Université Paris 8

Machines parallèles : Puissance 4

Panchalingamoorthy Gajenthran
Hu Sacha

8 avril 2019

Table des matières

1 Introduction

Le projet que nous avons choisi est de résoudre un jeu de puissance 4, dont les dimensions peuvent varier, à l'aide de l'algorithme minimax et de la coupure alpha/beta puis, de paralléliser le processus avec des threads. Tout d'abord, définissons les règles de base du puissance 4. Le jeu de puissance 4 se joue sur une grille de 7 colonnes et 6 lignes. Lorsque l'on joue un coup, on lâche la pièce dans une colonne et celle-ci tombe jusqu'à ce qu'elle rencontre une pièce ou qu'elle touche le fond. Le but du jeu est d'aligner 4 pièces soit verticalement, soit horizontalement, soit en diagonale. Le premier à aligner 4 pièces remporte la partie. Si aucun des deux jours ne gagne, il y a égalité. Le jeu de puissance 4 en taille 7 * 6 est un jeu résolu dans lequel le premier joueur, s'il joue de manière optimale, remporte nécessairement la partie.

2 Les débuts du projets

Avant de commencer à résoudre le jeu, il nous a fallu implémenter un jeu de puissance 4. Etant donné le nombre de calculs qu'il faut faire pour résoudre le puissance 4, nous avons vite compris qu'il fallait réduire au maximum la mémoire occupée par la grille du puissance 4. Nous avons donc dans un premier temps représenté les pièces par des bytes. Nous avons également codé les fonctions permettant de vérifier à chaque instant si la partie est finie et renvoyant le gagnant, 1 si le premier joueur gagne, -1 s'il perd et 0 en cas d'égalité.

3 La résolution du jeu

Résoudre le jeu de puissance 4 est une tâche fastidieuse. Résoudre signifie déterminer exactement le gagnant dans un étant donné, en supposant qu'il joue de manière optimale, ainsi que le premier coup qu'il doit jouer. Pour ce faire, étant donné qu'il faut déterminer une solution exacte, il est nécessaire de parcourir tous les états possibles du jeu. Autrement dit, un algorithme de recherche en profondeur comme minimax peut fonctionner. L'inconvénient est qu'étant donné le nombre d'états différents possible, l'algorithme de minimax est très lent. En effet, il doit à chaque instant déterminer le meilleur coup pour un joueur, mais également prévoir que l'adversaire jouera son meilleur coup. Pour palier à ce problème, nous avons implémenté une variante : Négamax.
Le jeu de puissance 4 est un jeu à somme nulle, ce qui signifie qu'à un instant t, le meilleur pire coup pour le joueur B est le meilleur coup pour le joueur A. De cette manière, il suffit de constamment déterminer le meilleur coup du joueur A et d'en déduire les coups du joueur B. Malgré cela, le jeu de puissance 4 possède 4,531,985,219,092 états possibles ce qui est bien trop long à traiter dans son intégralité. L'utilisation d'une coupe alpha/beta permet de réduire le nombre d'états à traiter.
Lorsqu'un joueur joue un coup, il crée un nœud dans l'arbre de la partie. Chaque coup rajouté crée un nouveau nœud jusqu'à ce qu'un état final soit atteint, celui-ci est une feuille. La coupe alpha/beta permet de ne pas analyser des branches entières dont le résultat serait nécessairement moins bien qu'une autre branche déjà examinée.
Dans notre implémentation, chaque nœud contient l'état du jeu, le joueur actuel, les nœuds fils, le score du coup joué et le coup en question. Comme dit précédemment, le jeu de puissance 4 est un jeu résolu. Le coup gagnant en début de partie est le coup tout au centre, en colonne 4. De la même manière, les coups dont les suites sont les plus rentables sont les coups du centre. Pour cette raison, notre grille de jeu n'est pas numérotée de droite à gauche ni de gauche à droite mais plutôt du centre vers les bords comme ceci :

```
7 5 3 1 2 4 6
```

Ainsi, un bot fonctionnel doit jouer son premier coup en 1. Ceci permet d'évaluer en priorité les coups du centre et de ne s'attarder sur les autres coups que si cela est nécessaire. En effet, détenir une ou plusieurs pièces au centre peut faciliter grandement la victoire.

4 Utilisation de bitmap

Bien que la structure de bytes était très avantageuse au début, nous avons dû changer de modèle. En effet, nous avons par la suite opté pour un bitmap. Le principe du bitmap est que chaque grille est représentée par un mot de 64 bits, dont chaque bit est la position d'un joueur. Les bits ne pouvant être que des 0 ou des 1, chaque état du jeu possède deux bitmaps, une pour chaque joueur. On peut ensuite utiliser un masque à l'aide du « ou » binaire pour rassembler toutes les pièces des deux grilles, grâce à cela, on crée un bitmap unique. Celle-ci permet par ailleurs de vérifier directement l'égalité entre deux états dans l'arbre de jeu. D'autre-part, en utilisant ce bitmap unique comme clé, on peut rechercher dans une hashmap si cet état existe déjà et donc si ses suites ont déjà été traitées. En effet, il existe une multitude de successions de coups qui mènent tous au même état, si celui-ci a déjà été examiné alors il est inutile de le réexaminer. Cette hashmap permet aussi de traiter les états symétriques dont l'issue est forcément la même.
L'inconvénient de l'utilisation d'un bitmap est que la taille maximale de grille possible est de 8 colonnes et 8 lignes (64 bits). Toutefois, au vu des améliorations en temps observées nous avons tout de même utilisé cette structure. Les améliorations sont dues principalement au fait que toutes les opérations sont faites en binaire.

5 Négamax itératif

Nous avons tenté d'implémenter un Négamax itératif, qui renverrait à chaque instance un score mais cette version ne fonctionnait pas très bien. Après avoir tenté de résoudre les problèmes liés à cette version de Négamax, nous avons décidés de nous reconcentrer sur un Négamax récursif qui est celui utilisé dans la version finale.

6 Parallélisation

Malgré les optimisations réalisées ci-dessus, l'algorithme de Négamax est très gourmand en temps ce qui est un peu frustrant. Afin de réduire le temps d'exécution de celui-ci, nous avons parallélisé son exécution de la manière suivante. Lorsqu'un joueur joue un coup il crée un nœud, toutefois il ne peut jouer que dans l'une des 7 colonnes non remplie. Nous avons donc décider de lancer un thread pour chaque colonne et ainsi de laisser Négamax retourner un score pour cette colonne, puis une fois tous les threads terminés et

toutes les colonnes examinées, comparer les scores de chaque colonne pour déterminer le meilleur coup.

7 Répartition du travail et problèmes rencontrés

Bien que le travail ait été fait dans la quasi-totalité ensemble, on peut discerner une répartition comme ceci. Sacha a réalisé la grille en byte et Gajenthran a implémenté le bitmap. L'algorithme de Négamax a été réalisé ensemble mais nous avons tous deux recherché séparément, des solutions aux problèmes rencontrés en cours de route. La parallèlisation était elle aussi réalisée ensemble.