Projet de Programmation des Architectures Parallèles
Travail en binôme

Jeu de la vie

Le jeu de la vie (cf. https://fr.wikipedia.org/wiki/Jeu_de_la_vie) inventé par J. Conway en 1970, simule une forme très basique de vie en partant de deux principes simples : pour pouvoir vivre, il faut avoir des voisins ; mais quand il y en a trop, on étouffe. Le *monde* est ici un grand damier de cellules vivantes ou mortes, chaque cellule étant entourée par huit voisines. Pour faire évoluer le monde on découpe le temps en étapes et, pour passer d'une étape à la suivante, on compte pour chaque cellule le nombre de cellules vivantes parmi ses huit voisines puis on applique les règles suivantes :

- Une cellule morte devient vivante si elle a exactement 3 cellules voisines vivantes – autrement elle reste morte.
- Une cellule vivante reste vivante si elle est entourée de 2 ou 3 cellules vivantes – autrement elle meurt.

Notre objectif est de comparer l'efficacité différentes implémentations de ce jeu. À partir d'une version séquentielle très naïve vous allez, dans un premier temps, produire trois versions séquentielles :

- Version de base : le monde est parcouru à l'aide de deux boucles *for* imbriquées, on cherchera à optimiser le calcul en réduisant *considérablement* le nombre de sauts conditionnels ;
- Version tuilée : on considère que le monde est divisé en tuiles carrées (tuiles de 32 x 32 cellules, par exemple) - le monde est parcouru selon les tuiles (quatre boucles *for* imbriquées) ;
- Version optimisée : le monde est parcouru selon les tuiles mais on évite de recalculer les tuiles que l'on sait stagnantes c'est-à-dire que l'on sait qu'elles ne vont pas évoluer lors de la prochaine itération.

Puis il s'agira de paralléliser ces versions à l'aide d'OpenMP, OpenCL et MPI. Voici les versions attendues :

- Versions OpenMP for : base, tuilée, optimisée ;
- Versions OpenMP task : tuilée, optimisée ;
- Versions OpenCL : tuilée ;
- Versions MPI : base.

De plus vous traiterez une fonctionnalité avancée parmi les suivantes :

- Version MPI + OpenMP ;
- Version MPI + cellules fantômes ;
- Version OpenCL optimisée ;
- Version mixte OpenCL + OpenMP en répartissant une partie du monde sur CPU et l'autre partie sur le GPU.

NB1. On ne cherchera à détecter la stabilité du système que pour les versions optimisées.

NB2. Un trinôme traitera trois fonctionnalités avancées – une personne seule en traitera deux.

Étude expérimentale à mener sur les machines de la salle 203.

Il s'agit de comparer l'efficacité des différentes techniques de parallélisation en produisant des courbes de *speed-up* fonction du nombre de cœurs employés voire du nombre de machines. Pour ce faire, on considèrera au moins les paramètres suivants :

- Motif initial de la configuration : « guns » ou « random » ;
- Taille du monde : 512 et 4096 ;
- Type de scheduling OpenMP : static, dynamic;
- Taille des tuiles : 16 & 32 pour OpenCL, à déterminer pour OpenMP.

L'objectif est donc de comparer les performances des techniques employées sur différents cas mais aussi d'expliquer autant que possible le comportement du programme. L'idée n'est pas de produire toutes les courbes possibles mais de proposer et de suivre un plan d'expérimentation. Ainsi on commencera par étudier individuellement chaque technique en faisant varier quelques paramètres afin de produire des courbes ou des tableaux en vue de dégager les meilleurs paramètres. Il s'agit aussi de décrire les courbes obtenues, de les comparer et de repérer les phénomènes étranges (pour vérifier qu'on ne s'est pas bêtement trompé). Idéalement, on raisonnera à partir de la description des courbes afin de formuler des hypothèses pour expliquer les comportements observés et puis on validera / infirmera les hypothèses à l'aide de nouvelles expériences. Voici quelques exemples d'observations et de pistes d'interprétations :

- Un speedup supérieur au nombre de cœurs est constaté. Avez-vous utilisé le meilleur algorithme séquentiel ? Gagne-t-on grâce au non-déterminisme du parallélisme ? Bénéficie-t-on d'effets de cache favorables ?

- On observe que l'ordonnancement "dynamic" est beaucoup moins performant que le "static". Est-ce un problème de contention sur un mutex ? Est-ce un problème de localité mémoire (cache / NUMA) ?

- On constate que l'ordonnancement "static" est beaucoup moins performant que le "dynamic". La charge de travail est-elle bien équilibrée ? la mémoire est-elle bien répartie sur la machine ?

- On observe que la courbe d'accélération croit puis décroit. Y-a-t-il suffisamment de travail pour compenser le surcoût de la parallélisation ? Y-a-t-il un déséquilibre dû à *l'hyperthreading* ? Y-a-t-il de la contention (cache / mémoire) ?

Finalement on pourra comparer les différentes techniques programmées et pour conclure sur une discussion « effort de programmation vs performances obtenues ».

Options de compilation

```
make NOSDL=1              # désactive l'utilisation de la SDL
make ENABLE_MPI=1         # utilisation de mpicc pour la compilation
```

On utilisera `#ifdef ENABLE_MPI … #endif` pour baliser le code MPI.

Rendu : Jeudi 3 mai 14h

Trois fichiers sont à recopier dans le répertoire `/net/cremi/pwacreni/PROJET-PAP :`

- nom1-nom2-clef-secrete.pdf : rapport du projet ;
- nom1-nom2-clef-secrete.cl : fichier source opencl du projet ;
- nom1-nom2-clef-secrete.c : fichier source C du projet.

Le rapport présentera des extraits du code illustrant les différentes techniques employées, les expériences réalisées et leurs commentaires.

Soutenances : Vendredi 4 mai entre 10h et 18h