DÉPARTEMENT D’INFORMATIQUE

Faculté des sciences

Université de Sherbrooke

Solutionnaire de sudoku en parallèle

par

TOMMY GRÉGOIRE (10 126 605)

ÉTIENNE DUPÉRÉ LARIVIÈRE (12 059 611)

travail présenté à

GABRIEL GIRARD

dans le cadre du cours

IFT 630

Processus concurrents et parallélisme

Sherbrooke

AVRIL 2014

# 1 Description d’un Sudoku

Le jeu du Sudoku se joue à l'aide d'une grille composée de 9 blocs disposés en trois lignes / colonnes comportant chacune trois blocs. Les blocs se composent à leur tour de 9 cases disposées en trois lignes / colonnes de 3 cases. L'objectif du jeu est de compléter une grille partiellement remplie avec les chiffres de 1 à 9 sans que le même chiffre figure plus d'une fois dans chaque ligne, chaque colonne et chaque bloc. La figure 1.1 suivante illustre une telle grille.

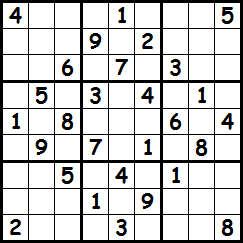


Figure 1.1 : Grille de sudoku

# 2 Description générale du projet

Les projets que nous avons créés consistent en deux programmes qui résolvent certaines grilles de Sudoku. La première solution utilise l'intelligence artificielle pour résoudre séquentiellement la grille et la deuxième solution utilise les «pthreads» pour permettre l'exécution en parallèle. Il existe plusieurs approches pour résoudre des grilles, celles que nous avons choisi d'implémenter s'inspirent des méthodes de résolutions humaines.

Certains puzzles de Sudoku nécessitent des techniques de résolution complexe qui considèrent l'ensemble des cases de la grille pour prendre une décision (ex: backtracking). La solution en parallèle que nous avons développée n'est pas capable de résoudre de tels puzzles. Celle-ci permet de résoudre des grilles qui sont considérées comme niveau facile ou moyen.

## 2.1 Solution séquentielle

Pour notre solution séquentielle, nous avons choisi un algorithme CSP (Constraint satisfaction problem)[[1]](#footnote-1). La solution comprend une liste de variables, un domaine appliqué à celles-ci ainsi qu’une liste de contrainte. Les variables correspondent à chaque case de la grille, le domaine quant à lui correspond au chiffre de 1 à 9 puis les contraintes sont l’unicité de chaque valeur dans une même ligne, colonne et bloc.

Premièrement on lit le fichier contenant le sudoku afin de remplir le tableau avec chaque élément déjà trouvé. Ensuite, une première itération de l’algorithme peuple toutes les valeurs possibles dans les cases restantes, ces valeurs sont déterminées avec la ligne, la colonne et le bloc de chaque case. Une fois cette étape faite, un processus récursif de marche arrière débute pour résoudre le sudoku.

La première étape de la récursivité est de trouver la position de l’élément de la grille qui a le moins de valeur possible. Ensuite, on essaie chaque valeur de cette case jusqu’à en trouver une qui ne cause pas de conflit. Dans cette boucle on cherche d’abord la valeur qui a le moins de contraintes au niveau de ses pairs, en d’autre mot la valeur se retrouvant le moins souvent dans les valeurs possibles des pairs de la position cherchée. Une fois la nouvelle valeur trouvée, on appelle récursivement la procédure de marche arrière avec cette nouvelle valeur. Cette récursivité s’effectue jusqu’à trouvé la valeur cherchée ou jusqu’à atteindre un chemin sans issue. Une fois une valeur erronée testée jusqu’au bout, celle-ci sera retirée de la liste de valeur possible et la prochaine valeur pourra être essayée à son tour.

### 2.1.1 Architecture du programme

Le programme est divisé en deux fichiers, soit le fichier principal « SudokuSolver.cc » du programme qui contient l’algorithme et toutes les fonctions nécessaires à celui-ci. Le programme principal fait appel à la classe « cell.h » qui encapsule l’information reliée à une case de la grille.

### 2.1.2 Format du fichier d’entrée

Le format du fichier d’entrée est simple, il y a neuf lignes contenant chacune neuf valeurs séparées par des espaces. Les valeurs possibles sont :

* 1 à 9 pour les valeurs initiales de la grille;
* 0 pour les valeurs manquantes.

### 2.1.3 Exécution du programme

Le programme fut développé sous Linux et compilé avec g++ 4.8.1. La compilation et l’exécution sont effectuées à partir de ligne de commande sous Linux. Le fichier d’entrée utilisé pour la résolution du sudoku est le même pour les deux types d’algorithmes, soit « sudoku.in.txt ». Afin d’en choisir un autre, il faut modifier la fonction « main() » du programme principal pour utiliser le fichier désiré et ensuite recompiler l’application.

Voici les commandes de compilation et d’exécution du programme séquentiel :

* Compilation : g++ -w -o sudoku SudokuSolver.cc cell.cc cell.h
* Exécution : *./sudoku*

## 2.2 Solution parallèle

Le programme procède par élimination avec la supposition initiale que toutes les valeurs sont possibles pour chaque case. Lorsqu'une valeur est définie pour une case, elle est éliminée comme possibilité pour les autres cases du bloc, de la ligne et de la rangée qui la contiennent.

### 2.2.1 Architecture du programme

Le programme a été écrit en C++ et les threads se partagent en mémoire global la grille principale (CREW).

La classe Sudoku est essentiellement un gestionnaire qui charge la grille initiale et qui démarre tous les threads. Lorsque tous les threads sont parvenus à une solution (Join de tous les threads) elle s’occupe d’afficher la solution finale et de nettoyer les pointeurs désuets.

Il y a neuf threads démarrés, soit un pour chaque bloc de la grille. Chaque thread possède une instance de la classe SubGrid(bloc). La classe SubGrid regroupe toutes les fonctions qui permettent d’initialiser un bloc et de nettoyer les pointeurs désuets du bloc. Elle contient aussi les fonctions qui cherchent une solution possible pour une case du bloc. Les lectures dans la grille principale pour vérifier s’il y a une possibilité pour la case sont concurrentes. Le principe de ces fonctions marche ainsi: si après avoir enlevé toutes les possibilités du bloc, de la ligne et de la colonne il reste une seule solution, on la retourne, sinon on retourne la valeur zéro.

La classe « solver » sert principalement à prendre les solutions trouvées et les insérer dans la grille principale. L’écriture dans la grille principale respecte l’exclusion mutuelle. Chaque thread boucle un certain nombre de temps et si on n’a pas trouvé la solution, c’est que la grille est trop difficile à résoudre.

### 2.2.2 Format du fichier d’entrée

Les grilles sont stockées dans un fichier texte comportant une ligne pour chaque ligne de la grille. La valeur de la première ligne indique la grosseur de la grille (ex: 9 = 9X9). Les valeurs doivent être séparées par un caractère d’espace. La valeur 0 indique que la case est vide.

### 2.2.3 Exécution du programme

L'exécution du programme doit se faire sous Linux.

* Compilation: sh makeLinux.sh
* Exécution: *./sudoku*

# 3 Conclusion

Finalement, beaucoup de temps ont été investi dans la recherche d’algorithme et en autre dans l’algorithme séquentiel. La différence de temps de résolution entre le programme parallèle et séquentiel n’était pas significative. Donc, développer en parallèle ne vaut pas nécessairement la peine pour une grille 9x9, car la quantité de calcul parallèle n’absorbe pas les coûts de changement de contexte. De plus, la difficulté d’implantation de l’algorithme parallèle est énorme, comparer au gain de temps gagné sur l’algorithme séquentiel qui est plus simple. La complexité de l’algorithme parallèle ne nous a donc pas permis de gérer des grilles plus grandes. Il serait intéressant dans un projet de comparer les temps pour de telles grilles ou encore implanter un algorithme de génération de sudoku en parallèle.

1. <http://www.codeproject.com/Articles/34403/Sudoku-as-a-CSP> [↑](#footnote-ref-1)