Projet : MPI et le jeu de la vie

Andréas Guillot

April 28, 2017

1 Notes générales :

Afin de différencier vos commentaires des miens j'ai choisi de mettre les miens en anglais.

J'ai remplacé la boucle

```
for (size_t j=1 ; j<LONGCYCLE ; j++)
```

par

```
for (size_t j=LONGCYCLE-1; j>0; j--)
```

conformément aux instructions que vous avez donné à Brandon et qu'il nous a par la suite transmises.

En revanche, nous sommes beaucoup à avoir constaté qu'il y avait un problème quand le nombre de processus est un multiple de 4. C'est pourquoi j'ai 2, 6, et 10 pour le *banchmark*.

Enfin, les programmes seront tous compilés par mon *Makefile*, et seront exécutés avec la commande

```
mpiexec -np X ./gvie_cycle_mpiY
```

, où X est le nombre de processus et Y est le numéro de la tâche.

2 Tâche 1 :

Cette tâche mets surtout en place la partie MPI du programme.

Il y a néanmoins un cas intéressant à gérer: si jamais le nombre de lignes n'est pas divisible par le nombre de processus alors il faut allouer un nombre différent de lignes au dernier processus. Cela est géré en faisant en sorte que le dernier processus fasse un peu plus de travail.

3 Question 1 :

Toute configuration du jeu de la vie finit dans une configuration cyclique étant donné que le plan est fini : un plan fini veut dire un nombre fini de configurations possibles, ce qui a pour conséquence le fait que la configuration finisse forcément par boucler.

4 Question 2 :

La longueur globale du cycle est le PPCM (ou *Least Common Multiple* en anglais et dans mon code) de la longueur de tous les cycles car le cycle global ne peut être obtenu que si tous les sous-cycles de la configuration se terminent au même moment.

Cela veut donc dire que le temps auquel le jeu de la vie atteindra son ultime configuration devra être un multiple du temps de tous les sous-cycles, i.e. le PPCM.

Tous les multiples de ce temps global seront eux aussi le dernier état du jeu de la vie, mais le PPCM sera le premier temps auquel cela se produira : dire que le PPCM n'est pas le plus petit temps reviendrait à dire qu'il existe un temps inférieur au PPCM qui soit multiple de tous les sous-cycles, ce qui est absurde vu que le PPCM est le plus petit multiple de tous les temps des sous-cycles.

5 Tâche 2 :

Cette tâche doit calculer le PPCM. Un processus qui a trouvé son cycle devrait normalement continuer à calculer son prochain état afin de pouvoir envoyer ses frontières à ses voisins, mais il n'y a pas encore de frontières ici.

Un processus peut donc simplement se terminer quand il a trouvé un cycle et qu'il a envoyé son PPCM au processus root à l'aide d'un appel bloquant à "Ssend", ce qui fait que les processus non-root vont attendre que le root soit capable de réceptionner leur ppcm.

6 Tâche 3 :

L'implémentation de l'opérateur de réduction simplifie le code étant donné que l'opération *reduce* va aussi envoyer les messages. De plus, le processus root n'a pas a stocké toutes les valeurs dans un tableau étant donné qu'il peut calculer le PPCM dès qu'il reçoit une valeur d'un autre processus (le PPCM est commutatif).

Tâche 4 :

6.1 Envoi des frontières

L'envoi des frontières se fait au tout début d'une itération. Le cas où il n'y a pas de frontières à envoyer est géré grâce au rang.

6.2 Calcul de l'essentiel du jeu

Une fois les frontières envoyées on calcule la prochaine étape sur tout le tableau sauf sur les frontières afin de ne pas attendre inutilement l'arrivée des frontières des voisins.

6.3 Calcul des frontières

Enfin, on réceptionne les frontières et on les calcule afin de tester si jamais un cycle est trouvé.

6.4 Détection de la terminaison

Je me sers d'un *MPI_Allreduce* afin de faire la somme des processus s'étant terminés, et de répercuter le résultat sur tous les processus.

Les processus s'arrêteront lorsque cette réduction leur annoncera que tous les processus se sont terminés.

Il est intéressant de noter qu'une fois qu'ils ont trouvé leur cycle alors les processus continuent de calculer leurs frontières afin de les envoyer aux voisins. En revanche, ils ne testent plus si jamais ils ont trouvé un cycle, ce qui réduit le temps d'exécution.

Tâche 5 :

L'optimisation que j'ai effectuée est de remplacer les *recv* par des *irecv* et de coupler le tout avec des *wait* afin de pouvoir se préparer à recevoir les 2 en même temps.

Tâche 6 :

Mon ordinateur contient un i5-7200U avec 2 coeurs, 4 threads et 12Go de RAM.

Les résultats sont dans les tableaux suivants. Table X correspond au programme gvie_cycle_mpiX.

On peut observer qu'à partir de 6 processus le temps d'exécution devient beaucoup plus faible. Cela est du au fait que trouver un cycle dans une plus petite partie de la matrice augmente la vitesse.

La différence entre le programme fonctionnel et optimisé n'est néanmoins pas flagrante.

Table 1: Version 1

Nombre de processus :	1	2	6	10
Moyenne	19.48	12.39	0.16	0.22
Ecart type	0.23	0.06	0.02	0.03

Table 2: Version 2

Nombre de processus :	1	2	6	10
Moyenne	18.04	22.02	0.23	0.27
Ecart type	0.32	0.16	0.01	0.04

Table 3: Version 3

Nombre de processus :	1	2	6	10
Moyenne	17.59	8.81	0.16	0.23
Ecart type	0.25	0.1	0.02	0.03

Table 4: Version 4

Nombre de processus :	1	2	6	10
Moyenne	11.77	15.53	0.55	0.56
Ecart type	3.5	4.8	0.04	0.03

Table 5: My caption

Nombre de processus :	1	2	6	10
Moyenne	11.69	14.43	0.49	0.55
Ecart type	8.2	10.1	0.01	0.008