FERNANDEZ
Thomas
PI2

Compte rendu du projet d'informatique du semestre 7.

1. Projet choisi :

Le sujet choisi se porte sur la recherche des chemins de percolation.

2. Etat du logiciel :

La version minimale et la création de GIF animés fonctionnent. La parallélisation du problème est quant à elle un échec (cf readme « Problèmes de parallélisation). L'implémentation de piles avec un tableau redimensionnable ainsi que l'adaptation à d'autres types de réseaux n'ont pas été explorées.

3. Structures de données utilisées :

- PILE :

La recherche de chemins de percolation itérative demande l'implémentation du type PILE. Ainsi, par définition, il faut définir la structure :

```
struct stack {
        int tete;
        PIXEL* liste;
} ; typedef struct stack PILE;
```

La liste où sont stockées les données empilées est allouée dynamiquement par la fonction nouvellePile() (p.liste = calloc(nbColonnes * nbLignes, sizeof(PIXEL));) afin de régler un problème créé par la parallélisation (cf readme « Piles allouées dynamiquement »)

- *PIXEL :*

Grâce à l'allocation contiguë de la matrice réseau, chaque noeud de ce dernier est caractérisé par ses deux coordonnées (x,y). Ainsi, pour pouvoir les traiter comme une seule entité, il faut implémenter la structure suivante :

```
struct pxl {
        int x;
        int y;
} ; typedef struct pxl PIXEL;
```

- *RESEAU :*

Les réseaux, quant à eux, se doivent de stocker les états des différents noeuds qu'ils contiennent. Pour les implémenter, on utilise la structure :

```
struct res {
	int** mat;
	int* nds;
	int nbLi;
	int nbCol;
	int percole;
} ; typedef struct res RESEAU;
```

Avec nbLi le nombre de lignes et nbCol le nombre de colonnes du réseau, percole un booléen indiquant si un chemin de percolation a été trouvé (1) ou non (0), nds un tableau à une dimension stockant les états des différents noeuds (0 = mur, 1 = espace vide, 2 = conducteur) et mat un tableau contenant des pointeurs vers le début de chaque ligne. Il permet l'utilisation de la syntaxe mat[y][x] (soit mat[numéro de la ligne][numéro de la colonne].

- *ARG :*

Cette structure permet d'offrir à chaque thread les informations dont il a besoin pour exécuter sa charge de travail. Ainsi, elle contient le nombre de test à exécuter ainsi que la probabilité p pour laquelle le thread va créer des réseaux.

```
struct args {
	unsigned long workload_per_thread;
	float prob;
} ; typedef struct args ARG;
```

4. Méthodologie utilisée pour les tests :

L'écriture de chaque nouvel aspect du programme se faisait isolé dans un répertoire différent. Une fois greffé au projet principal et en cas d'erreur, le débogage consistait encore une fois à isoler la partie problématique en commentant une à une les nouvelles fonctions en partant de celle exécutée dans le main et en retraçant manuellement les opérations effectuées par la machine jusque'à trouvée la ligne coupable.

5. Test des fonctionnalités demandées : (cf. Readme)

Après avoir *git pull* le répertoire, le programme peut être exécuté pour la premier fois avec la commande *./bin/main.exe* après avoir tapé les commandes *Make clean* et *Make*.

Le choix de la fonctionnalité à tester est effectué après exécution en tapant le numéro correspondant.

Une fois les calculs terminés, les résultats des fonctionnalités 2. Calcul et tracé de la probabilité qu'un réseau percole en fonction de p et 3. Calcul du gain de temps pratique après parallélisation peuvent être obtenus en utilisant respectivement les commandes *Make graph* ou *Make gif*.

6. Analyse des performances et mémoire.

L'échec face à la parallélisation du problème le rend particulièrement inefficace. Obtenir un résultat proche de celui donnée dans le sujet lors de la partie 2. Calcul et tracé de la probabilité qu'un réseau percole en fonction de p peut prendre plusieurs centaines voire milliers de secondes.

De plus, l'utilisation de structures PILE lors des recherches itératives implique, dans ce programme, que très souvent, les noeuds au dessus de celui étudié seront pris en compte avant ceux du dessous, affectant considérablement les performances (problème illustré dans la partie « Création d'un GIF animé. » - « resultat » du Readme)

Il ne devrait normalement pas avoir de fuite de mémoire mais aucune vérification avec Valgrind a été faite après changement de la méthode d'allocation des piles.

7. Outils de développement utilisés.

- Valgrind
- HomeBrew
- Git
- Gcc

8. Organisation au sein de l'équipe.

Projet réalisé seul.

9. Conclusion.

Bien que malheureusement inachevé, ce projet en semi-autonomie fut très formateur. En effet, devant un domaine non abordé en cours (le multithreading) et des erreurs peu explicites, il m'a été nécessaire de faire beaucoup de recherches afin de pouvoir implémenter et tester des nouvelles idées (très souvent vaines) dans l'espoir d'enfin obtenir un temps de calcul acceptable.

Ce projet fut également l'occasion pour moi de toucher à de l'informatique « moins scolaire » qu'en classe préparatoire ce qui fut une expérience très positive.