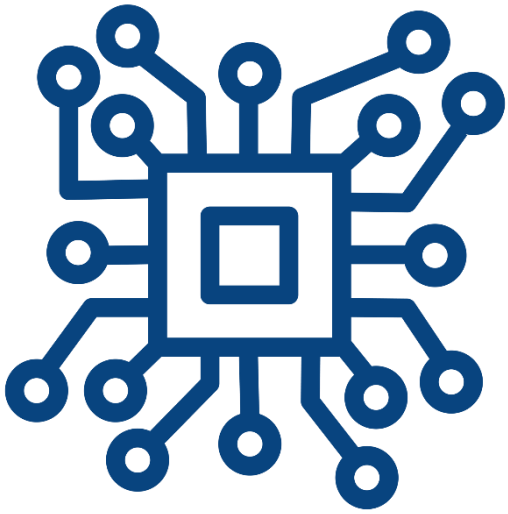
UFR de mathématique et d'informatique
Université de Strasbourg

PROGRAMMATION PARALLELE

Projet Graphe



16 AVRIL 2023

Alexandre DUBERT et Hayk ZARIKIAN

1 Les difficultés rencontrées / Idée de parallélisation

Dans un premier temps nous avons essayé d'implémenter l'algorithme de l'article donné en annexe. Nous avions réparti les k entre les différents processus MPI, puis si la condition booléenne était vraie, le processus faisait un Issend au processus à qui appartenait l'élément. À chaque itération, les processus effectuaient des Irecv pour savoir si un autre processus avait trouvé une entrée appartenant au processus actuel. Malheureusement nous étions embêtés par un segfault que nous n'arrivions pas à résoudre. Nous avons donc choisi une approche plus simple mais probablement moins performante. Les k sont répartis sur des threads OpenMP et les i sur les processus MPI. Chaque processus calcul sa version du graphe puis toutes les versions sont mises en commun sur le processus root par une réduction OU. Un des désavantages de cette version est que chaque processus doit allouer une copie temporaire de c. Par la suite, nous avons fait des tests de performances sur les machines de l'UFR et nos ordinateurs personnels afin d'évaluer la performance de la parallélisation de l'algorithme.

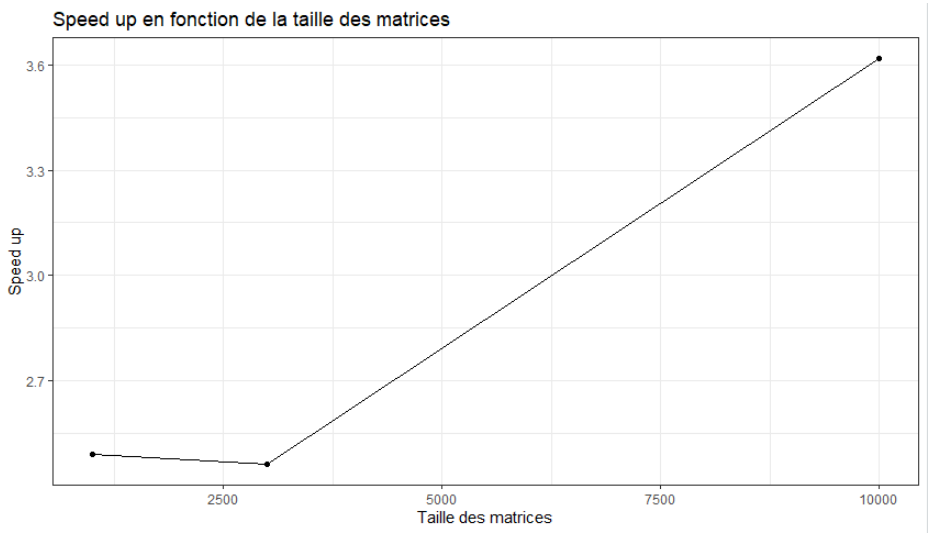
2 Mesures de performances

2.1 Full Open MP

4 Threads et 4 Processus			
Taille des matrices	Temps sequentiel	Temps parallélisé	Speed Up
1000	1.57 secondes	0.66 secondes	2.37
3000	41.63 secondes	17.77 secondes	2.34
10000	1473.9 secondes	414.99 secondes	3.55

2.2 Half Open MP, Half MPI : 4 Threads/Processus

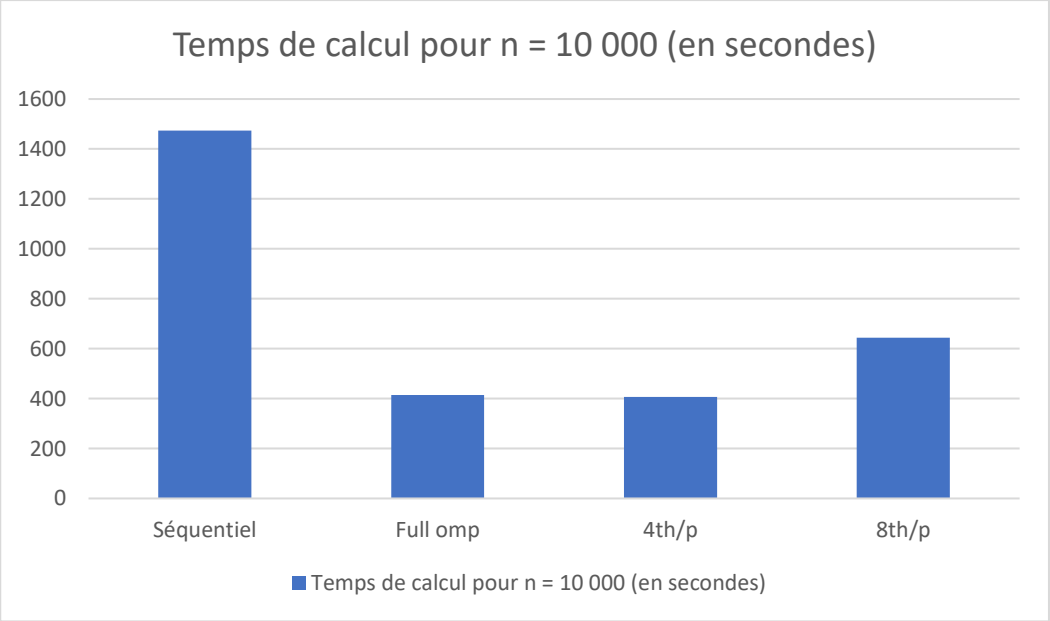
4 Threads/Processus			
Taille des matrices	Temps sequential	Temps parallélisé	Speed Up
1000	1.57 secondes	0.63 secondes	2.49
3000	41.63 secondes	16.92 secondes	2.46
10000	1473.9 secondes	406.37 secondes	3.62



2.3 Half Open MP, Half MPI : 8 Threads/Processus

8 Threads/Processus			
Taille des matrices	Temps sequential	Temps parallélisé	Speed Up
1000	1.57 secondes	0.51 secondes	3.07
3000	41.63 secondes	17.1 secondes	2.43
10000	1473.9 secondes	644.29 secondes	2.28

2.4 Comparaison matrice 10 000 en séquentiel, full omp, 4th/p et 8th/p



2.5 Analyse des mesures

On note que la courbe de l'accélération en fonction de la taille des données est croissante, en effet plus les données sont conséquentes plus le rapport bénéfice / coût de la parallélisation augmente. Par ailleurs, on remarque que les meilleures performances sont obtenues avec 16 threads répartis sur 4 processus, soient 4 threads / processus. Au-delà les performances diminuent. Nos résultats auraient été encore meilleurs si nous avions réussi à mettre en œuvre l'algorithme de Warshall parallèle.

3 Conclusion

Pour conclure, ce projet nous a permis de travailler deux points : la réflexion et l'implémentation de MPI et d'Open MP. Le point sur la réflexion est primordial car il nous a permis de comprendre comment l'implémentation allait se dérouler et d'anticiper la performance de celle-ci. Nous avons bien compris qu'il ne fallait pas négliger la préparation (réfléchir aux dépendances, comprendre l'algorithme séquentiel, chercher la meilleure section à paralléliser). Bien préparé, il était plus simple de gérer les problèmes de communications entre processus, les erreurs de syntaxe et les résultats incohérents.