Devoir #2 - MPI - Jeu de la Vie

Division par lignes Ou par sous-matrices

Le problème que j'ai choisi est le même que pour le devoir #1, à savoir le jeu de la vie.
Tout simplement car j'avais beaucoup apprécié le concept et que je souhaitais tester le même programme mais avec MPI.

Il y a donc certaines parties de code source qui sont récupérée de ma version #1 du projet, cependant une partie de ces fonctions récupérée on au final totalement ou partiellement été réécrite.

1 Description du problème

Dans le jeu de la vie, l'élément à paralléliser parait assez évident.
En effet, dans ce jeu le principe est de partir d'une grille de N x M (N > 0 & M > 0), dans laquelle les cellules peuvent se trouver seulement dans deux états. Une cellule est, soit morte, soit vivante. Le principe du jeu est alors, à partir de cette grille et d'une série de règles très simple, de calculer la grille suivante. Le jeu de la vie n'a pas de but, il s'agit uniquement de faire évoluer un automate cellulaire au cours du temps.

Les règles se base sur un calcul en fonction du nombre de voisins d'une cellule ; ainsi une cellule ayant 3 cellules vivantes à coté d'elle sera vivante à l'étape d'après, une cellule ayant moins de 2 cellules en vie ou plus de 3 en vie à coté d'elle mourra, et enfin une cellule ayant exactement 2 cellules vivantes à côté d'elle restera dans le même état. Ainsi à partir de ces règles il nous faut calculer la grille suivante.

Ainsi en suivant ces règles la grille suivante n'a pas d'interdépendance lors de son calcul (nous n'avons pas besoin de savoir quoi que ce soit de la seconde grille pour la calculer). Ainsi la parallélisation paraît assez évidente. Il suffit de demander à chaque processus de calculer une partie de la grille.

A noter : La tâche de base étant assez complexe, j'ai pris le choix de prendre UNIQUEMENT des grilles de N * N dans le cas de la division en sous-matrice. Il faut également que la matrice de N x N soit divisible en P sous matrices de taille identique, où P est le nombre de processus.
Dans le cas de la division en ligne, des matrices de N x N sont utilisées cependant il faut OBLIGATOIREMENT que la division de M par le nombre de processus P retourne un entier.

2 Les approches

Comme demandé deux (2) approches ont été fait.

La première est la division en blocs de lignes, cette division est fait avec l'aide de la fonction MPI_Scatter.
Une fois que chaque processus à reçu sa partie de grille à traiter, ils commencent alors à s'échanger entre eux les bordures dont ils vont avoir besoin pour calculer leurs morceaux de grille.
En effet, par exemple le processus 1 à besoin de la partie de matrice supérieur contenu dans le processus 0 mais également de la partie en dessus contenue dans le processus 2.
De même le processus 0 à besoin de la partie basse de la matrice contenu dans le processus 1.

Ainsi une étape d'échange est effectuée pour que chacun des processus puissent calculer leur matrice de sortie, une fois cette opération effectuée, chaque processus renvoi sa matrice de sortir au processus 0 qui se chargera de tout re-assembler puis l'afficher si besoin.

La seconde méthode est la division en sous-matrice, cette méthode m'a donnée beaucoup de mal vu la quantité de message à échanger et donc à préparer et à prévoir.
Le principe est le suivant :

(1) Le processus 0 lit la matrice d'entrée dans un fichier ou la génère.

(2) Le processus 0 envoi aux autres processus les informations dont ils auront besoin (par exemple le nombre d'itération, la taille de la grille ...)

(3) Le processus 0 divise la matrice puis envoi chaque partie aux autres processus. La fonction permettant l'envoi retourne également la matrice que le processus 0 devra traiter.

(4) Les processus s'échangent les bordures dont ils vont avoir besoin pour le calcul.
Il faut donc envoyer les côtés aux matrices gauches et droites (si elles existent) puis les parties hautes et basses aux matrices au dessus et en dessous (si elles existent), et enfin les coins aux matrices en diagonale. (encore une fois si elles existent).

(5) Une fois tout ces échange fait, chaque processus calcul sa matrice de sortie.

(6) Enfin, chacun des processus renvoi sa matrices modifiée au processus 0.

(7) Si il reste des itérations à faire ont recommence à l'étape 4.

On notera que pour éviter de copier le tp sur la multiplication de matrice je me suis demandé si il était possible d'effectuer le même type d'opération mais sans "splitter" le canal de communication, j'ai donc pris le choix de ne pas utiliser cette fonctionnalité pour tester une nouvelle approche.
En fin de compte, le fait de ne pas avoir "splitter" les canaux de communication fini par créer une trop grande quantité de message qui a dû, je suppose, ralentir l'exécution du programme.

3 Les tests

Pour lancer les tests vous pouvez utiliser la commande `make tests`. D'un autre côté, il est possible de lancer le script avec la commande `./Script/test.sh X Y` ou X et Y présentent les bornes de début et de fin pour la taille de la grille.
Par exemple, si X = 196 et Y = 200, seules les grilles ayant une largeur comprise entre 196 et 200 seront traitées.

Chose particulière de ce script est le fait que, de base, les tests sont lancés avec un nombre de processus égal à la taille de la largeur de la grille.
Il est ensuite possible de personnaliser le nombre de processus à tester à l'aide de la ligne 10 du fichier ./Script/test.sh. Le tableau NB_PROC_SPEC permet alors, pour une largeur de grille donnée, de définir le nombre de processus à tester.
Par exemple, [9]="1,9", signifie que si la largeur de grille est de neuf (9), alors il faudra lancer le programme une fois avec un (1) processus, puis une fois avec neuf (9).
Cette méthode me permet alors de lancer une infinité de tailles de grille tout en ayant une personnalisation possible sur les tests à faire afin de montrer que le programme fonctionne bien avec un nombre de processus différent de la largeur de la grille.

Le test se déroule de la façon suivante :
- La grille de tests est générée
- Le test séquentiel est lancé, il sert de comparaison pour vérifier la sortie des deux (2) autres versions
- Les itérations des deux (2) versions du programme sont lancées, une première fois avec une division sous forme de blocs de ligne, puis une seconde fois avec la division en sous matrice.
- Les deux (2) sorties sont alors comparées à la version obtenue par le programme séquentiel à la suite des Y itérations, si une différence est trouvée celle-ci est montrée et une erreur est affichée.

Les tests s'effectuent avec peu de processus (minimum un (1)) mais également avec beaucoup de processus, comme par exemple le tests sur la grille de 255 x 255 ou 255 processus sont utilisés.

Il est à noter que, sans modifier la limite du nombre de "pipes" maximum autorisé, à partir d'une certaine taille le message suivant est affiché :

```
ORTE_ERROR_LOG: The system limit on number of pipes a process can open
was reached in file odls_default_module.c at line 895
```

Pour l'éviter il faut donc augmenter la limite, ou bien préciser des nombres de processus inférieur à ceux donné.

A noter : A la fin des programmes de tests une chaine de caractère est affichée sous la forme : ".#...." par exemple, ou "." signifie qu'un test est passé avec succès, "#" signifiant que le tests à échoué. Ainsi ici le tests deux (2) aurait échoué.

4 Résultats expérimentaux

Les résultats expérimentaux pour les valeurs suivantes : 64, 192, 320, 512, 1088.
Pour avoir les valeurs exactes, merci de vous reporter au fichier ./Time/size_'SIZE'.dat, ou 'SIZE' est la taille

souhaitée.

Les valeurs utilisées pour les mesures de temps sont différentes de celles des tests.
Attention : Les échelles ne sont pas les mêmes dans tous les exemples.

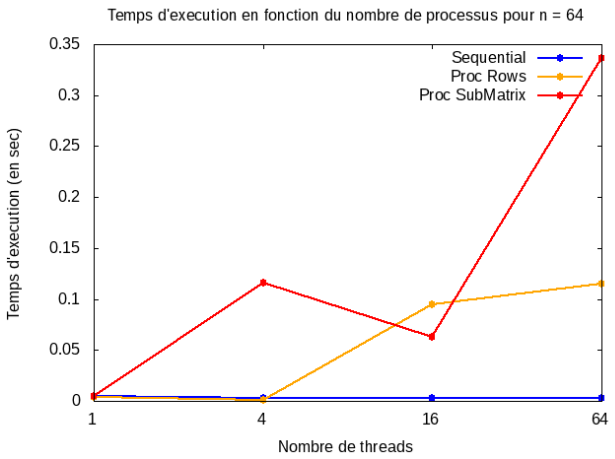

FIGURE 1 – Temps d'exécution pour une grille de 64 x 64

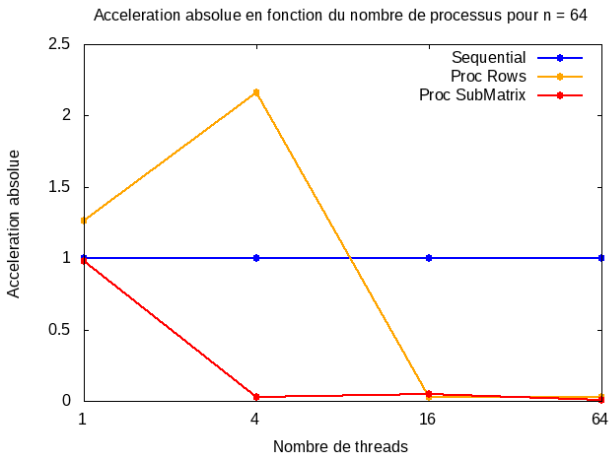

FIGURE 2 – Accélération absolue pour une grille de 64 x 64

Première chose que l'on peut remarquer aisément et le fait que la méthode de division par matrice n'est pas viable dans un si petit example, un effet, la quantité de message à échanger étant trop important par rapport à la taille du problème cause un grand temps de latence.
Nous pouvons remarquer également que l'accélération de la division par ligne avec uniquement quatre (4) blocs de lignes semble être une bonne idée pour un petit exemple comme celui-ci, en effet l'algorithme de MPI_Scatter, doit être relativement bien optimisé pour permettre une si bonne accélération.

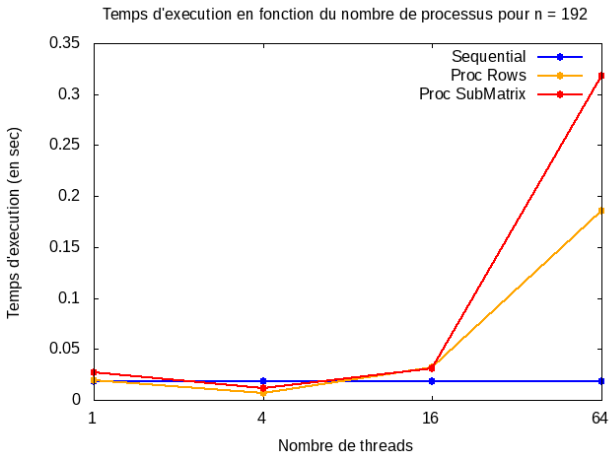

FIGURE 3 – Temps d'exécution pour une grille de 192 x 192

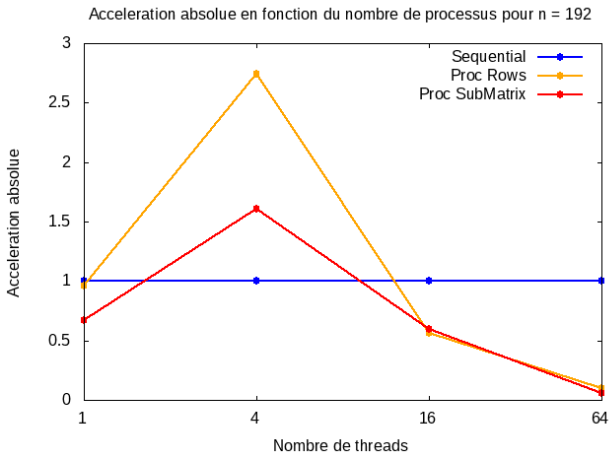

FIGURE 4 – Accélération absolue pour une grille de 192 x 192

Dans cet exemple, on peut remarquer qu'un trop grand nombre de processus par rapport à la taille du problème représente une grande perte de temps, on peut voir ceci avec le test avec 64 processus.
Dans les 2 méthodes, les temps deviennent relativement grands avec ce trop grand nombre de processus qui cause donc un grand nombre d'échanges de messages inutilement.

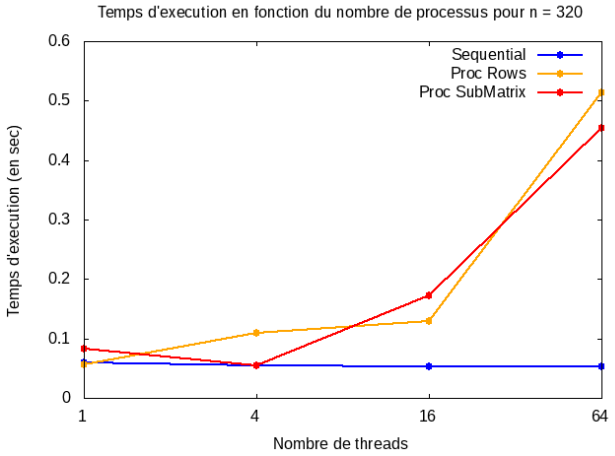

FIGURE 5 – Temps d'exécution pour une grille de 320 x 320

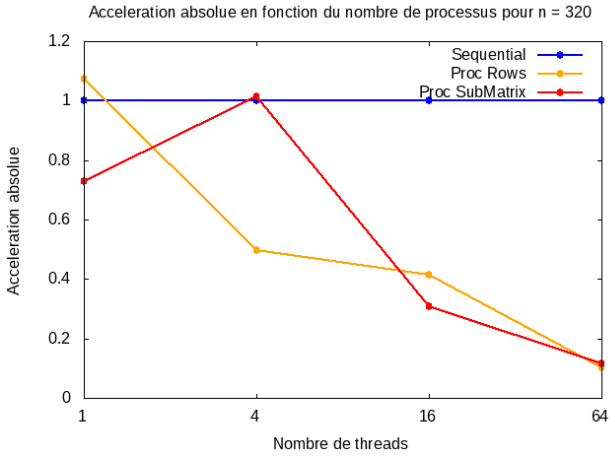

FIGURE 6 – Accélération absolue pour une grille de 320 x 320

Cet exemple est assez intéressant, notamment car l'on peu remarquer que avec cette grandeur de problème la division en sous-matrice arrive à rivaliser avec la version séquentielle.
En effet, avec 4 processus chaque sous-matrice à une taille de 80 x 80, avec que peu d'échange de messages, seuls 3 échanges sont nécessaires pour calculer chacune des quatre (4) matrice. Ainsi dans ce cas précis, la division en sous-matrice semble adaptée, bien que le gain ne soit pas énorme.

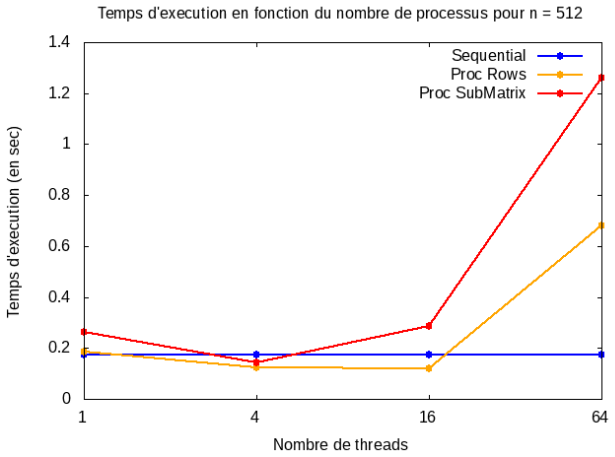

FIGURE 7 – Temps d'exécution pour une grille de 512 x 512

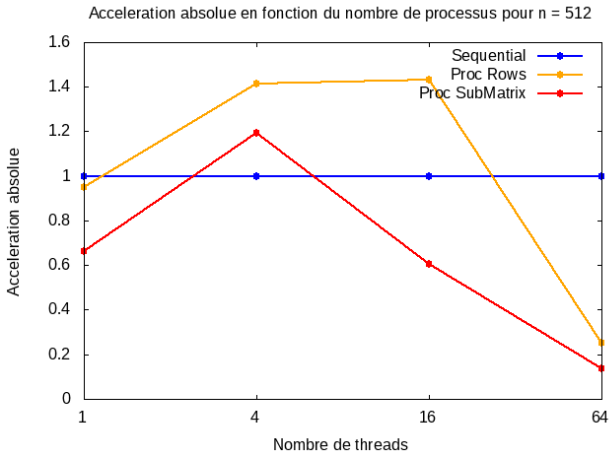

FIGURE 8 – Accélération absolue pour une grille de 512 x 512

Encore une fois la méthode en utilisant des sous-matrices donne des résultats relativement bien avec 4 processus, cependant la division par ligne nous donne des résultats encore mieux. Surement grâce aux grandes optimisations de la fonction MPI_Scatter et MPI_Gather.
Qui plus est, on peut remarquer que la division avec 16 processus est encore meilleure qu'une division en ligne ou en sous-matrice avec 4 processus. Ainsi pour des problèmes de cet ordre de taille une division par blocs de ligne est plus adaptée.

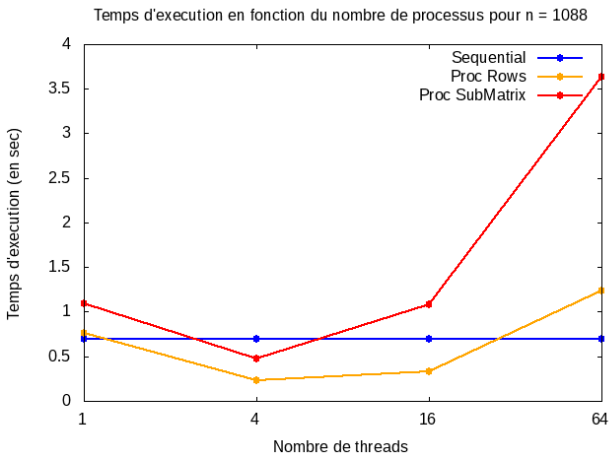

FIGURE 9 – Temps d'exécution pour une grille de 1088 x 1088

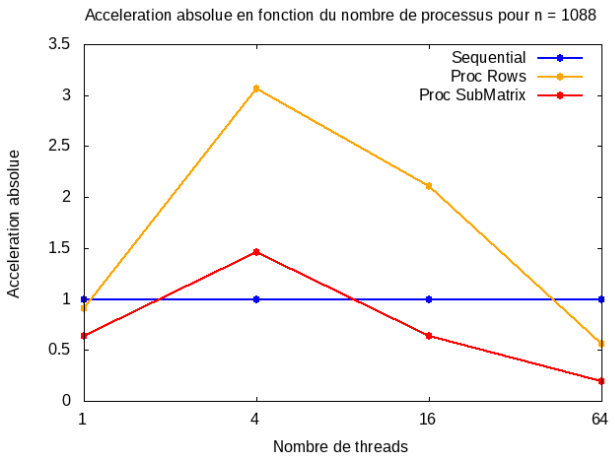

FIGURE 10 – Accélération absolue pour une grille de 1088 x 1088

Enfin, sur ce dernier exemple, on remarque encore une fois que l'utilisation d'un trop grand nombre de processus n'aide pas à résoudre le problème plus rapidement, des valeurs plus petites telles que quatre (4) processus ou seize (16) sont plus adaptés à ce genre de problème.
On remarque encore une fois un écart entre la version avec des sous-matrices et une division en blocs de ligne au niveau de l'accélération absolue.

5 Conclusion

On peut aisément remarquer que la version avec une division par matrice n'est pas forcément une bonne option, cependant celle-ci arrive à rivaliser avec la division par ligne quand le nombre de processus n'est pas trop élevé. Pour rappel plus le nombre de processus est élevé plus il y aura de message à faire passer. Certes, ces messages seront de tailles inférieurs mais beaucoup plus nombreux.
Le très grand nombre de messages à échanger ralentit alors l'exécution du programme et résulte en des performances moins bonnes pour la division en sous-matrice.

Par exemple, si l'on considère la sous-matrice qui se situerait au milieu de la grille, doit, pour commencer ses calculs obtenir de ces voisins les huit (8) parties qui lui manquent, à savoir les côtés gauche et droit, le haut et le bas et enfin chacune des quatre (4) diagonales. Là ou, la division en blocs de ligne à besoin seulement de deux (2) messages.

En plus de faire moins d'échange de messages, la division par blocs de lignes utilise uniquement les méthodes MPI_Scatter et MPI_Gather, ces méthodes sont certainement énormément plus optimisées que les Send et Recv utilisé pour la division en sous-matrice.

En somme, le nombre trop grand de messages à échanger et la sous-optimisation ou la non-optimisation de mes méthodes d'échange de bordure des matrices notamment en essayant de ne pas "spliter" les canaux de communication fait que l'on se retrouve avec une trop grande quantité de messages échangé qui au final ralentissent le programme et le rende sous performant par rapport à la version avec la division en blocs lignes qui elle utilise des méthodes natives optimisée de MPI pour passer aux autres processus les données à traiter.

6 Difficultés

Ma grande difficulté a été de bien comprendre quelles parties était à envoyer à chacun des processus voisin notamment sur la division en sous-matrice qui implique une grande quantité de message. Ainsi dû à des erreurs d'envoi de ma part (envoi de la mauvaise zone par exemple) j'ai été fortement ralenti dans ma progression.

Qui plus est, n'ayant pas réussi à utiliser des programmes aidant au débogages (mon ordinateur n'étant pas assez puissant je suppose), je me suis retrouvé à faire le débogging principalement à la main ce qui a été pour moi une autre difficulté, notamment dû au critère assez aléatoire des messages.