Assignment 3: Sudoku Orbits

Justin Miller

March 26, 2012

Purpose

We want to see how many orbits there are for a given puzzle. On top of this we are going to try and use openmp Sections and For to help speed it up.

Running

The command that I use to run it is ./sudoku $(cat input.txt) > output.txt &.

There is MAX_THREADS for max threads of the computer and can be set by hand.

There is PARR to decide to run in unique finding mode (0) or parallel sections (1)

Calculated Amount

We find that for each type of change to a puzzle there are

- Relabel: $9! = 362880$
- Row Permutation: $3!^3 = 216$
- Band Permutation: $3! = 6$
- Column Permutations: $3!^3 = 216$
- Pillar Permutations: $3! = 6$
- Rotation Permutations: 4
- Reflection Permutations: 2

Now we see that the combination of row, band, column, pillar, rotation, and reflection means that rotations = 2 since if we rotate once more we have just reflected and we see that reflections are not needed since row and band do the same. Thus we find that

- Row+Band+Column+Pillar+Rotation Permutations: $3!*3!^3*3!*3!^3*2 = 3!^8 * 2 = 3,359,232$

Now we see that if we try and add in relabel we get

- $3,359,232 * 9! = 1,218,998,108,160$

Thus I decided that I will not do that instead I will just add that relabel and rotation thus I get

- $3,359,232 + 9! * 2 = 4,084,992$

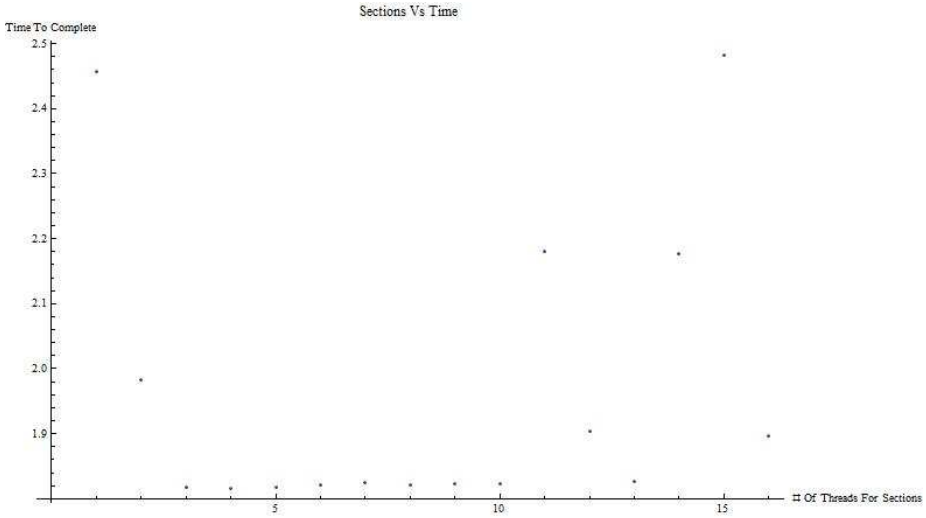
Computer Found Amount

Doing the same thing as the calculated but finding the permutations and the puzzle that it has found it finds that there are 4,084,992 solutions. We see that the solutions match, which it should.

Time Improvements

Section

The first thing that we are going to try and find the improvements by putting the code of permutations and relabeling in different sections. Here we get the chart

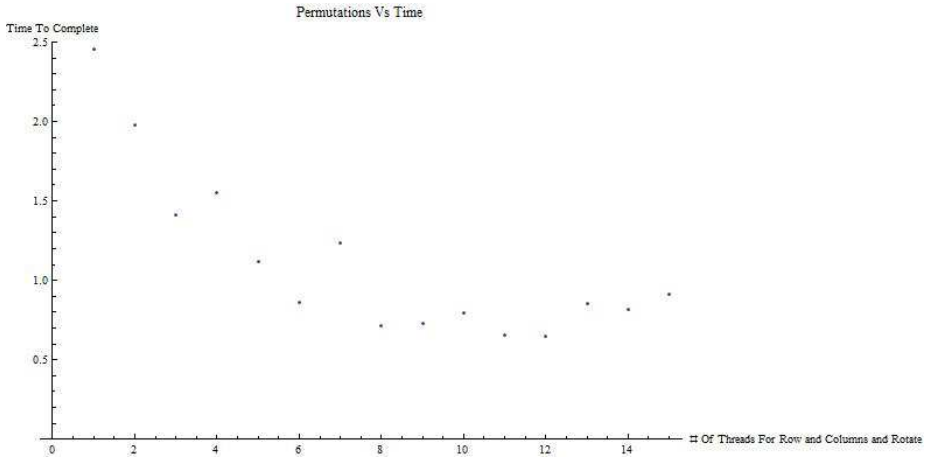


What we see is that it does improve performance but only to 3 threads, after that the computer doesn't get any improvements and later gets big overhead.

Finding the best is at threads = 4, and the time is 1.815079 seconds at an efficiency of 0.7385091553 or 1.35 times faster.

For: Permutations

The next thing that we want to see is the improvements on using a openmp For parallel for the permutations section.



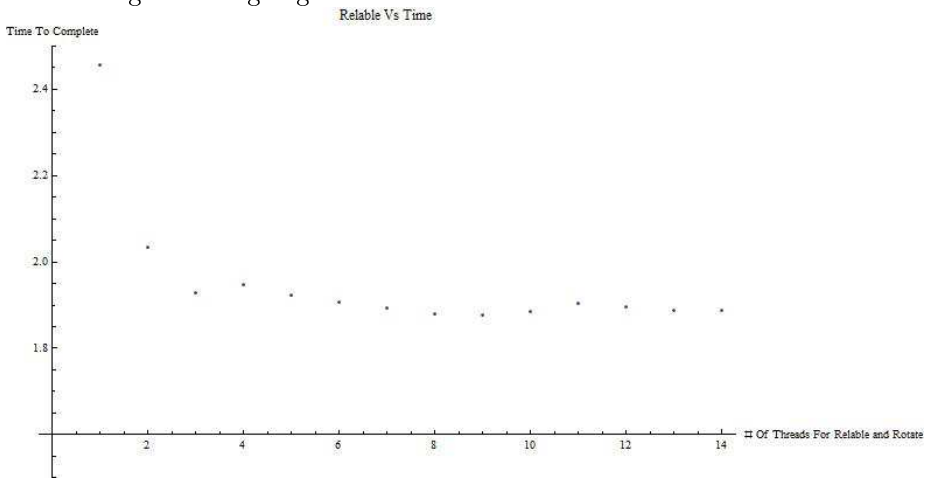
And here we see that it does improve.

Finding the best is at threads = 12, and the time is 0.648182 seconds at an efficiency of .2637286538 or 3.79 times faster.

This is so far the best improvement.

For: Relabel

The next thing that we going to find is the best number of threads for relabel.



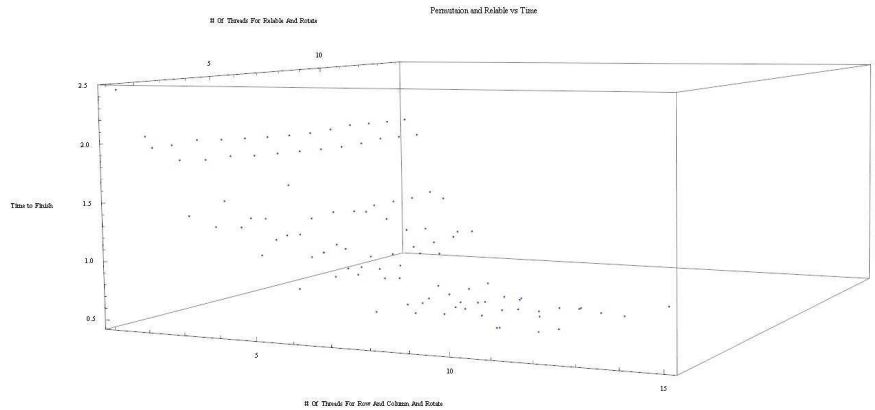
And it also improves.

Finding the best is at threads = 9, and the time is 1.875766 seconds at an efficiency of 0.7632011412 or 1.31 times faster.

This is the worst optimization so far.

Sections: For: Relabel + For: Permutations

Now we are going to see what the best combination of all three is. For the graph we did just For of Relabel and For: Permutations.



And the best combination that we can find is 1 thread for Sections, 11 Threads for Permutations, and 4 Threads for Relabeling for a time of 0.458857 or a 5.356267857 speedup knowing from Amdahl's law that we see that $\frac{\frac{1}{5.356267857}-1}{\frac{1}{14}-1} * 100 = 87.58645963\%$ of the code is paralleled.

Finding Unique Solutions

Memory

To store a Sudoku, we can store it in 6 32-bit integers. There are 9 blocks total. If we start with the upper left being 1 and the next one to it's right is 2 and so on. 4 for example, is going to be the middle left. The blocks that we need to store are 2,3,4,6,7,8 leaving a diagonal through the middle. Then for each block we just record the first 8. Thus we can reconstruct any puzzle from these 6 32-bit numbers. And assuming that the memory is 8 bytes for an integer than the total memory for storing all puzzles is bounded by $8 * 8 * 4,084,992 = 261,435,008 Bytes$. Though this will slow down finding all the solutions now because it will have to run through (by the end) 4 million other puzzles to check if it is unique.

Programming

For the program to change modes there is a #define PARR that if PARR == 1 then do regular, but if PARR == 0 then do the unique finding. We have also taken out the parallel sections and for because when we check on the data it is going to go through a bottle neck of checking thus reducing all that parallel useless. But we will add in one parallel for in the check unique function so it can check faster.

Results

What Next

- For next time we can do a true combination of everything.
- Use more cores
- Average of runs