TPs 1, 2, 3 et 4

Master 1 SID
Benoist GASTON
benoist.gaston@univ-rouen.fr

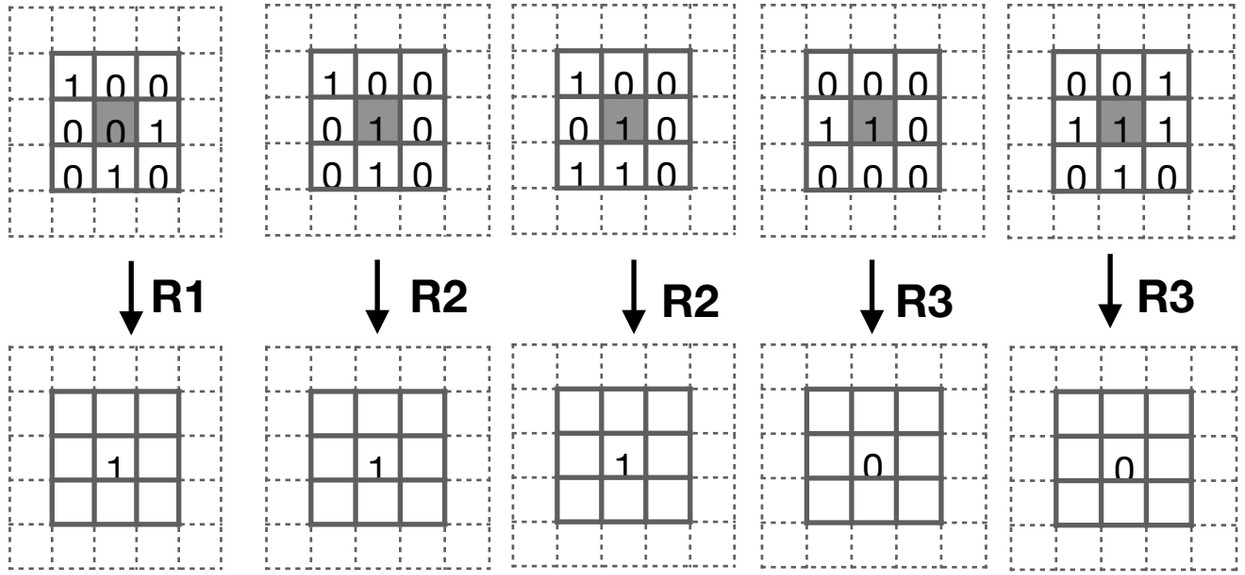
Jeu de la Vie

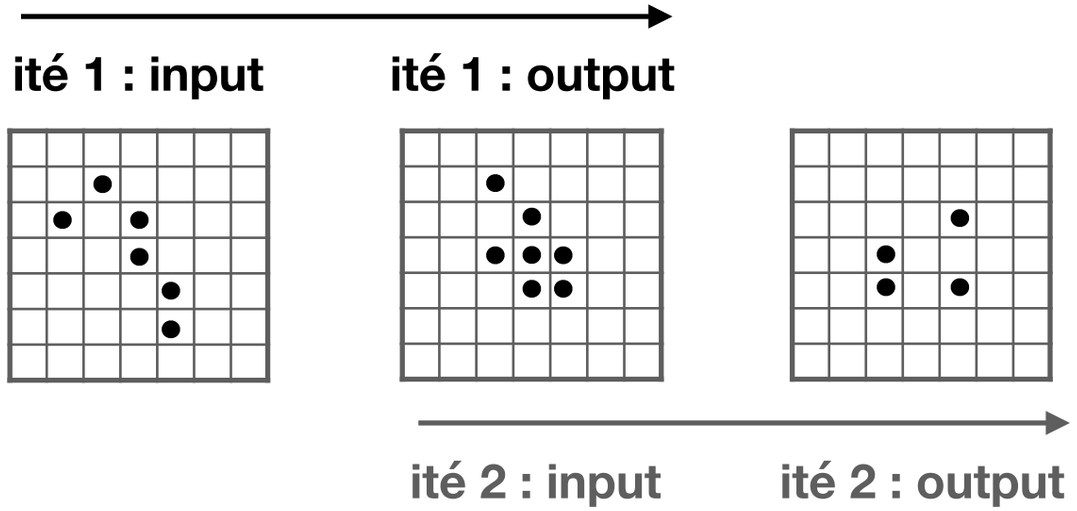
Présentation

- Les jeux de la vie, ou automates cellulaires, sont définis sur une grille de cellules. Les cellules sont dans un état donné (mort ou vivant). L'état des cellules évolue dans le temps en fonction de l'état des cellules voisines selon des règles simples.

Règles de base

- Etat 0 ou 1, i.e. morte ou vivante
- R1 : une cellule morte possédant exactement 3 voisins (vivants) nait, i.e. Etat 0->1
- R2 : une cellule vivante possédant 2 ou 3 voisins (vivants) reste vivante, i.e. Etat 1->1
- R3 : une cellule vivante qui possède moins de 2 voisins (vivants) ou plus de 3 voisins (vivants) meurt par isolement ou surpeuplement i.e. Etat 1->0





Jeu de la Vie

On vous propose dans ces TPs de réaliser une implémentation efficace du Jeu de la Vie

Pour cela vous utiliserez comme matière première le fichier **`masterjdv.py`** (disponible sur https://github.com/benoistgaston/m1sid-2020.git TPs/JdV) qui contient trois fonctions de base

1. La fonction **`init_grid(n)`**
   - prend en entrée une dimension **`n`**
   - retourne une grille 2D de dimension ***`nxn`*** rempli aléatoirement de 0 et de 1
2. La fonction **`get_nbneigh(grid,coord)`**
   - prend en entrée une grille **`grid`** et un couple coord de coordonnée **`i,j`**
   - retourne la liste des valeurs des voisins de **`i,j`** dans la grille
3. Une fonction **`evolution1(grid)`**
   - prend en entrée une grille **`grid`**
   - retourne une grille 2D qui est l'évolution de la grille **`grid`** en application des 3 règles énoncées

Questions

1. Appliquer la fonction **`evolution1`** et estimer les temps de calcul pour des grilles de tailles 10x10 100x100 1000x1000
2. Utiliser **`cProfile`** afin d'identifier les fonctions les plus coûteuses (sur des grilles 2000x2000)
3. Créer une fonction **`evolution1_corr`** qui est une intégration du corps de **`get_nbneigh`** dans **`evolution1`**
4. Utiliser de nouveau **`cProfile`** afin d'estimer un potentiel gain
5. À partir de **`evolution1_corr`**, écrire une fonction **`evolution1_ndarray`** qui modifie la structure de données de res_grid afin d'utiliser un **`ndarray`** 2D
6. Quels impacts sur les performances du code ?

Jeu de la Vie

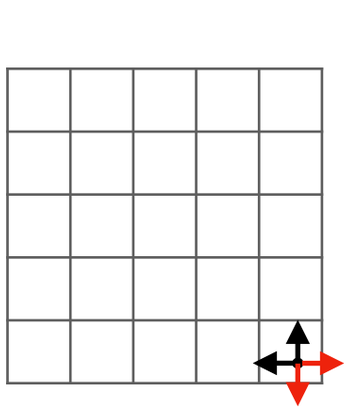
Pour ce TP vous utiliserez comme matière première le fichier **`masterjdv2.py`** (disponible sur https://github.com/benoistgaston/m1sid-2020.git) qui contient (entre autre) les deux fonctions de base

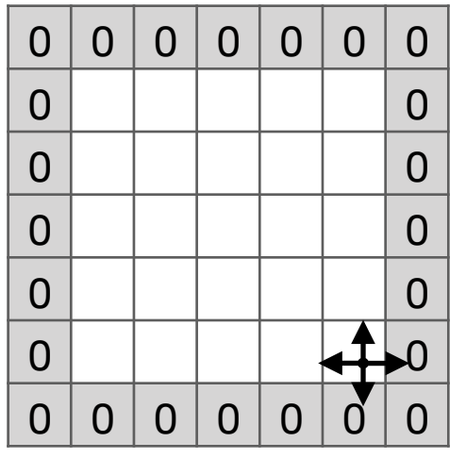
1. La fonction **`init_grid(n)`**
    - prend en entrée une dimension **`n`**
    - retourne une grille 2D de dimension ***`nxn`*** rempli aléatoirement de 0 et de 1
2. Une fonction **`evolution2(grid)`**
    - prend en entrée une grille **`grid`**
    - retourne une grille 2D qui est l'évolution de la grille **`grid`** en application des 3 règles énoncées

- Les structures de données utilisés sont des tableaux 2D **`ndarray`** de numpy
- NB : la fonction **`evolution2`** correspond à une version de la fonction **`evolution1_ndarray`** demandé au TP précédent

Gestion des bords

La gestion des bords nécessite de traiter des cas particulier qui complexifie le code et entraîne des pertes de performance. On propose d'éliminer ce problème en créant artificiellement un bord de cellule morte tout autour de la zone de jeu





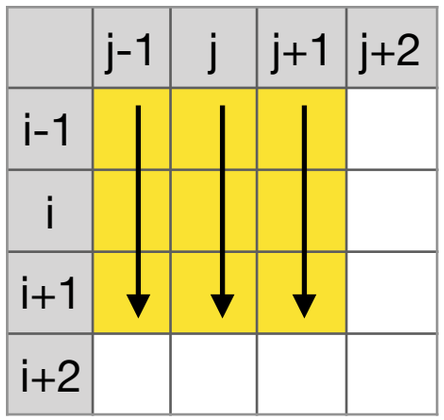
Questions

1. Créer une fonction **`enlarge_grid(grid)`** qui, en fonction d'une grille **`grid`**, retourne une grille élargie d'une colonne et d'une ligne de part et d'autre de la grille **`grid`**
2. À partir de **`evolution2`** créer une fonction **`evolution_eg`** qui applique les règle du jeu de la vie sur une grille élargie. Attention on ne doit pas modifier les valeurs des cellules artificiellement ajoutée au bord
3. Quels impacts sur les performances du code ?

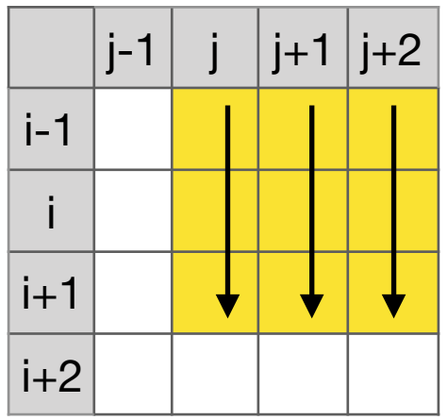
Jeu de la Vie

Enregistrement

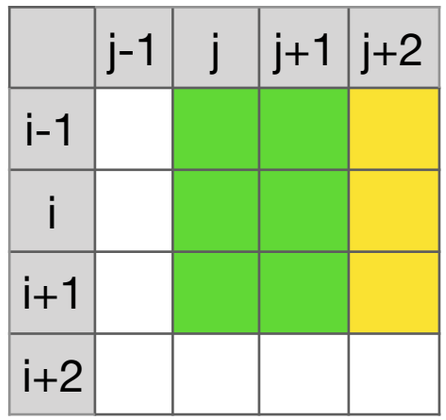
Lors de l'application des règles cellule par cellule on constate qu'un certains nombre de calcul sont répétés à plusieurs reprises comme décrit sur le schéma suivant.



evolution(i,j)



evolution(i,j+1)



valeurs j et j+1 déjà calculées

Questions

1. sur la base de **`evolution2_eg`** une fonction **`evolution2_store`** qui se base sur le principe de stocker les sommes déjà calculées.

2. Quels impacts sur les performances du code ?

MPI : prise en main

1. Écrire un script **`testmpi.py`** qui réalise les actions suivantes (pour chaque processus ***p***):
   - Envoie la valeur ***100+p*** au processus ***p+1*** excepté pour le processus de **plus haut rang** qui n'envoie rien
     Attention : chaque processus excepté le processus 0 reçoit un message
   - Affiche la valeur reçue

2. Exécuter le programme pour 4 processus
   `mpirun -n 4 python3 testmpi.py`

Jeu de la Vie parallèle

Pour ce TP vous pouvez utiliser comme matière première le fichier **`masterjdv3.py`** (disponible sur https://github.com/benoistgaston/m1sid-2020.git) qui contient (entre autre) les fonctions **`evolution_eg`** et **`evolution_store`**, ou repartir de vos propres fonctions.

Jeu de la Vie parallèle

Préliminaire calcul complet

1. Écrire une fonction **`statalive(egrid)`** qui prend en entrée une grille élargie **`egrid`**, calcule le nombre de cellule vivante et retourne le nombre de cellule vivante ainsi que le pourcentage de cellule vivante par rapport au nombre de cellules totales

Attention : ne pas tenir compte des cellules virtuelles/ fantômes

2. Écrire une fonction **`gamelife(grid,n)`** qui prend en entrée une grille **`grid`** et un nombre d'itération **`n`** et applique successivement **`n`** fois l'évolution (en utilisant **`evolution_store`**) et à chaque itération affiche les statistiques calculées par **`statalive`**

Attention : penser à élargir la grille ; à chaque itération on applique évolution_store sur le résultat précédent

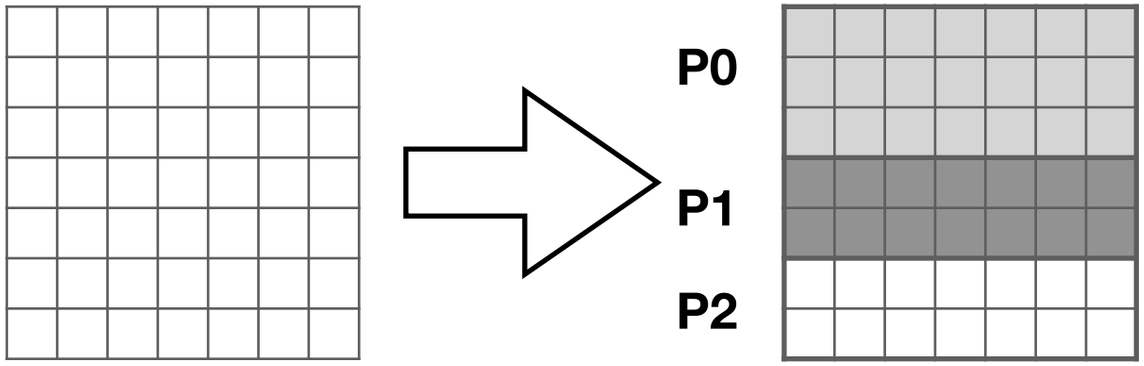
3. Tester sur une grille 1000x1000 et une 10aine d'itérations, quels temps d'exécution ?

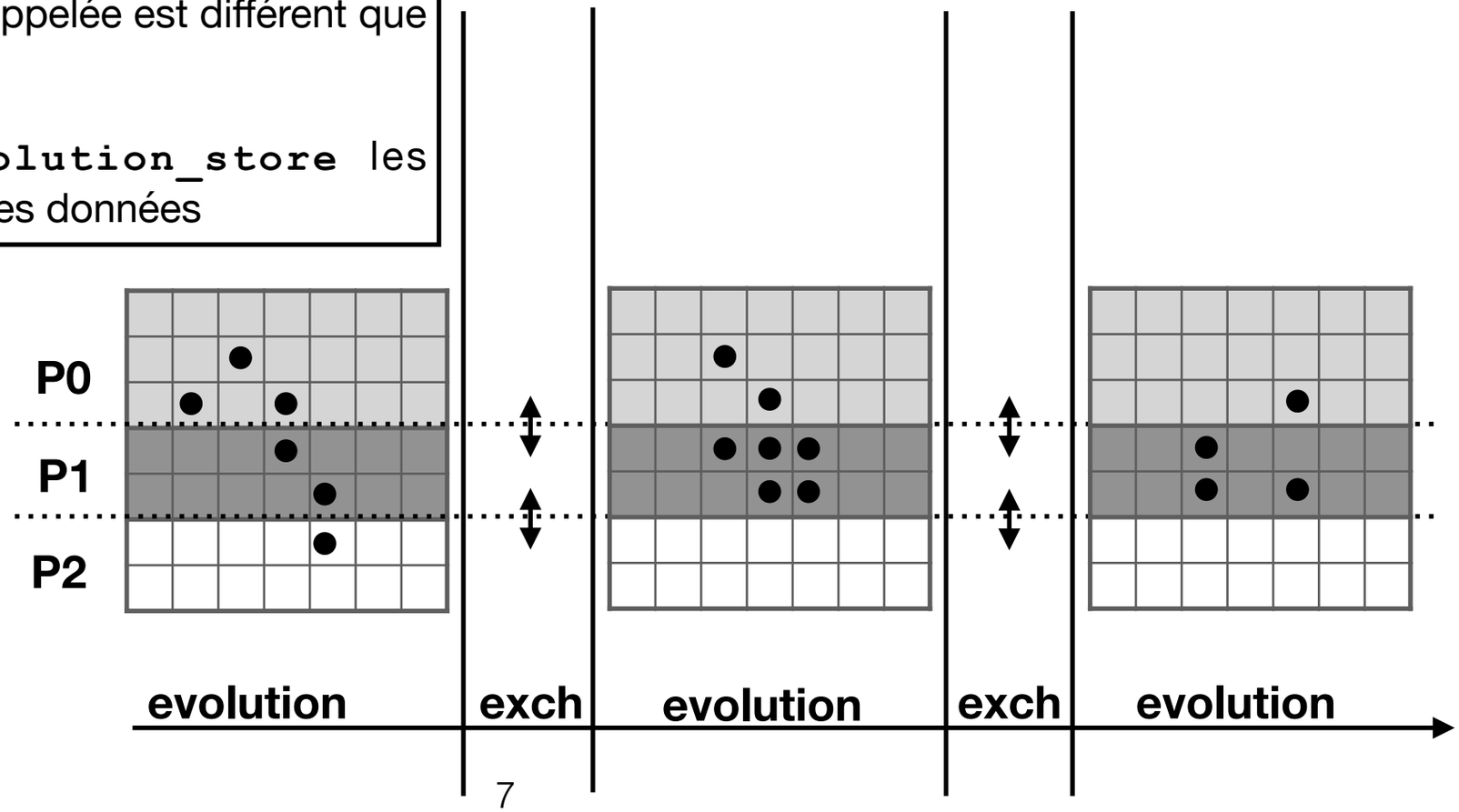
Jeu de la vie Parallèle
Décomposition de domaine

Parallélisation

- On propose de paralléliser **`gamelife(grid)`** par décomposition de domaine
- La décomposition se fait par ligne
- Chaque processus initialisera la grille complète puis travaillera sur un sous-domaine qui lui est propre
- La fonction **`evolution_store`** reste inchangée (seule le domaine sur laquelle elle est appelée est différent que dans le cas séquentiel)
- Entre chaque appel à **`evolution_store`** les processus devront s'échanger des données

Exemple pour 3 processus





Jeu de la vie Parallèle
Décomposition de domaine

1. Écrire une fonction **`idim_local(grid)`** qui en fonction du rang du processus retourne le nombre de ligne locale à traiter ainsi que la position (coordonnée de la 1ère ligne) de la sous-grille dans la grille globale

2. Écrire une fonction **`create_local_grid(grid)`** qui prend en entrée une grille et crée la sous-grille **élargie** locale traitée par le processus courant en créant les zones fantômes et interfaces

- *NB : la décomposition se faisant par ligne, les zones fantômes sont contiguës en mémoire*

3. Modifier **`gamelife`** afin d'y inclure la décomposition de domaine et les échanges entre processus afin de calculer en parallèle

4. Tester sur 4 processus