Parallel Sudoku solver

Graphic Processors in Computational Applications

Daria Hubernatorova

1. Introduction

The aim of the project is to use GPU to implement efficient solution of Sudoku puzzle.
Sudoku - logic-based, number placement puzzle.
The general problem of solving Sudoku puzzles on n^2×n^2 grids of n×n blocks is known to be NP-complete. Many computer algorithms, such as backtracking and dancing links can solve most 9×9 puzzles efficiently, but increase in n is a combinatorial problem which makes the whole problem as NP-complete.

The aim of the standard size game is to place numbers from 1 to 9 in rows, columns, sub-boards 3x3 without repetitions.

2. Recursive CPU algorithm

A common algorithm to solve Sudoku boards is called backtracking. This algorithm is essentially a depth first search in the tree of all possible guesses in the empty space of the Sudoku board. The algorithm finds the first open space, and attempts to put number there. If that board is valid, it will continue trying values in the other open spaces. If the board is ever invalid, backtrack by undoing the most recent attempted value and try another value there. If all values have been tried, backtrack again. If the board is valid and there are no more empty spaces, the board has been solved! If the backtracking goes back to the first empty space and there are no more possible values, we have tried every possible combination of numbers on the board and there is no solution. Clearly, this uses recursion to try all possible values.

Making parallel seems not possible, because recursion uses the stack and access it in parallel is hard in general.

Parallelization using BFS

Finally, we can modify the depth first search approach so that threads can function independently without all reading from the same stack (no recursion).

Let’s imagine tree with several levels, where firstly we fill some number of empty spaces in the board, generating several boards and then passing them to next level.

On each thread on level backtracking is applied. If any thread finds a solution, all threads stop, and that solution is passed back to the host.

Hence, BFS is parallelizable instead of DFS.

3. Parallel Backtracking

The parallel approach to backtracking can be broken down into two main steps:

1. BFS from the beginning board to find all possible boards with the first empty spaces filled. This will return possible starting boards for the next step of the algorithm.
2. Attempt to solve each of these boards separately in different threads on the GPU. If a solution is found, terminate the program and return the solution.

The speedup of this approach is in the parallelization. This allows us to search through the depth first search approach in parallel, which allows for great speedup.

Board Generation Kernel

This kernel will expect the following things as input:

- the previous frontier of boards from which we will be searching
- a pointer to where the next layer of boards will be stored
- total number of boards previously
- a pointer to where the indices of the empty spaces in the next layer of boards will be stored
- a pointer to where the number of empty spaces in each board is stored

The kernel will spawn threads where each one generates the children of the board. This can be done without knowing the actual graph, because the children board of each input is simply the possible boards given the first empty space in the old board.

Backtracking Kernel

This kernel will expect the following things as input:

- all the boards to search through
- number of boards to search through
- location of the empty spaces
- number of empty spaces

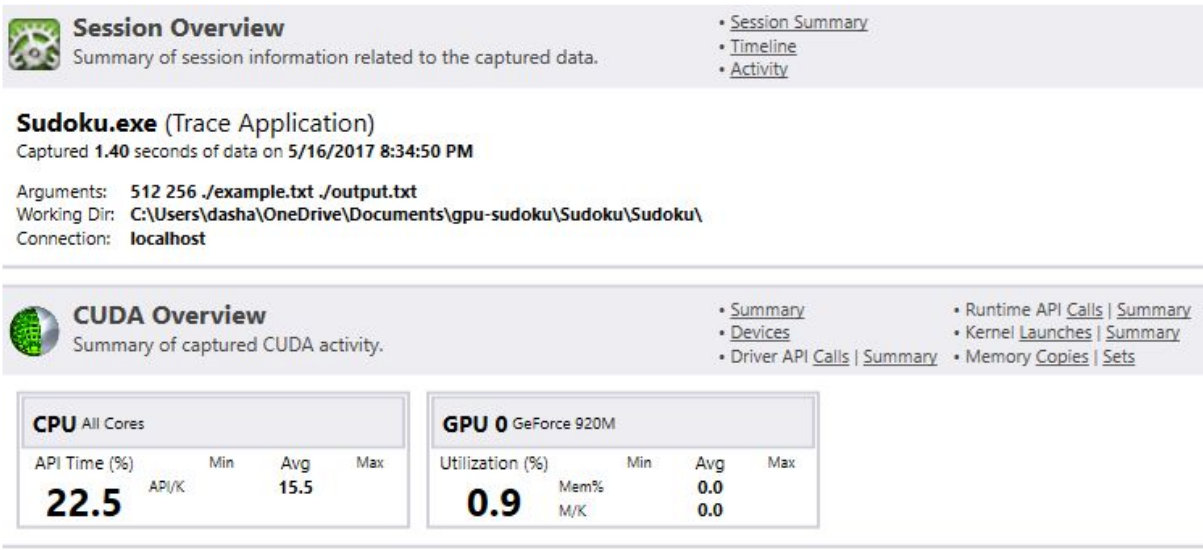
On each thread classic backtracking algorithm is then done.

The stack for backtracking is explicit given empty spaces locations.

When solution is found, all threads are notified by global flag.

Results

Comparing the execution of CPU and GPU solution, the GPU solution is almost 20 times faster. The CPU time ~22 seconds, GPU is ~ 1 second.



How to run

Provide command line arguments: threads per block, blocks, the input file, the output file.