**École de technologie supérieure**
*Département de génie logiciel et des TI*

# Introduction au traitement parallèle avec MPI

| | |
|---|---|
| **N° du laboratoire** | 01 |
| **N° d'équipe** | 14 |
| **Étudiants** | Alexandre Richard<br>Valentine Berge |
| **Codes permanents** | RICA10028806<br>BERV02619109 |
| **Cours** | LOG645 |
| **Session** | Hiver 2015 |
| **Groupe** | 01 |
| **Professeur** | Carlos Vazquez |
| **Chargé de laboratoire** | Kevin Lachance-Coulombe |
| **Date** | Mercredi 4 février 2015 |

# Introduction

Le but de ce laboratoire est de se familiariser à la programmation parallèle. Pour accomplir cela, nous avons créé un petit logiciel qui consiste à faire une série d'altérations aux éléments d'une matrice. Ce logiciel est fait en programmation séquentiel et parallèle. Pour la version Parallèle, l'outil MPI est utilisé.
Finalement, ce document contient une analyse et une discussion des défis rencontrés et de leurs solutions.

# Analyse des problèmes

***Pour une exécution du programme donnée (prog 1 5 8), combien de fois la fonction de calcul est-elle exécutée? Donner la moyenne par processeurs?***

Pour la version séquentielle, le processeur doit exécuter la fonction de calcul 8x64 fois (512) ou le nombre d'itérations x la dimension de la matrice.
Pour la version parallèle, nous avons 8 processeurs (un processeur par ligne de la matrice) qui vont exécuter la fonction de calcul 8 fois chacun (nombre de colonnes dans la matrice) x le nombre d'itérations. Donc, selon notre exemple, on obtient une moyenne de 64 appels de la fonction de calcul par processeur.

***Quels sont les vecteurs de dépendance des cellules non limitrophes?***

Pour le problème 1, les cellules n'ont aucune dépendance les unes entre les autres.
Pour le problème 2, chaque cellule est dépendante de la cellule de la même ligne de la colonne précédente, sauf pour les cellules de la première colonne. Autrement dit, Mi,j dépend de Mi,j-1.

***Quels sont les impacts de ces dépendances sur la communication et comment cela affecte-t-il votre agglomération? Utiliser un schéma si nécessaire.***

Pour le problème 1, puisqu'il n'y a aucune dépendance entre les cellules, on pourrait, avec 64 processeurs, réaliser le calcul de chaque cellule indépendamment.
Pour le problème 2, la parallélisation du problème est limitée, puisqu'il y a des dépendances entre les colonnes. Les cellules de la première colonne sont calculables indépendamment mais les cellules des 7 autres colonnes nécessitent les valeurs des cellules de la colonne précédente.

***Combien de messages les processeurs vont-ils échanger au total (pour chaque problème)?***

Nos 8 processeurs de calcul vont échanger au total 1 message chacun vers notre processeur de résultats. Cela s'applique pour les 2 problèmes.

***Quelle est l'efficacité de votre programme parallèle et expliquez pourquoi elle n'est pas de 100%?***

Pour chacun des problèmes, nous avons utilisé 9 processeurs au total, alors que nous avions un maximum de 24 processeurs à disposition.

Pour le problème 1, étant donné que le calcul de chaque cellule est indépendant nous aurions pu augmenter le nombre de processeurs utilisés, mais cela impliquait une séparation des calculs plus compliquée.

Pour le problème 2, nous aurions pu utiliser 15 processeurs, en mettant en place une torsion. Cela aussi aurait entraîné plus de calculs et séparations de calculs.

En comparant les temps d'exécution des programmes en séquentiel et en parallèle, on remarque que le temps est plus court en parallèle seulement à partir d'un certain nombre d'itérations puisque le temps d'initialisation de MPI prend environ 1 seconde.

# Conception de la version séquentielle

- On initialise une matrice 8x8 avec tous ces éléments avec la valeur de départ passée en paramètre.
- Si le problème 1 est choisit:
    - On boucle sur le nombre d'itérations
    - On boucle sur le nombre de lignes de la matrice
    - On boucle sur le nombre de colonnes de la matrice
    - On applique la formule de calcul : *maxtrix[i][j] = maxtrix[i][j] + (i + j)*no_iteration*
- Si le problème 2 est choisit:
    - On boucle sur le nombre d'itérations
    - On boucle sur le nombre de lignes de la matrice
    - On boucle sur le nombre de colonnes de la matrice
    - Si la colonne = 0, on calcule : *maxtrix[i][j] = maxtrix[i][j] + (i*no_iteration)*
    - Si la colonne != 0, on calcule : maxtrix[i][j] = maxtrix[i][j] + maxtrix[i][j - 1] * no_iteration

# Conception de la version parallèle

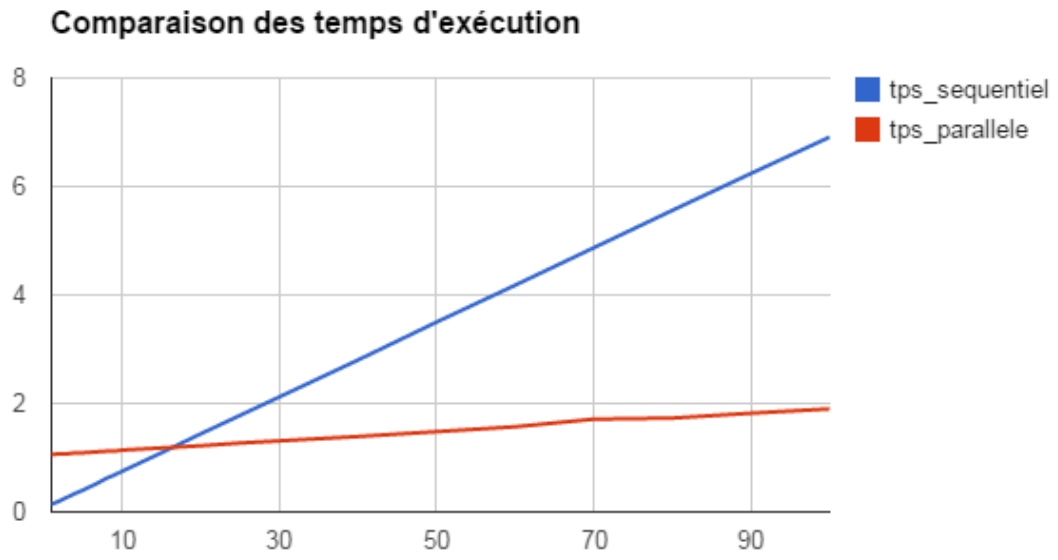
- On démarre l'application avec 9 processeurs
- On initialise MPI
- On initialise un array de 8 avec tous ces éléments avec la valeur de départ passée en paramètre
- Si le ID du processeur est plus petit que 8 (0 inclut):
    - Si le problème 1 est choisit
        - On boucle sur le nombre d'itérations
        - On boucle sur le nombre d'éléments de l'array
        - On calcule: array[i] = array[i] + (i + processus_id)*no_iteration
    - Si le problème 2 est choisit:

        - On boucle sur le nombre d'itérations
        - On boucle sur le nombre d'éléments de l'array
        - Si l'élément = 0, on calcule: *array[i] = array[i] + processus_id*no_iteration*
        - Si la colonne != 0, on calcule: array[i] = array[i] + array[i-1] * no_iteration
    - On appelle MPI_Send avec le array et le processeur ID 8 comme paramètre principal.
- Si le ID du processeur = 8
    - On appelle MPI_Recv de chaque processus (0 à 7). On extrait leur array et selon le ID de leur processus, on place l'array dans une matrice 2D dans la bonne ligne.

Les processeurs 0,1,2,3,4,5,6,7 servent à faire les calculs d'une ligne de la matrice. Leur processus ID servent a déterminer quelle ligne ils vont calculer. Par la suite, ils envoient leur array vers le processeur 8 qui va compléter la matrice finale.

## Discussion

Lors de nos tests, nous avons remarqué que le temps d'exécution du programme n'est pas toujours plus court en parallèle qu'en séquentiel. Après avoir fait une série de tests avec différents nombre d'itérations des calculs, on a pu remarquer que les calculs en parallèles sont plus efficaces qu'en séquentiel à partir de 14 itérations. Cela est dû au temps d'exécution assez élevé de l'initialisation de MPI.



# Conclusion

Nous aurions pu encore plus paralléliser nos calculs, cependant pour un nombre d'itérations assez élevé, les calculs en parallèle sont bien plus efficaces qu'en séquentiel. On peut imaginer qu'avec 1000 itérations, le calcul en séquentiel n'est même pas imaginable.