CHALMERS UNIVERSITY OF TECHNOLOGY

DAT280 Parallel Functional Programming Lab B

Johan Swetzén, 890916-5576
swetzen@student.chalmers.se
Sebastian Lagerman, 890608-5017
seblag@student.chalmers.se

April 25, 2014

BACKGROUND

We started of by parallelizing the benchmarks() function which benchmarks each sudoku sequentially. This was done with ease and we had a speed up of 2.46 times faster for a four core which was less then we were expecting. With a dual core then we had a 1.72 speed up which at least is decent in our opinion. Just for fun we decided to add the Inkala sudoku which is supposedly very difficult. When it came to pick one of the sudokus to focus our attention on then we decided on challenge1. The reason for choosing this sudoku was that it took the longest to solve sequentially, as seen in table 1.1. We thought this one had the most potential to be improved.

ROWS

As asked for in the lab assignment then we started off by parallelizing the refining of each row. We found that none of the sudokus got any speed-up at all. This we attribute to the fact that refining a row is very quick, so the overhead of spawning a process is probably greater than performing the actual work.

	Org	Row	Speedup	R C&B	Speedup	Guess	Speedup
Wildcat	0.3	12	0.025	1.3	0.2307692308	0.3	1
Diabolical	41.3	1737.5	0.02376978417	125.5	0.3290836653	17.6	2.346590909
Vegard_hanssen	88.8	4323.2	0.02054034049	263.3	0.3372578807	53.8	1.650557621
Challenge	6.1	321.1	0.01899719713	18.1	0.3370165746	3.9	1.564102564
Challenge1	329.8	26617.7	0.0123902516	969.6	0.340140264	105.2	3.134980989
Extreme	8.5	882.1	0.009636095681	23.2	0.3663793103	14.4	0.5902777778
Seventeen	30.6	3327.7	0.009195540463	90.6	0.3377483444	226.7	0.13498015
Inkala	263	34356.5	0.007655028888	742.7	0.3541133701	159.5	1.648902821
Geometric Mean			0.007916591728		0.2781013915		1.139533476

Table 1.1: These reading were taken on a quad core machine.

ROWS, COLUMNS AND BLOCKS

Parallelizing the refining of rows, columns and blocks was more challenging than just splitting up the rows into different processes. At first, we got it working with a very naive join function for merging the three results into one. This was not much better than parallel row refining. However, better join and intersect functions made it far better than our first attempt. Unfortunately this didn't produce better results for any of the sudokus either, but about a factor 3 slowdown for all of them.
This could have a connection with the number of processes running, which is 3 a lot of the time (see table A.2). This would mean that the process handling is a lot more taxing on the CPU than actually performing the refining. Once again, the chunks are too small, so refining a whole sudoku is simply too simple.

GUESSES

This parallelization explores different guesses in parallel, so the solve_one function has been adjusted. When we started to implement this we were hopeful, since the amount of work passed to each process had the potential for being larger than the ones we tried before. It turned out we were right, and we got our first speedup. We got the best result for the Challenge1 puzzle, which was the one we focused on, and almost all of them were at least a bit faster. This can be seen in table 1.1. Two of them, however, Extreme and Seventeen, were as much as ten times slower. Here, Row, Column and Block parallelization was better event though it was still worse than the original.
Our maximum speedup of 3.13 for one of the sudokus seems pretty good on a quad core processor, but the slowdown for the two bad ones weighs the geometric mean down quite a bit. In an attempt to mitigate this, we tried to take the hard() value into account and use the sequential version if the problem had a hard() value of less than 100. While this did improve results for the two slow puzzles, we had hardly any speedup for the rest of them so it was not worth it. With more effort, maybe we could have found a good trade-off by experimenting with different values for hard() or using some other measurement for deciding if we should

try to solve a problem in parallel at all.
An interesting observation is the the two puzzles that are slow using the parallel guess exploration are ones that are pretty quick sequentially. So in some sense we are having problem dealing with the simple problems. One possible reason for this is that we have to explore every guess for a longer time before we can discard it. Solving Seventeen did actually require less guesses than solving Challenge1, so the latter would seem like more work unless each guess is more quickly dismissed.

A. GRAPHS

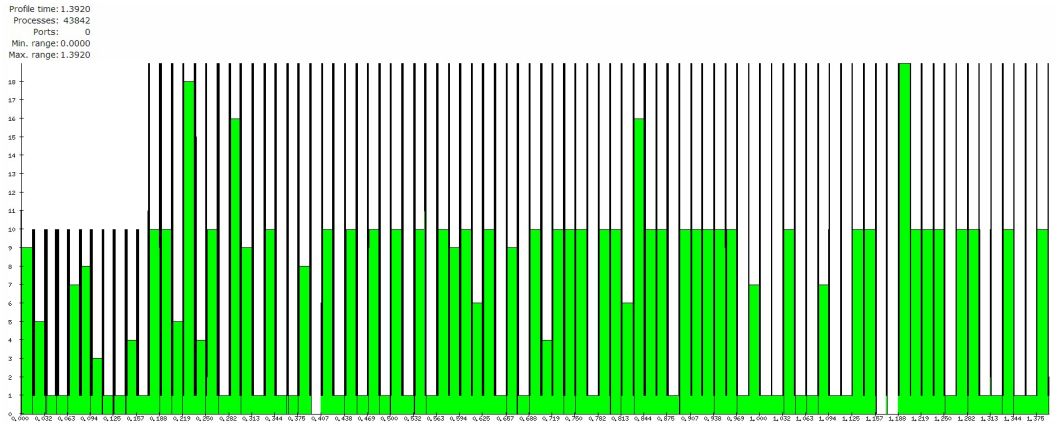

Figure A.1: Challenge1 Row Graph. There is continuous fluctuation because the processes are so short lived. Around 44 000 processes are created during one run.

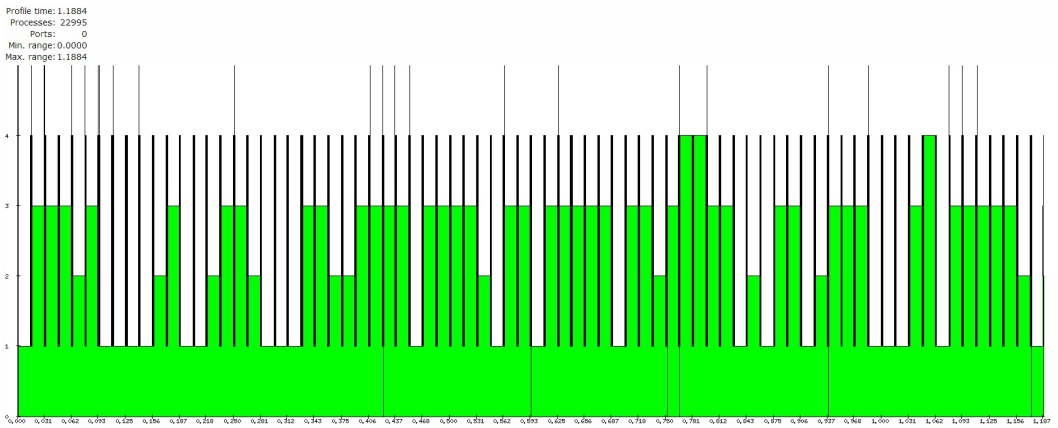

Figure A.2: Challenge1 Row, Column & Block Graph. There is parallel work most of the time, but far too often there is no parallelism available and almost never for all four cores. Approximately 23 000 processes are created during one run.

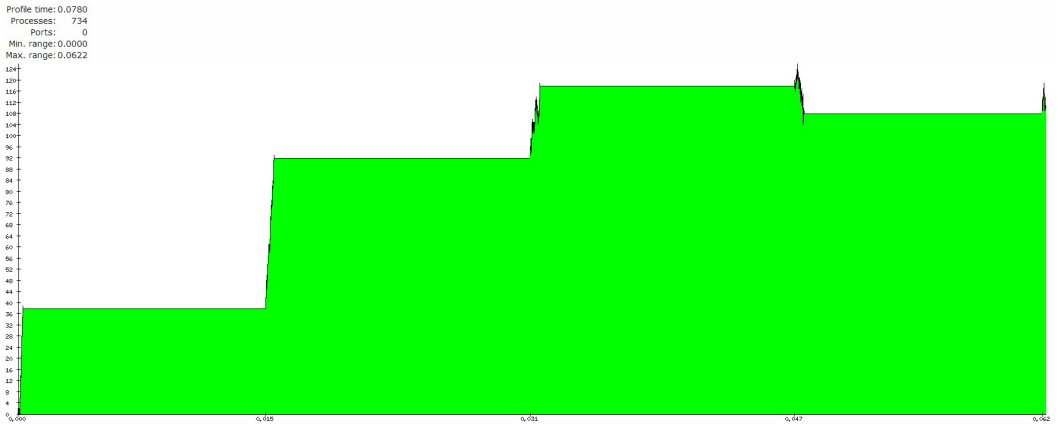

Figure A.3: Challenge1 Guess Graph. We have parallelism all the time, and there are plenty of processes for all the four cores. Almost 750 processes were spawned during one run.

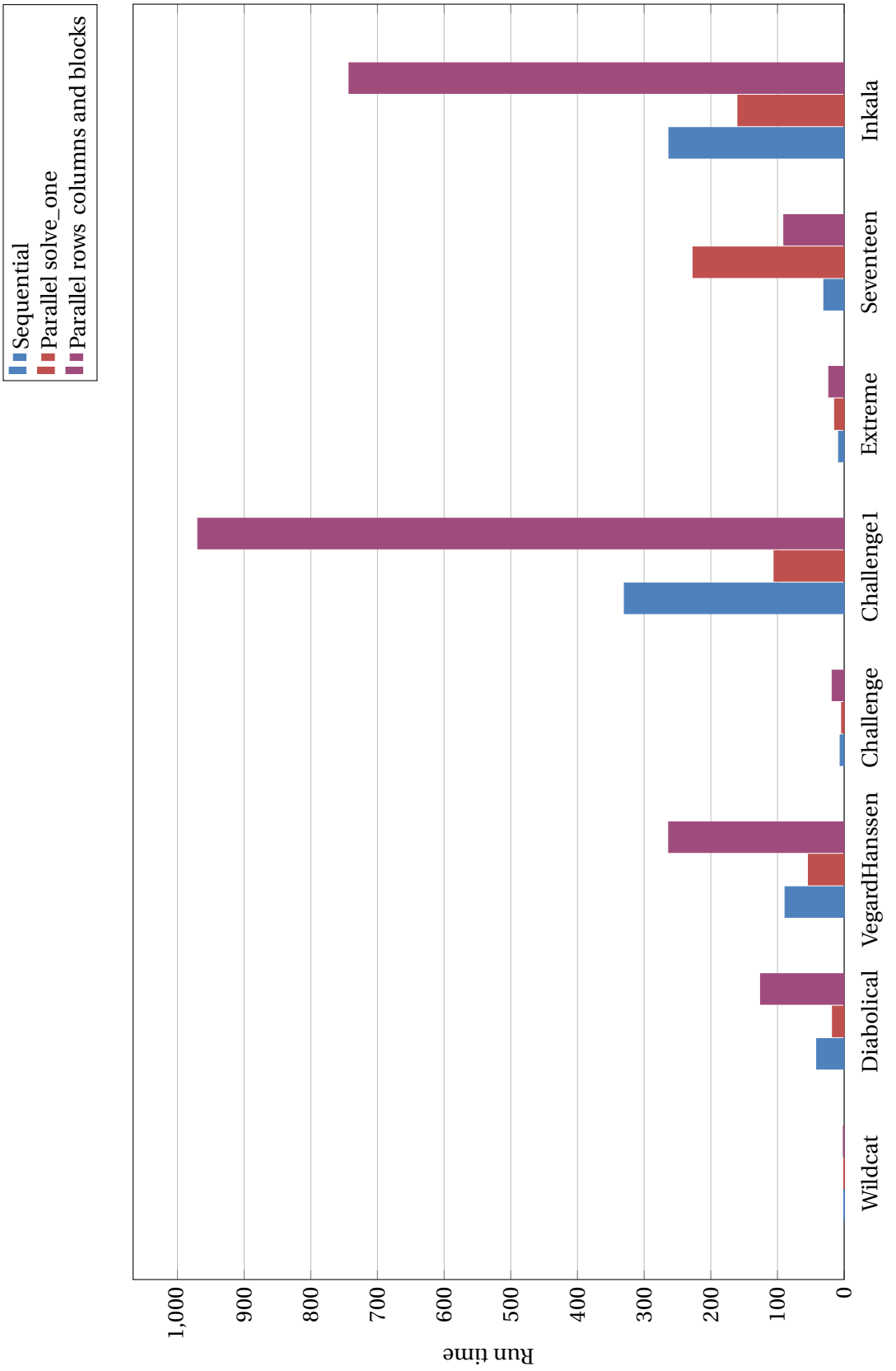

Figure A.4: This is our speed-up graph with side by side comparison though we had to remove the row speed-up values otherwise you wouldn't be able to see anything except how bad row was.