Optimising Sudoku Solver on GPU

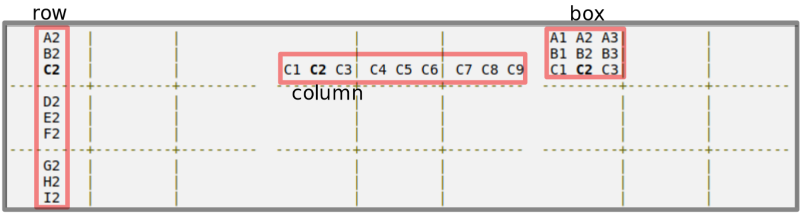
Prasanna Bartakke

CS6023 GPU Programming
Indian Institute of Technology, Madras

Project Presentation

Sudoku

Notation



peers of cell C2

Objective

For a N x N Sudoku, each unit must be filled with a permutation of digits from 1 to N.

Backtracking

Backtracking method on CPU

- Traverse all cells from the top left to the bottom right.
- For each empty cell, try digits from 1 to N. Check if this number is valid given the partially filled board.
- If yes, write the digit on the board, and proceed to the next cell.
- If all digits are invalid, backtrack to the last cell, and continue to try other numbers. Return when all cells are filled.

Backtracking method on GPU

There are two steps in this method.

- Step 1: Initial expansion of search tree on CPU.
- Step 2: Parallel search on GPU: Each thread searches on an independent board which was expanded on the CPU. This is similar to the backtracking method on CPU w.r.t to each thread. All threads execute till a thread finds a solution. Recursion is simulated using a stack in shared memory. It stores the grid location which is getting filled.

This method is just parallelising the backtracking process for different boards. Instead of searching a single board independently on each thread, can we parallelise the time-consuming step of checking the partially filled board?

Optimising backtracking for GPU

Each block of NxN threads works on a single board. Each block has a leader thread which manages the stack and the backtracking process for a board. Other threads parallelise the process of checking if a number is valid at a particular position, given the partially filled board.
At each iteration, the peers of the cell which is at the top of the stack set the proper constraints for the valid values for that cell. Then the leader thread tries only the valid values.
The first step of this method is the initial expansion of the search tree on CPU.
Parallel search on GPU :

- The leader thread fills all empty positions from the board into the stack.
- While any block has not found the answer, the leader thread checks the top of the stack at each step, which has the first empty position. If we reach the end of the board, set the answer.
- Then reset the constraints array.
- The threads that correspond to the current cell's peers, eliminate possibilities from 1 to N. Thus, in the next step, we can iterate over only the correct values, which will ensure the constraints of Sudoku are satisfied.
- Proceed to the next empty cell if a value is not found; else, backtrack, i.e. pop from the top of the stack and try the next possible position for this cell.

Constraint Propagation method

Algorithm

- If a cell has only one possible value, then eliminate that value from the cell's peers.
- If a unit has only one possible place for a value, then put the value there.

As an example of strategy (1) if we assign 7 to A1, yielding 'A1': '7', 'A2':'123456789', ..., we see that A1 has only one value, and thus the 7 can be removed from its peer A2 (and all other peers), giving us 'A1': '7', 'A2': '12345689', ... As an example of strategy (2), if it turns out that none of A3 through A9 has a 3 as a possible value, then the 3 must belong in A2, and we can update to 'A1': '7', 'A2':'3', ... These updates to A2 may cause further updates to its peers, the peers of those peers, and so on. This process is called constraint propagation.

Implementation Details

This is done in parallel on GPU. Each thread is responsible for one cell. A mask of N bits is maintained for each cell, representing the possible values in this cell. This method drastically reduces the possibilities for each cell. After constraint propagation, most of the cells have only 1 value. To get the correct value of the remaining cells, we use backtracking.

Reference: http://norvig.com/sudoku.html

Results

Backtracking

- It takes around 210 s to solve 100 hard sudokus on the CPU.
- The simple backtracking method takes 67 s to solve the 100 hard sudokus on a local GPU.
- The optimised backtracking method for GPU takes 11 s to solve the 100 hard sudokus on a local GPU.

Constraint Propagation

- The constraint propagation + backtracking method takes more than 1500 s to solve the 1 million sudokus on the CPU using the constraint propagation + backtracking method.
- It takes less than a second to solve the 1 million sudokus on GPU.

Datasets

http://magictour.free.fr/top95
https://www.kaggle.com/datasets/bryanpark/sudoku