# 211hw12

## Homework 12: Parallel Sudoku Solver

**Due**: Friday 23 April 2010 at 9:59:59 am

## The assignment

The goal of Homework 12 is to write a **parallel** Java program to solve Sudoku puzzles. We are providing a complete sequential solution to Homework 11 as a starting point, but you are welcome to use your solution if you prefer.

**If you have not submitted Homework 11 as yet, please stop reading now. It is an honor code violation to see the materials for Homework 12 before you have submitted Homework 11.**

Note that method `findSolution()` in `PartialSolution.java` has the task of enumerating different solutions starting from the square located at `(initialI,initialJ)`. It does so by binding the value of this square to each possible value and recurring on an unexplored square until no square has more than one possible value. If the set of possible values for a square becomes empty, then the partial solution with that square is a dead end; it cannot be extended (by binding the unexplored squares to values) to a final solution. When all squares in a partial solution have exactly one value, it is a final solution. In this assignment, you will build a parallel Sudoku puzzle solver by performing the enumeration of possible values for a square in parallel and combining the results of these parallel computations to return the set of all possible solutions to the initial puzzle.

Your assignment is as follows:

**Part 1:** Perform a sequential task decomposition on a solution to Assignment 11, creating a separate `Callable` task (as discussed in Lecture 31) for each possible value for the square located at `(initialI,initialJ)`. Test your code by running findSolution() on the given tests, and devise at least 5 more tests for findSolution(). Then record the execution time output for solving `puzzle1` in `Sudoku.java` using this sequential version.

**Part 2:** Convert the sequential task decomposition from Part (1) into a parallel task decomposition. Each `Callable` task will now be executed in a separate thread, as discussed in Lecture 34. Test your code using the tests in 1). Then record the execution time output for solving `puzzle1` in Sudoku.java using this parallel version. If you run your code on a processor with more than 1 core, you should see some improvement in execution time compared to 1). It may be as small as 10% rather than a factor of 2. Discuss the possible trade-offs as to why the parallel version may not be much faster than the sequential version, or may even be slower in some cases. Make your best effort to create a parallel version with the smallest execution time for puzzle1. (Hint: you do not necessarily need to create a parallel thread at each level of the call to findSolution().)

Some Java classes you may find useful are: Thread, Callable, FutureTask.

**The support code provided** contains a DrJava project and a few simple Junit tests. Download the code from here.

## Submission

Submit via Owlspace a `.zip` file containing all the files from the support code including those that you modified. Don't forget to add as header to the `PartialSolution` class, your names and ids. Also include a `README.txt` file summarizing the execution times that you recorded ion parts (1) and (2).