Lecture: Concurrency Theory and Practise
Project: Chess engine
`http://proglang.informatik.uni-freiburg.de/teaching/concurrency/2010ws/`

Project: Chess engine

1 Introduction

From the Wikipedia article on *Chess*:

> Chess is a two-player board game played on a chessboard, a square-checkered board with 64 squares arranged in an eight-by-eight grid. Each player begins the game with sixteen pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. The object of the game is to checkmate the opponent's king, whereby the king is under immediate attack (in "check") and there is no way to remove or defend it from attack on the next move. The game's present form emerged in Europe during the second half of the 15th century, an evolution of an older Indian game, Shatranj. Theoreticians have developed extensive chess strategies and tactics since the game's inception. Computers have been used for many years to create chess-playing programs, and their abilities and insights have contributed significantly to modern chess theory. One, Deep Blue, was the first machine to beat a reigning World Chess Champion when it defeated Garry Kasparov in 1997.

Do not worry if you are not familiar with the rules of chess! You are not asked to change the methods which calculate the moves, nor the internal representation of boards or moves, nor the scoring of boards.

2 Peachess

Peachess is a chess engine written in Java. The implementation consists of several components.

2.1 The chess board

The chess board representation stores the actual state of the game. This includes not only the position of the figures, but also a partial history of the moves and further meta-data such as the score of the board or castle rights.

The chess board implements the IBoard interface which you should use to extract the information that you need. This interface provides the following methods:

- nextMoves() returns a list of all boards that are reachable from the current state by making a single valid move.
- getHash() returns a hash code for the board. You can use it as an index in a hashtable, for example. Beware, hash codes are not necessarily unique! You should use the equals(Object o) method from the Comparable interface to test if boards are equal.
- getScore() yields the scoring of a board such that you can determine which is the best move to make.
- isWhiteNext(), isDraw(), isCheckmate() yield further information on the game state.

2.2 The strategy

Chess is a two-person game with full visibility and no randomization. Therefore, strategies from combinatorial game theory are applicable in order to determine best moves. The IStrategy interface allows to abstract from the actual strategy such that it is possible to compare different strategies easily. The method findNextMove(IBoard b) calculates the board with the next suggested move. It can be obtained by calling getBest(), its predicted is available via getPrediction().

As an example, the provided class AlphaBetaStrategy implements a simple alpha-beta search strategy. Your objective is to implement further strategies and to parallelize them.

2.3 Performance tests

To determine the scalability of your implementation with a growing number of cores available you are supposed to do some benchmarking. There are already some stubs with a selection of boards on which you can test your implementations.

You may adapt and extend any of these tests. It is your responsibility to convince us that your code is correct! Make sure that your results are reproducible.

2.4 XBoard

XBoard is a chess server which also provides graphical output. It allows two players (or chess engines) to play a match against each other.

You do not need to use the xboard interface for the project. However, you may test your chess knowledge by playing against Peachess, or have it play against itself or other engines. We are also planning to have final competitions of your engines at the end of semester.

Instructions on how to start the xboard engine and how the protocol works can be found in the file.

3 The task

During the project you will implement a concurrent version of the alpha-beta game strategy.

You have to submit two things for participation of the program

1. a description of what you did and why you did it, and
2. the implementation of your strategy.

As a first step, familiarize yourself with the alpha-beta search strategy. A good and concise description can be found in the wikipedia article *Alpha-beta pruning*.

To get you started, implement a (single-threaded) version of the alpha-beta strategy which uses some form of memoization for the recursive calls. This means you should store in some map the result of the call in the form of a task object Task(board,alpha,beta,depth).

Use this as the basis to parallelize your search strategy!

Your description should answer the following questions:

1. Identify the task(s) that can get parallelized. What data needs to be shared, what data is thread-local? How do you synchronize the access to shared data?
2. What data structures did you choose for the shared data? Why did you choose it? What would have been alternatives?
3. How does the performance of your engine change when you increase the number of processors? Provide timing measurements for 1,2, and 4 cores and compare them to the sequential baseline! Do not forget to specify the details of your benchmarking (how many cores, how much memory, which machine, how many runs, etc.).
4. What did not work out? Which parallelization effort made the program slower, and why? Did you implement any other optimizations?

Submission: 11.02.11

- Send your solutions via email to `heidegger@informatik.uni-freiburg.de`.
- Each student has to submit his/her own solution. You may discuss solution strategies, but you have to implement your own code!
- The description of your solution is to be submitted as a .pdf file and should be no more then 8 pages.
- The implementation should be a .jar file with your strategies and tests, including all other libs and packages that are necessary to run them. Submissions without source files will get no points. Remember that sources should always be documented!