Homework 3

Ege Aktemur

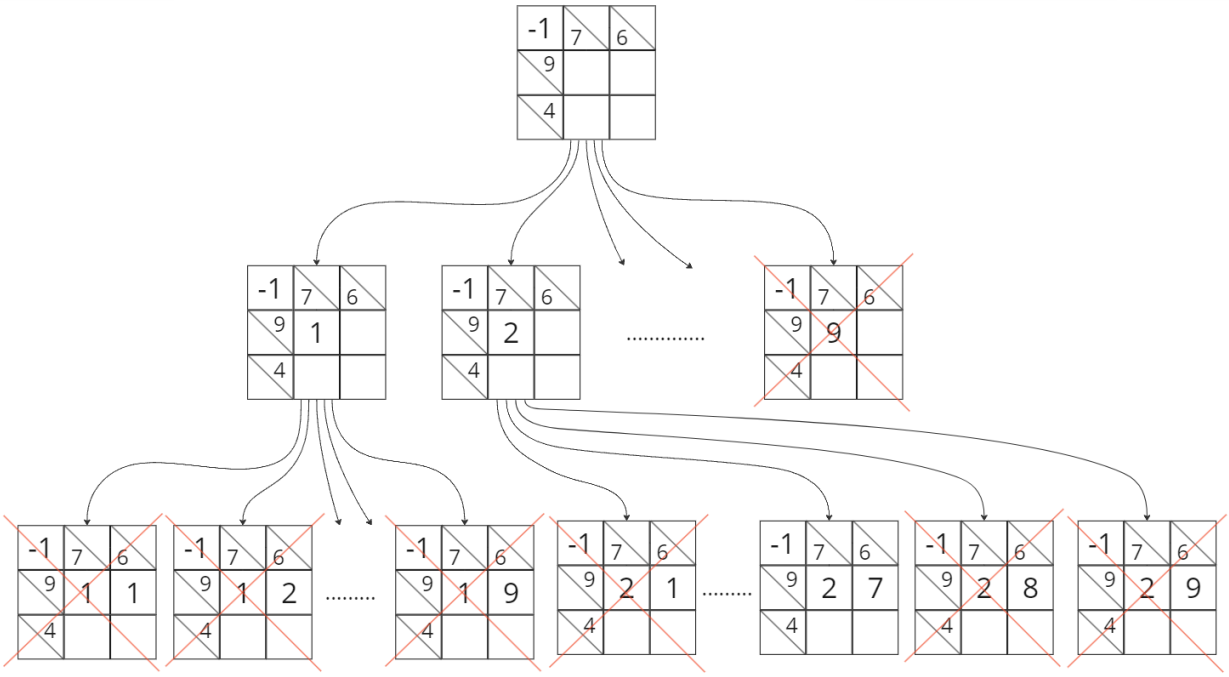
May 2023

1 General Explanation of the Implementation and Used Algorithm

The primary algorithm used in this solver is Backtracking with Pruning. Backtracking is a common technique for solving constraint satisfaction problems such as this, where the goal is to find a solution that meets a series of constraints, and it involves trying out potential solutions and then undoing them if they don't work out (i.e., "backtracking").

Pruning is a technique that involves eliminating branches in the solution space that cannot possibly contain a valid solution. It is used here to reduce the number of potential solutions that need to be examined.

2 Example Tree Formation of Board3_1



3 Code Flow

An initial empty solution matrix (d_sol_mat) is created and placed in the parent variable (parent).

The code then loops over each cell of the matrix. For each cell, it creates 9 child solutions, each representing one of the nine possible values for the cell. These children are stored in the childs array.

The solve_kakuro function is launched as a CUDA kernel to evaluate the validity of each child solution in parallel. It checks each associated sum for the current cell if it is still valid after inserting the new number. If a solution is found to be invalid, it's pruned (set to NULL in childs).

After all child solutions have been evaluated, the countpruned kernel is run to count the number of pruned solutions.

The eliminatepruned kernel is then run to remove the pruned solutions from the list of child solutions, moving the valid solutions to the parent array for the next iteration.

The process repeats until all cells have been filled, at which point all remaining solutions in the parent array are valid solutions to the Kakuro puzzle.

4 Heuristics

The code uses a simple heuristic to speed up the process of finding solutions which is Pruning. By immediately eliminating any child solutions that do not meet the game's rules, the code is able to significantly reduce the number of potential solutions that it needs to check. This greatly reduces the total computational cost of the solver.

5 Efficiency Tricks

5.1 Parallelism

By implementing the algorithm on a GPU using CUDA, the code can evaluate multiple potential solutions simultaneously. This massively parallel approach greatly reduces the overall running time of the solver.

5.2 Memory Management

The code uses CUDA memory management functions to allocate and deallocate memory on the device. Using cudaMallocManaged allows the code to allocate memory that is accessible from both the host and the device, simplifying memory management.

5.3 Reusing Memory

The parent and childs pointers are reused at each level of the recursion, reducing the amount of memory allocation and deallocation required.

5.4 Coalesced Memory Access

The childs array is accessed in a coalesced manner, meaning that the threads in a warp access consecutive memory locations. This leads to efficient memory access and maximum memory bandwidth utilization.

5.5 Pruning

It uses a pruning mechanism to eliminate the wrong solutions from the computation process early (by the function wrongSolution). It eliminates branches that lead to an incorrect solution, reducing the number of possibilities to be checked.

5.6 Concurrency

The code also uses the function cudaDeviceSynchronize to synchronize the execution of threads, ensuring that all preceding commands have completed before proceeding. This is important in situations where the sequence of execution matters.

6 Execution Example

nvcc kakuro_solver_backup.cu -o kakuro.out

./kakuro.out board3_1.kakuro

7 Results

board3_1.kakuro: Number of solutions: 1 Time to generate: 3.2 ms

board3_2.kakuro: Number of solutions: 1 Time to generate: 2.9 ms

board3_3.kakuro: Number of solutions: 1 Time to generate: 2.8 ms

board4_1.kakuro: Number of solutions: 44 Time to generate: 35.2 ms

board4_2.kakuro: Number of solutions: 1 Time to generate: 11.3 ms

board5_1.kakuro: Number of solutions: 12 Time to generate: 70.4 ms

board5_2.kakuro: Number of solutions: 1 Time to generate: 12.8 ms

board20_1.kakuro: Number of solutions: 1 Time to generate: 1031.1 ms

8 Further Improvements

In the version that I am uploading I am allocating memory for successor in parent and copying each parent to child. This method is slow compared to each child copying parent in their execution. However this implementation was giving me crashes in 20_1 so I will upload the first version. (I can sen the better version if needed)