Rémy Malgouyres
Université Clermont 1
http : //laic.u − clermont1.fr/~mr/

Programmation système et réseaux

TP n° 3
Jeu de la vie parallèle

Durée 1 semaine

Objectifs :
Le but de ce TP est d'implémenter une version parallèle jeu de la vie.

Note. Chaque semaine, le listing commenté du TP est à rendre en début de la scéance de la semaine suivante.

1 Qu'est-ce que le jeu de la vie ?

Le jeu de la vie est un jeu sur les images composées de pixels qui vallent soit 0 soit 1 (les pixels du bord de l'image valent 0).

Chaque pixel est entouré de 8 autres pixels. En comptant le pixel lui-même, le voisinage immédiat du pixel est donc formé de 9 pixels :

```
XXX
XXX
XXX
```

Jouer un coup au jeu de la vie consiste à parcourir l'image (à l'exception des pixels du bord qui restent à 0), et à appliquer une règle pour chaque pixel pour décider si la valeur du pixel va passer à 0 ou à 1. Exemples de règles :

- **Règle 5.** un pixel passe à 1 si le nombre de 1 dans son voisinage est inférieur ou égal à 5, sinon il passe à 0
- **Règle 4.** un pixel passe à 1 si le nombre de 1 dans son voisinage est inférieur ou égal à 4, sinon il passe à 0
- **Règle 3.** un pixel passe à 1 si le nombre de 1 dans son voisinage est inférieur ou égal à 3, sinon il passe à 0
- **Règle 2.** un pixel passe à 1 si le nombre de 1 dans son voisinage est inférieur ou égal à 2, sinon il passe à 0

Le jeu de la vie consiste à itérer les coups.

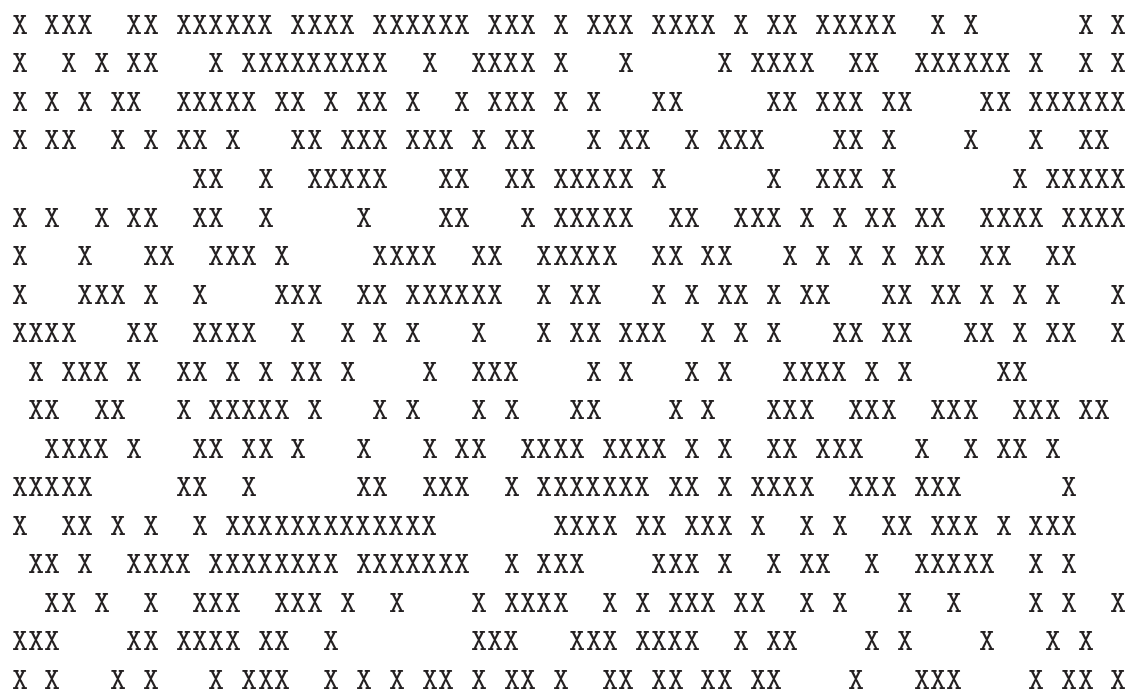

FIG. 1: Exemple d'image composée de 0 et de 1

2 Travail à réaliser

Exercice 1 Créer un programme C, prévu pour utiliser des threads et des mutex. Définir en variable globale une matrice `tableau`, avec 40 lignes et 70 colonnes. Définir des constantes NL et NC pour le nombre de lignes et le nombre de colonnes du tableau.

Exercice 2 Créer une fonction qui génère aléatoirement des 0 et des 1 dans le tableau (sauf les éléments du bord qui valent tous 0), avec une proportion d'un tiers de 1 et de deux tiers de 0.

Exercice 3 Créer une fonction d'affichage du tableau, qui efface la console, et affiche le tableau dans la console. La fonction affichera aussi la proportion de 1 dans le tableau.

Exercice 4 Écrire une fonction

```
int Regle4(int i, int j);
```

qui retourne la valeur du pixel (i, j) (ligne i, colonne j) suite à l'application de la règle 4 du jeu de la vie. Écrire de même des fonctions de même prototype pour les règles 2, 3, et 5.

Exercice 5 Écrire une fonction `JouerCoup` qui prend en paramètre un pointeur sur la fonction de règle, et qui joue un coup du jeu de la vie (parcours une fois du tableau).

Exercice 6 Écrire une fonction de thread qui prend en paramètre un pointeur sur la fonction de règle et qui joue des coups. La fonction se poursuit tant que le nombre total de coups est inférieur à une constante globale `NB_ITERATIONS`. On prendra garde que plusieurs threads chercheront à accéder au tableau, et on créera un mutex pour éviter les conflicts. Le compteur de coups sera lui aussi global et sera protégé par un autre mutex.

Entre deux coups, le thread attendra un temps alléatoire par l'instruction :

```
usleep(100000*(rand()%5+1));
```

Exercice 7 Créer une fonction de thread qui affiche de temps à autre le tableau et qui se termine quand le nombre total de coups joués atteint `NB_ITERATIONS`.

Exercice 8 Créer une fonction `main` avec une variable locale `tabRegle` qui contient un tableau de 4 fonctions de règles. La déclaration (et initialisation) est la suivante :

```
int (*tabRegle[4])(int, int) = {Fonction1,Fonction2,Fonction3,Fonction4};
```

Dans le `main`, on génèrera 5 threads (stockés dans un tableau de `pthread_t`). Le 4 premiers threads joueront des coups du jeu de la vie chacun avec sa fonction de règle, et le dernier thread fera des affichages. Le `main` attendra que tous les coups aient été joués avant de se terminer.

Exercice 9 Dans le jeu d'un coup jusqu'à maintenant, la nouvelle valeur d'un pixel dépend des **nouvelles** valeurs des pixels (voisins) qui le précèdent dans le parcours du tableau. Modifier le programme pour qu'il joue sur deux tableaux, l'ensemble d'un tableau étant calculé à partir de l'autre tableau lorsqu'on joue un coup. De cette manière, la nouvelle valeur d'un pixel ne dépend que de l'ancienne valeur des pixels voisins. On pourra modifier la structure de données pour créer une variable globale :

```
int tableau[2][40][70];
```