T.P. Parallélisme
Parallélisation d'un programme de lancer de rayons

17 janvier 2014

1 Programme de lancer de rayons séquentiel

On dispose des sources d'un programme séquentiel de lancer de rayons. Ce programme a été écrit par Gilles Subrenat, ancien Étudiant de l'ENSEIRB, qui a eu la gentillesse de le donner à des fins de pédagogie. Les sources de ce programme sont accessibles à partir de l'archive `http://uuu.enseirb.fr/~mfaverge/PRCD/LancerRayons/LancerRayons.tar.bz2`.

Entrée du programme : un fichier .scn qui décrit une scène, c'est à dire :
- la description et la position d'un ensemble d'objets constituant la scène
- la position des sources lumineuses
- la position et l'angle du vue de la caméra

Sortie : une image au format .ppm

Le programme utilise un algorithme de lancer de rayons afin de produire l'image à partir de la description de la scène. Dans une scène réelle, les rayons de lumière issus des sources lumineuses se propagent dans toutes les directions et se réfléchissent et/ou sont réfractés sur les objets qu'ils rencontrent sur leur passage. Ce que nous percevons de la scène correspond uniquement à l'ensemble des rayons qui convergent vers notre oeil. Dans le cadre d'un calcul sur ordinateur, afin d'éviter d'avoir à simuler le parcours d'une grande quantité de rayons, il serait judicieux de ne calculer que le parcours des rayons arrivant sur l'oeil. Pour cela, on utilise le principe du retour inverse de la lumière : le parcours d'un rayon lumineux ne dépend pas de son sens. On fait donc partir un ensemble de rayons depuis l'oeil (c'est-à-dire la position de la caméra), chacun des rayons correspondant à un pixel de l'image qu'on souhaite calculer. On simule alors la propagation de ces rayons en suivant les lois de réflexion et de réfraction, jusqu'aux sources lumineuses. Rassurez-vous, il n'est pas nécessaire d'avoir compris cet algorithme pour pouvoir accomplir le TP. En effet, le programme de lancer de rayons est déjà entièrement écrit. Tout ce que vous avez besoin de savoir, c'est qu'une image est un tableau à deux dimensions de pixels, et qu'à chaque pixel est associé une couleur décrite sous la forme de ses trois composantes rouge, verte et bleue de la lumière. La partie principale du programme consiste à faire une boucle séquentielle sur tous les pixels (i, j) de l'image, et à appeler la fonction `pixel_basic` sur ce point pour calculer les composantes du pixel (fichier `src/img_basic.c`).

Ce programme, une fois compilé, prend un fichier de description de scène au format `.scn`, et produit en sortie un fichier `.ppm`. Plusieurs exemples de fichiers scènes et images sont fournis dans le répertoire `./scn/` de l'archive. Dans le répertoire `./scn/exemple`, on trouve par exemple les fichiers `test.scn` et l'image résultante `test.ppm.gz`.

On calcule une image en se plaçant dans le répertoire contenant le fichier de description de la scène (et ses sous-fichiers `.rriff` éventuels), et en y exécutant le programme lanceur, avec comme seul paramètre le nom de base du fichier scène. Ainsi, la commande `lanceur brol` prendra comme fichier scène le fichier `brol.scn`, et produira l'image `brol.ppm` dans le même répertoire.

2 Parallélisation du programme

Le but de ce TP est de réaliser plusieurs versions parallèles de ce programme. On essaiera pour cela de contenir au maximum les modification dans le fichier `src/img.basic.c`. La technique de parallélisation choisie consiste à dupliquer l'ensemble de la scène sur chaque processus (tous les processus liront le fichier scène), et à répartir le calcul des pixels sur les processus en attribuant à ceux-ci des portions d'image de petite taille (que nous appellerons ici "carreaux", de taille 8×8 ou 16×16). Une fois l'image calculée, le processus 0 sera en plus en charge de récupérer les carreaux de l'image calculés par les autres processus, afin de produire l'image finale, toujours au format `ppm`.

La difficulté de ce TD réside dans la répartition de la charge de manière équitable sur l'ensemble des ressources. Pour cela, il faut s'assurer que chaque processus possède dans sa file locale de travaux un même nombre de carreaux, et que ceux-ci proviennent de l'ensemble de l'image, afin que les hétérogénéités de la complexité de l'image soient réparties sur l'ensemble des processus. Une version avec distribution statique des données constituera la première version parallèle du programme à produire.

Comme cette répartition statique peut ne pas être suffisante pour assurer un équilibrage optimal de la charge, on mettra en place un mécanisme de répartition dynamique de charge basé sur la demande de travail aux autres processus, parfois appelé "vol de travail" ("work stealing") dans la littérature. Ceci constituera la deuxième version du programme à fournir.

3 Travail à réaliser

1. Compilez les sources de la version séquentielle sur votre système et testez le programme.
2. Réalisez la version parallèle de ce programme.
   - **Modifiez le programme séquentiel le moins possible**. Le calcul de l'image proprement dite ayant lieu dans le fichier img.basic.c pour la version séquentielle, une copie appelée img.mpi-static.c existe, essayer de ne modifier que ce fichier, en appelant si besoin des routines contenues dans d'autres fichiers que vous créerez.
   - Le but de l'exercice est de calculer une distribution *correcte* des carreaux sur les différents processus sans aucune communication. Pour cela, vous décrirez dans votre rapport différentes solutions et leur avantages/inconvénients. La solution qui devra être implémentée repose sur les propriétés du théorème des restes chinois. On considère que l'on a P processus et C carreaux au total, tout processus P_i aura $N_i = \lfloor (C - i)/P \rfloor + n_i$ carreaux. n_i est égal à 1 si $i < C\%P$, 0 sinon. Tout processus P_i peut donc connaître le nombre de carreaux distribués à chaque processus. Mais il doit également (principalement pour le processus 0) connaître la distribution des carreaux sur les processus. Le théorème des restes chinois permet de s'attribuer ses propres carreaux sans risques de conflits (et de calculer ceux des autres) à condition d'avoir un nombre entier N premier avec C. Le processus P_i aura alors les carreaux dont les indices appartiennent à l'ensemble de tailles N_i suivant :
   $$\{(j * N)\%C, j \in [\![\sum_{k=0}^{i-1} N_k, \sum_{k=0}^{i} N_k]\!]\}$$
   .
3. Mettez en place une politique de vol de travail pour réguler dynamiquement la charge. Ce système fonctionne de la façon suivante.
   - Lorsqu'un processus n'a plus de carreaux à traiter dans sa file locale de travaux en attente, il envoie de proche en proche un message de demande de travail à travers un anneau logique des processus.
   - Si l'un des processus récepteurs possède encore au moins deux carreau dans sa file de travaux en attente, il renvoie directement au processus demandeur la description du travail à réaliser, et le défile de sa file locale.

— Si le message de demande de travail revient à son émetteur, celui-ci sait que tous les processus sont sur le point de terminer, car ils n'ont plus de travail à lui déléguer. Il peut donc envoyer un message de demande de terminaison sur l'anneau logique, afin que tous les processus qui le reçoivent ne demandent pas de travail à leur tour mais terminent lorsqu'ils auront épuisé leurs file locale. Dès qu'un processus qui a envoyé un message de terminaison reçoit un message de terminaison (que ce soit le sien qui a fait le tour de l'anneau, ou bien celui d'un autre processus), il peut alors terminer sans le faire suivre au préalable sur l'anneau logique.

— La politique de vol de travail est totalement indépendante du code de lancer de rayons. Elle peut donc être implémentée en parallèle du développement de la version 1 à l'aide de tache factice qui exécute un fonction du type :

```
void traitement_tache() {
  int i, cpt=0;
  int z = (rand() / RAND_MAX) * 20; /* entier aléatoire entre 0 et 20 */
  sleep(z);
}
```

4. Couplez le vol de travail au code de lancer de rayons dans le fichier img.mpi-dyn.c.
5. Chaque noeud possédant plusieurs coeurs, on souhaite avoir une implémentation hybrides MPI/Thread. Un thread s'occupera de l'anneau et plusieurs threads de calcul l'accompagneront pour faire le calcul. Cette version sera implémentée dans le fichier img.mpi-thread.c