**Question 2.1**

Dans cette question, on ne s’occupe que de la mesure de temps d’exécution en séquentiel. Les résultats obtenus sont les suivants :

Temps de calcul de la **simulation** par jour : 0.0221145s

Temps de calcul de l'**affichage** par jour : 0.0550189s

Temps de calcul **global** par jour : 0.0779469s

On voit bien que le temps global de calcul est la somme des temps de calcul de la simulation et de l’affichage. Cependant, on remarque aussi que le traitement de l’affichage est ~2.5 fois plus lent que celui de la simulation.

**Question 2.2**

Dans la question suivante on attribue les tâches de la simulation te de l’affichage aux deux processus différents. On obtient les temps suivants :

Temps de calcul de la **simulation** par jour : 0.0742159s

Temps de calcul de l'**affichage** par jour : 0.0738965s

Temps de calcul **global** par jour : 0.0742213s

Le **speed-up** global est presque non-existant (~1.05) à cause de blocage de la part de l’affichage (car les envois/réceptions sont bloquantes et, donc, le processus de simulation doit attendre que le processus d’affichage finit sa tâche avant de passe au jour suivant)

**Question 2.3**

Il semble, alors, logique d’essayer de « sauter par dessus » les jours dans le processus de la simulation si le processus d’affichage n’est pas prêt de recevoir des messages. C’est ce qu’on a fait dans cette question.

Temps de calcul de la **simulation** par jour : 0.021508s

Temps de calcul de l'**affichage** par jour : 0.0214048s

Temps de calcul **global** par jour : 0.0215684s

Les envois non-bloquants permettent d’atteindre le speed-up beaucoup plus important (~3.62), car le processus ne se bloque pas pour chaque envoi (pourtant, on n’affiche pas certains jours).

Maintenant, on devient freiné par le temps de traitement de la simulation, donc, il convient d’esseyer de paralléliser cette partie. C’est ce qu’on fait dans la question suivante.

**Question 2.4**

**Partie 1** (nombre d’individus global constant)

Avec **2 threads omp** :

Temps de calcul **global** par jour : 0.0126236s

S(2 omp + 2mpi) = ~6.24

Avec **3 threads omp**  :

Temps de calcul **global** par jour : 0.00936754s

S(3 omp + 2mpi) = ~8.38

Avec **4 threads omp**  :

Temps de calcul **global** par jour : 0.00788292s

S(4 omp + 2mpi) = ~9.87

Avec **8 threads omp**  :

Temps de calcul **global** par jour : 0.00670961s

S(8 omp + 2mpi) = ~11.64

Dans cette partie on a simplement parallélisé chaque boucle « dans » le processus MPI de la simulation, ainsi que dans la fonction « màjStatistique() », car elle est appelée, elle aussi, dans ce processus-là.

*(Mais je n’ai pas réussi à paralléliser la boucle de l’initialisation de la population. Je comprends qu’il est possible de le faire, mais je n’ai pas arrivé à obtenir des mêmes résultats)*

**Partie 2**(nombre d’individus global constant)

Avec **2 threads omp** :

Temps de calcul global par jour : 0.0251733s

Avec **3 threads omp**  :

Temps de calcul global par jour : 0.0292314s

Avec **4 threads omp** :

Temps de calcul global par jour : 0.0330085s

Ici, on voit bien qu’avec *n* processus, le code est exécutée presque *n* fois moins vite ce qui est un résultat attendu.

**Question 2.5**

Dans la dernière question on cible à paralléliser le code en utulisant plus qu’une tâche MPI (et puis en y ajoutant les threads omp).

**Partie 1** (sans omp)

Avec **2 tâches**:

Temps de calcul **global** par jour : 0.0243669s

S(2mpi) = ~3.21

Avec **3 tâches** :

Temps de calcul **global** par jour : 0.0161931s

S(3mpi) = ~4.81

Avec **4 tâches** :

Temps de calcul **global** par jour : 0.0146227s

S(4mpi) = ~5.32

**Partie 2** (avec omp)

Avec **2 tâches MPI + 2 threads omp** :

Temps de calcul **global** par jour : 0.013354s

Avec **4 tâches MPI + 2 threads omp** :

Temps de calcul **global** par jour : 0.0113361s

Avec **2 tâches MPI + 4 threads omp** :

Temps de calcul **global** par jour : 0.00860194s

Avec **4 tâches MPI + 4 threads omp** :

Temps de calcul **global** par jour : 0.0571396s