Parallélisation d'un solveur d'échecs Comparaison d'implémentations HPC

Mathis Caristan & Alexandre Fernandez

5 mai 2017

Résumé

Ce rapport présente la démarche suivie pour paralléliser un code de solveur d'échecs. Trois types d'implémentations ont été réalisées. Une purement MPI (avec la bilbiothèque OpenMPI), une deuxième purement OpenMP, et une troisième utilisant un mélange des deux. Le travail a été découpé en deux blocs distincts, tout d'abord la paralélisation du code « naïf ». Puis L'extension de la paralélisation à une approche plus intelligente du problème qui utilise « l'élagage alpha-bêta ». Les trois implémentations ont été réalisées pour les deux blocs, et les résultats sont comparés ici.

1 Introduction

Nous considérons ici une version simplifiée du jeu d'échecs, dans laquelle on n'utilise que les pions et rois. Nous nous sommes vus fournir le code séquentiel du solveur. Celui-ci présentait deux paramètres d'optimisations, dont l'activation rendait le code plus dur à paralléliser. Le premier était l'utilisation de l'algorithme négamax avec un arbre alpha-bêta, et le second l'utilisation d'une table de transposition. Nous n'avons pas traité le second paramètre d'optimisation. Le programme prend en entrée un état du plateau en notation Fosryth-Edwards. Il cherche ensuite tous les coups possibles jusqu'à un nombre de coups donné. Il joue tous les coups, et analyse quel est le meilleur choix possible pour chaque joueur à chaque coups. Si il ne parvient pas à trouver une fin à la partie, il réitère en jouant un coup supplémentaire jusqu'à trouver une issue à la partie.

Les performances du programme séquentiel sont données dans le tableau 1. Ces valeurs serviront d'étalon pour mesurer les performances des parallélisations.

Optimisation	**Machine(s)**	**Entrée**	**Temps**
« Naïf »	gpu-3	? ? ?	? ? ?
« Naïf »	? ? ?	? ? ?	? ? ?
Alpha-bêta	gpu-3	? ? ?	? ? ?
Alpha-bêta	? ? ?	? ? ?	? ? ?

TABLE 1 – Temps d'éxecution du programme séquentiel pour différents paramètres.