Parallel Nonogram Solver

Benjamin Huang (zemingbh) and Eric Sun (ehsun)

Summary

This project aims to implement a fast parallel nonogram solver that scales to large puzzles and scales for puzzles with high ambiguity. We will also consider the design of data structures tha facilitate parallelism for such an application.

Background

Nonograms

Nonograms are a logic puzzle built around a rectangular grid divided into cells. The solver's objective is to determine, for each cell, whether the cell is shaded or not. The constraints on the shading of cells are given as number sequences on each row and column, indicating to the solver the number of contiguous regions of shaded cells in that line (row or column) and the number of shaded cells in each contiguous region. See https://en.wikipedia.org/wiki/Nonogram for more details.

Generally, given a single row and column, a solver would be able to determine the result in some of the cells. For a simple example, given such a row on a 16 column board:

```
10 2 2 |.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|
```

This indicates that there will be a contiguous region of 10 shaded cells, then 2 shaded cells, then 2 shaded cells. Since there is at least one unshaded cell between each contiguous region, we can determine the following cells are shaded and not shaded:

```
10 2 2 |X|X|X|X|X|X|X|X|X| |X|X| |X|X|
```

Suppose we have the following row instead:

```
10 1 1 |.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|
```

We will not be able to determine the entire row, but only the following cells:

```
10 1 1 |.|.|X|X|X|X|X|X|.|.|.|.|.|.|.|.|
```

This can be derived since the last two single shaded cells must have two unshaded cells before them, which restricts the first contiguous 10-region to within the first 12 cells. Thus, the 3rd to the 10th cell in the row will definitely be shaded as part of the 10-region. Howveer, we cannot make any other conclusions about the rest of the cells in the row without looking at the rest of the puzzle.

Another example row is given below:

```
2 4 |.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|
```

We cannot determine any shading by looking at this row in isolation, since the possible positions of both the 2- and 4-region do not all interesct at any cells. However, if we determine from other parts of the puzzle that the 3rd cell is shaded,

```
2 4 |.|.|X|.|.|.|.|.|.|.|.|.|.|.|.|
```

we can also determine that the first cell is unshaded. This is because if the first cell was shaded our 2-region would touch the 3rd cell and not be valid.

```
2 4 | |.|X|.|.|.|.|.|.|.|.|.|.|.|.|
```

In some cases the entire puzzle will not have any rows and columns that can be solved from the current state, and the solver is required to proceed via an assumption. Since puzzles have a single solution, the solver either arrives at a contradiction in which case the initial assumption is determined to be false, or the solver solves the puzzle (possibly via further assumptions). A state where an assumption is required to proceed is said to be an *ambiguous* state. Generally, the more ambiguous states a puzzle has, the more difficult it is to solve.

The Challenge

Nonograms are interesting because they are not a particularly mainstream type of logic puzzle, which means existing research is scarcer, and are not obviously reducible to other problems. There are also much fewer existing implementations of solvers.

With regard to parallelism, it is likely that three areas of the problem will be parallelized: **simple solving**, **lookahead solving**, and **heuristics**.

Simple solving

In simple solving, we have a single shared state. Since each line can be solved with only the data in the line, we can parallelize this by line with some form of synchronization between rows and columns (which share data).

Lookahead solving

In lookahead solving, the solver essentially explores a tree of possible future states based on an assumptions made in the current state. Simple solving is required when traversing to a future state. This provides two axes of parallelism: simple solving within each branch and testing different assumptions in parallel. Since exploring different assumptions will likely lead to contradictions in drastically varying amounts of time, some sort of dynamic scheduling will be required. It is also likely that different branches of the tree merge at future times since different assumptions can lead to the same state, and efficient communication between threads will be required to determine if and when this happens in order to avoid redundant work.

Heuristics

In particular, ambiguous states are difficult for solvers because there is very little per-cell heuristic on which assumptions are likely to cause a contradiction or likely to be part of the solution. Ideally, the solver would pick an assumption that will quickly lead to a contradiction to reduce time spent computing states that are not part of the solution. Determining which assumption of all possible assumptions is non-trivial to determine and relies on heuristics.

As such, the challenge will be to find a heuristic that lends itself well to parallelism. Here, we distinguish heuristics from lookahead solving by defining heuristics as computation involving only the current board state and no future board states.

It is likely that heuristics will involve data structures other than a boolean array representation of the board, and these data structures will also have to be optimized for parallel computation.

Resources

There have been two past projects on Nonograms and a number of past projects on other logic puzzles (mainly Sudokus). https://github.com/seansxiao/nonogram-solver (S17, using CPU, only naive heuristics) http://www.andrew.cmu.edu/user/marjorie/parallelpbn/parallelpbn.html (S15, OpenMP, poor parallel speedup)

Goals / Deliverables

Solver implementation

Low-ambiguity puzzles

Fast (good constants)

Scale well with size (linear/linearithmic)

Some speedup with number of processors for larger puzzles

High-ambiguity puzzles

Very good speedup with numner of processors

Some scaling with size

Theoretical

Parallelizable heuristics with theoretical foundation

Demo

Show graphical animation of puzzle being solved (and the steps taken by the solver) on varying puzzles

Show speedup graphs

Stretch goals

Extend to Nonogram variants (additional colors, unordered constraints)

Extend to other logic puzzles (Slitherlink, etc.)

Platform

Machine

Using a GPU or a heterogenous configuration is suitable because the problem inherently has many small parts (many cells, many rows, many columns). Therefore, puzzles can be divided up into many small semi-independent problems with shared memory (by line or by cell). However, there is unlikely to be much use for SIMD parallelism. As such, it may be useful to try it on a processor like a Xeon Phi which has a large number of independent processors but better performance with divergent execution.

Languages

C++: Fast, with good library support. CUDA: for GPU utilization

Schedule

12 Nov: Working solver

19 Nov: Working parallel solver with significant speedups on lookahead solving

26 Nov: Derive heuristics

3 Dec: Implement heuristics

10 Dec: Optimize for parallelism, take benchmarks

15 Dec: Final report due