Exercice 2 : Manipulation de tableaux

Tout comme OpenMP, un des enjeux de MPI est de découper le travail entre les processus. Ce découpage est entièrement à la charge du développeur avec la mise en place de conditions et communications.

Imaginons avoir comme traitement à trier un "gros" tableau d'entrée. Il est posible d'utiliser MPI afin de découper le tri à travers plusieurs processus.



Le répertoire *mpi/exercises/arrays* contient un programme séquentiel réalisant ce tri. Ce programme fait appel aux fonctions utilitaires de de la librairie *array_lib*, permettant de générer un tableau, de faire le tri et de rassembler un ensemble de tableau préalablement triés.

Démo : D'autres cas de calcul MPI utilisent un découpage mémoire pour estimer des équations. C'est le cas des équations de la chaleur. Un code codé en Python (avec un binding Python via mpi4py) est disponible sur le dépôt.