CS Games 2016

Compétition de programmation Parallèle

Nombre de participants	**2**
Nombre de postes	**1**
Valeur totale	**5%**
Durée Totale	**3 heures**

Simulation

Depuis la Catastrophe qui presque éradiqué l'humanité ainsi que le reste de la vie sur Terre, l'un grands espoirs des survivants est de pouvoir évaluer les chances de survie de différentes formes de vie primitive en dehors de la sécurité du Dôme.

Pour ce faire, un système de modélisation a été conçu par les chercheurs du Dôme. Il permet de simuler le développement de cellules primitives dans un environnement hostile.

La difficulté à laquelle nous faisons face demeure qu'une telle simulation, effectuée à une échelle réaliste afin de donner fruit à des résultats utiles, consommerait un nombre incalculable de ressources. Nous devons donc trouver un moyen d'exploiter pleinement les capacités des machines du Dôme pour améliorer la vitesse d'exécution de la simulation.

Ces résultats nous aideront enfin à déterminer si la survie est bien possible en dehors du Dôme...

Énoncé

Votre mandat est d'exploiter la programmation parallèle et concurrente afin d'optimiser la simulation.

Le système de modélisation développé par les chercheurs du Dôme prédit le cycle de vie d'un groupe de cellules selon leur environnement. Vous devez, à l'aide d'un environnement initial, simuler l'évolution des cellules au cours de plusieurs générations, selon les règles du modèle.

Description des Fonctionnalités

Votre programme recevra en entrée la description d'un environnement initial, représenté sous la forme d'une matrice $n \, x \, m$ contenant les valeurs 0 et 1. Ces valeurs représentent respectivement un cellule morte et une cellule vivante.

Chaque cellule aura un état déterminé par celui de ses 8 voisins. Pour les cellules en périphérie de la grille, on considérera que la grille est circulaire.
Ainsi lorsqu'un cellule en (0,0) voudra vérifier l'état de son voisin du dessus, on regardera l'état de la cellule (0, matrix_height - 1).

L'état d'une cellule à la génération suivante dépend de l'état actuel de son entourage, selon les règles suivantes :

- ❏ Une cellule ne qui ne possède pas au moins deux voisins meurt;
- ❏ Une cellule ayant exactement deux voisins conserve son état actuel;
- ❏ Une cellule morte ayant exactement trois voisins prend vie;
- ❏ Une cellule ayant plus de trois voisins meurt.

Exemple

A l'instant t, admettons que nous ayions la grille suivante:

1	0	1	0	0
1	0	1	0	0
1	0	1	0	0
1	0	1	0	0
1	0	1	0	0

Après une itération, nous aurons le résultat suivant:

1	0	1	1	1
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1
1	0	1	1	1

Votre programme doit être écrit en C ou C++. Sont permises les bibliothèques standard des langages, ainsi que les bibliothèques de programmation parallèle **pthreads** et **Threading Building Blocks**.

Specification des fichiers d'entrée

Un fichier de description d'état de base de jeu de la vie est basé sur la spécification suivante:

- La première ligne du fichier est réservée pour la taille de la matrice. Pour simplifier le problème, les matrices seront nécessairement carrées.
- Chaque ligne qui suivra cette première ligne sera le contenu d'une ligne, composée de successions de 1 et 0, 1 signifiant que la cellule est vivante à l'état de départ, 0 qu'elle est morte.

Votre programme devra prendre deux arguments pour s'exécuter: le chemin du fichier d'entrée, suivi d'un nombre d'itérations à effectuer.

Correction

Votre programme sera évalué par rapport à son efficacité, soit la vitesse d'exécution - une moyenne du temps d'exécution sera calculée à l'aide de `GNU Time`.

L'utilisation mémoire est également une préoccupation, puisque les entrées peuvent grandement varier en terme de taille. Faites une utilisation judicieuse de vos ressources!

A cause de la grande taille des tests que nous possédons, nous vous remettons une partie de ces derniers pour que vous puissiez tester l'efficacité de votre solution sur ces derniers.

Pour faciliter la correction, vous devez rendre une solution compatible avec les squelettes fournis avec l'énoncé. Vous devrez modifier le Makefile autant que nécessaire pour assurer son fonctionnement. A la remise, le Makefile devra produire le même binaire que lorsqu'il vous est remis. Le non-respect de ces consignes entraîne une pénalité de 5%.

Annexe

Utilisation des bibliothèques de programmation parallèle

Thread C

```c
int pthread_create(pthread_t restrict thread,
                   const pthread_attr_t *restrict attr,
                   void *(start_routine)(void*),
                   void *restrict arg);
```

Mutex C

```c
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                       const pthread_mutexattr_t *restrict attr);

int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Thread C++

```cpp
std::thread leThread (foo);
```

Mutex C++

```cpp
lock();
Unlock();
```

Suggestions d'optimisations

Dans la mesure où le temps d'exécution est le principal critère de notation pour cette épreuve, les participants sont encouragés à améliorer l'efficacité de leur solution via des optimisations apportées sur la partie séquentielle de leur algorithme ou sur leur stratégie de parallélisation.

Des optimisations conseillées sont de limiter l'espace de travail en élaguant les zones mortes de la quantité de données à calculer pour passer à une itération ultérieure.

En plus d'une stratégie simple de découpage statique de la matrice pour la distribuer à d'autres threads, il est conseillé d'essayer d'autres stratégies comme des distributions dynamiques couplées à la stratégie d'optimisation séquentielle présentée plus haut.
Il est également important dans cette épreuve de ne pas négliger la micro-optimisation, l'ordre des opérations et la copie locale de parties de la matrice peuvent avoir un effet sur le cache du processeur et par conséquent améliorer encore plus les temps d'exécution de votre programme.