Charles Thonier

Programme colonisation

Je n'ai pu faire que la parallélisation des boucles avec OpenMP.

Lors du projet, j'ai pu tester les limites de OpenMP pour la parallélisation de certaines tâches. Lors de tous les essais décrits dans le tableau ci-dessous, les versions de code utilisant des fonctions de OpenMP ont mis plus de temps que l'algorithme séquentiel.

Cela peut parfaite surprenant, mais c'est en réalité compréhensible, les fonctions pragma for étant d'une part des boites noires, et d'autre part les fonctions aléatoires choisies par défaut utilisant des mutex qui n'ont rien à voir avec ceux qu'OpenMP déclare.

Dans le but de parer à cela, j'ai défini mes propres fonctions aléatoires sur le modèle de celles exposées dans les dernières séances de cours, mais l'algorithme s'en est retrouvé encore ralenti. J'ai ensuite demandé à Arthur Liu et Nicolas Drouin de me fournir des fonctions génératrices de nombres aléatoires, mais elles étaient un peu moins performantes que les miennes.

En outre, j'ai brièvement essayé d'implémenter une solution utilisant des threads, puis OpenMPI pour paralléliser l'affichage (cela me semblait plus simple que de paralléliser les calculs), mais sans succès.

Variante du code	Nombre de processus	CPU (ms)	Speedup calcul	Affichage (ms)	Speedup affichage	Stabilité (durée années)
Programme de base	1 (séquentiel)	6,456	-	1,046	-	Oui (3M)
Pragma pour les deux for de mise_a_jour	6	6,482	0,9960	1,054	0,9924	Oui (3M)
Pragma pour les deux for de mise_a_jour et les deux for de render	6 et 6	9,950	0,6488	1,3861	0,7546	Oui (2M)
Pragma pour les deux for de mise_a_jour avec manipulation du seed	6	14,392	0,4485	1,410	0,7418	Oui (3M)
Pragma pour les deux for de mise_a_jour et les deux for de render avec manipulation du seed	6 et 6	14,547	0,4438	1,171	0,8932	Oui (3M)
Pragma pour la plus grande boucle for de mise_a_jour avec les fonctions aléatoires de Arthur	6	19,078	0,3384	1,3383	0,7816	Oui (3M)
Pragma pour les deux for de mise_a_jour et les deux for de render avec les fonctions aléatoires de Arthur	6 et 6	18,777	0,3438	1,229	0,8511	Oui (3M)

Fonction pragma omp for appliquée dans paramètres.cpp et générateur aléatoire :

```
void
mise_a_jour(const parametres& params, int width, int height, const char* galaxie_previous, char*
    galaxie_next)
{
    int i, j;

    unsigned seed1 = std::chrono::system_clock::now().time_since_epoch().count() +
        173501*omp_get_thread_num();
    std::minstd_rand0 generator (seed1);
    std::uniform_real_distribution<double> distribution(0.0, 1.0);
    std::uniform_int_distribution<int> choix(0,3);

    memcpy(galaxie_next, galaxie_previous, width*height*sizeof(char));
    int static nb_process = 6;
#pragma omp for schedule(static, width*height/nb_process)
    for ( i = 0; i < height; ++i )
      {
        for ( j = 0; j < width; ++j )
        {
```

Exemple de fonctions utilisant le générateur aléatoire :

```
bool apparition_technologie(const parametres& p,
                            std::uniform_real_distribution<double>& rd,
                            std::minstd_rand0& generator)
{
    double val = rd(generator);
    if (val < p.apparition_civ)
        return true;
    return false;
}
```