1 Description du problème

Le problème que j'ai choisi est le jeu de la vie. Dans ce dernier, l'élément à paralléliser parait assez évidant. 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 règle 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 bases sur un calcule en fonction du nombre de voisin d'une cellule ; ainsi une cellule ayant 3 cellules vivante 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 vivante 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 calcule (nous n'avons pas besoin de savoir quoi que ce soit de la seconde grille pour la calculer). Ainsi la parallélisation parait 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 tailles identiques, ou P est le nombre de processus.
Dans le cas de la division en ligne, des matrices de N x M sont prise 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 un petit processus d'échange est effectuer pour que chaque processus puisse calculer la matrice de sortie sans soucis, une fois cette opération est fait, 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.
Le principe est le suivant :

(1) Le processus 0 lit la matrice d'entrée ou la généré.

(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. Il s'envoi également une partie à traiter.

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

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

(6) Enfin, chaque processus renvoi leurs matrices modifié au processus 0.

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

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` où 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 comprises entre 196 et 200 seront fait.

Chose particulière de se script est le fait que, de base, les tests sont lancé 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 taille 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 :
- Les itérations sont des deux (2) versions du programme sont lancée, 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 comparé à la suite des Y itérations, si une différence est trouvée celle ci est montrée.

Il est important de remarquer que j'ai une limite par rapport aux tests, j'ai pu remarqué que au delà d'une grille de plus de 400 de large (environs) la méthode de division en sous matrice ne fonctionne pas [il ne se termine pas], alors que en ligne de commande classique (hors du terminale) celui ci finis. Je soupçonne un soucis avec les "ulimits".

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.

Il est important de noter que les valeurs fournies pour un (1) processeur avec les grilles de tailles 512 & 1088 sont fausses. Elles on été mise à 0 pour éviter les soucis, cependant elles sont bien entendu non réelles. Elles sont fixées à cause du problème cité plus haut.

5 Conclusion

6 Difficultés

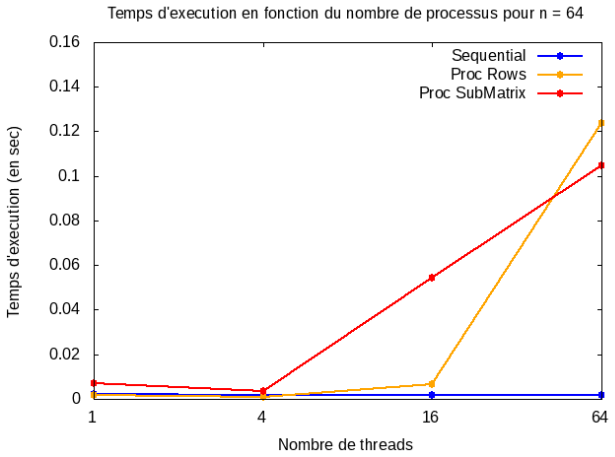

FIGURE 1 – Temps d'exécution pour une grille de 64 x 64

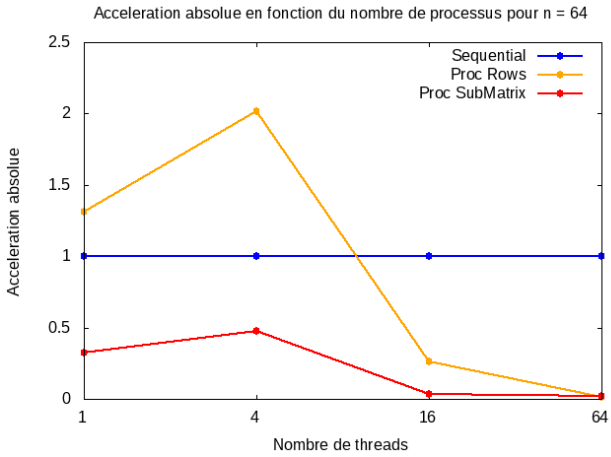

FIGURE 2 – Accélération absolue pour une grille de 64 x 64

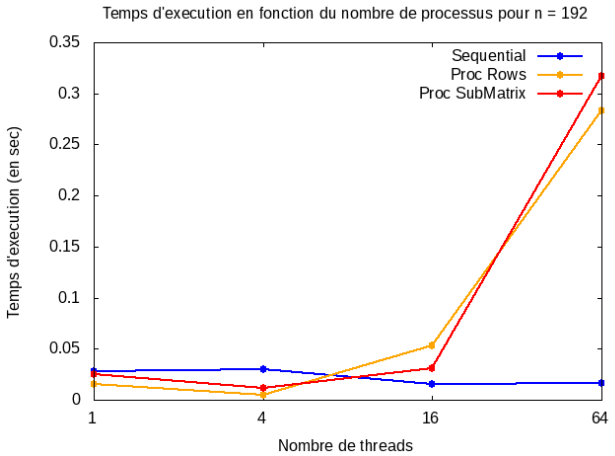

FIGURE 3 – Temps d'exécution pour une grille de 192 x 192

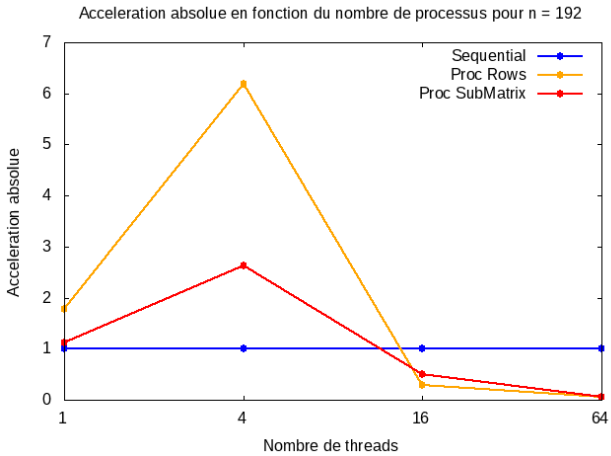

FIGURE 4 – Accélération absolue pour une grille de 192 x 192

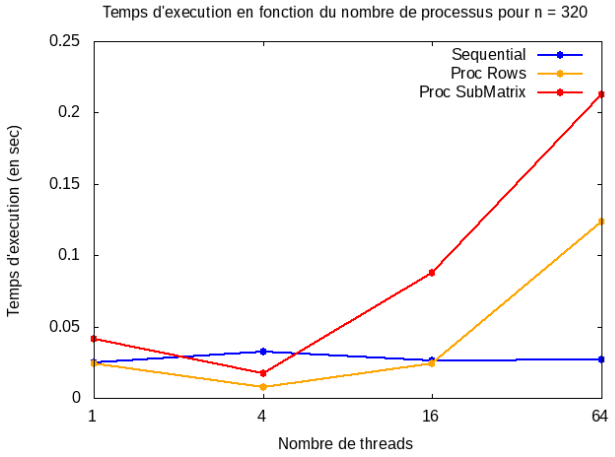

FIGURE 5 – Temps d'exécution pour une grille de 320 x 320

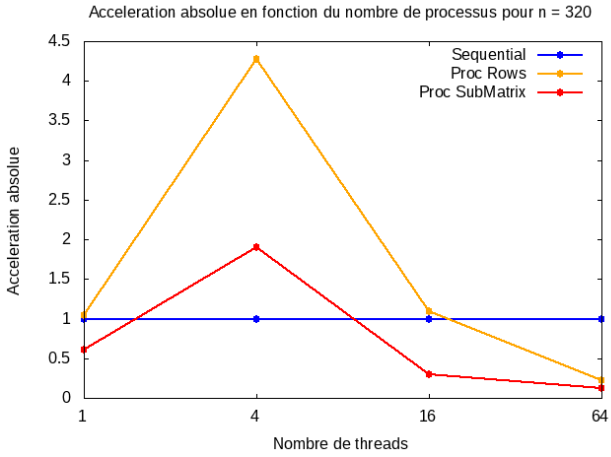

FIGURE 6 – Accélération absolue pour une grille de 320 x 320

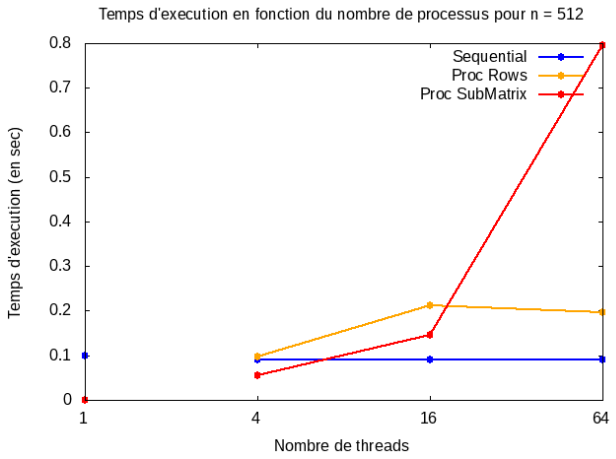

FIGURE 7 – Temps d'exécution pour une grille de 512 x 512

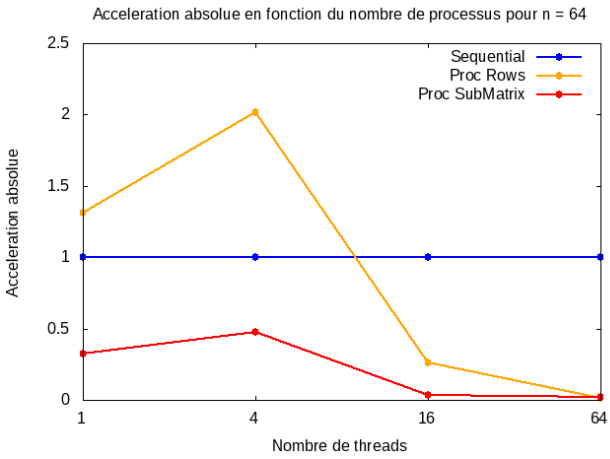

FIGURE 8 – Accélération absolue pour une grille de 64 x 64

./Time/size_1088_time.png

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

./Time/size_1088_acceleration.png

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