# DAT280 – Lab B: “Sudoku in Erlang”, Resubmission

*Olle Svensson (ollesv@student.chalmers.se)*

*Agazi Berihu (agazi@student.chalmers.se)*

## Attempts

In this version we removed the parallel map altogether and replaced it with the worker pool from the exercise. By switching to a worker pool, we are getting really good granularity control in the sense that we are not using any more processes than we have logical cores, and as soon as a process is done with some work it will be given new work if there is any. By this we are utilizing our cores in a more efficient way and are getting results or sub-results faster than by having hundreds or thousands of processes sharing the cores.

The code utilizing the worker pool is found in the *solve_all* method and works in the following way:

1. If we have a list of sub-problems (partially solved matrices), evaluate the second element in parallel and evaluate the first element in the current process.
2. If the first element didn't give a solution, check the result of the second element. If neither of them returned solutions, continue with the rest of the list.

We found this approach with a more depth-first strategy to give slightly better performance compared to a more breadth-first strategy where we evaluated the first element in the current process and the rest of the list in parallel.

We also tried to add further granularity control by using *hard* and introducing a threshold value that decided whether a sub-problem should be solved in parallel or not. Basically, if a sub-problem was below the threshold, it was not evaluated in parallel and vice versa. However, we were unable to find a threshold value that increased our performance so this addition was discarded.

# Improved version

The benchmark was done on a Macbook Pro with an Intel i5 with two physical and four logical cores.

**Benchmark original:**

## 1

```
{61530626,
 [{wildcat,0.34176999999999996},
  {diabolical,49.41115},
  {vegard_hanssen,110.95629},
  {challenge,7.76298},
  {challenge1,400.04997},
  {extreme,10.074},
  {seventeen,36.70979}]}
```

## 2

```
{62286634,
 [{wildcat,0.34414},
  {diabolical,49.86796},
```

```
  {vegard_hanssen,110.67564999999999},
  {challenge,7.540970000000001},
  {challenge1,406.64354},
  {extreme,10.375290000000001},
  {seventeen,37.41853}]}
```

## 3

```
{62169325,
 [{wildcat,0.37411},
  {diabolical,49.237269999999995},
  {vegard_hanssen,109.22297},
  {challenge,7.503939999999999},
  {challenge1,407.97423},
  {extreme,9.97407},
  {seventeen,37.40638}]}
```

**Geometric Mean** = cube_root(61530626 *62286634* 62169325)

```
        =  61994636.21109078
```

**Benchmark parallelized:**

## 1

```
{32964905,
 [{wildcat,0.36256},
  {diabolical,26.315810000000003},
  {vegard_hanssen,64.80456},
  {challenge,4.752680000000001},
  {challenge1,182.14541},
  {extreme,16.75684},
  {seventeen,34.51041}]}
```

## 2

```
{32618242,
 [{wildcat,0.37456},
  {diabolical,26.14618},
  {vegard_hanssen,64.13234},
  {challenge,4.80504},
  {challenge1,179.15288},
  {extreme,17.29643},
  {seventeen,34.27467}]}
```

## 3

```
{32276927,
 [{wildcat,0.35885},
  {diabolical,24.72998},
  {vegard_hanssen,64.02642},
  {challenge,4.795229999999999},
  {challenge1,180.31195000000002},
  {extreme,15.927610000000001},
  {seventeen,32.618919999999996}]}
```

**Geometric Mean** = cube_root(32964905 *32618242* 32276927)

```
         =  32618815.504131433
```

**Improvement in %** = ((61994636.21109078 - 32618815.504131433) * 100 )/ 61994636.21109078

```
         = 47.40 %
```