Solving Minesweeper in parallel

Qian-You Zhang
Department of Computer Science, National Yang Ming Chiao Tung University
qianyou.cs11@nycu.edu.tw

Wei-Xin Shih
Department of Computer Science, National Yang Ming Chiao Tung University
s0302.cs11@nycu.edu.tw

Jun-Yu Lai
Department of Computer Science, National Yang Ming Chiao Tung University
loseit7382.cs11@nycu.edu.tw

ACM Reference Format:
Qian-You Zhang, Wei-Xin Shih, and Jun-Yu Lai. 2023. Solving Minesweeper in parallel

1 ABSTRACT

We implement both Pthread and CUDA versions of parallelization mechanism to solve Minesweeper. We use different strategies in Pthread and CUDA for choosing next grid. In pthread, we find that it suffers a lot from synchronization overhead, which overwhelms the gains from parallelism. In CUDA, we scan through every grid in the map in each iteration and attain 1000x speedup while the size of map is (1000,1000).

2 INTRODUCTION

Minesweeper is a well-known puzzle game. Formally, the number of mines N and a map size (m, n) is given. Our goal is to find out all mines in parallel. As you can see in figure 1, the number on the grids denotes the mines surround it. When a grid is considered to be a mine, a flag is placed on the grid. We'll keep finding mines until all N mines in the map is found.

Here's how we find mines and update the map. Denotes $X(a, b)$ as the number of unknown grid surrounds (a, b), $F(a, b)$ is the number of flag surrounds (a, b) and $G(a, b)$ is the number on the grid (a, b). Defines two actions $open - unknown(a, b)$ and $place - flag(a, b)$, which will open the surrounding unknown grids and place the flag on the surrounding unknown grids at position (a, b) respectively. It's easy to infer that when $X(a, b) + F(a, b) == G(a, b)$, we should do action $place - flag(a, b)$. And if $F(a, b) == G(a, b)$, we should do action $open - unknown(a, b)$. Otherwise, we keep this grid (a, b) unchanged.

The challenges of solving minesweeper in parallel is that we cannot simply let each thread keep finding mines without communication. For example, when there's no change in the map in one iteration, we need to do "random select", which can only open one unrevealed grid randomly. As a result, the program will switch frequently between critical section and parallel region, which costs lots of synchronization overhead between them.

Parallel Programming, Jan 2023, Hsinchu,Taiwan
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

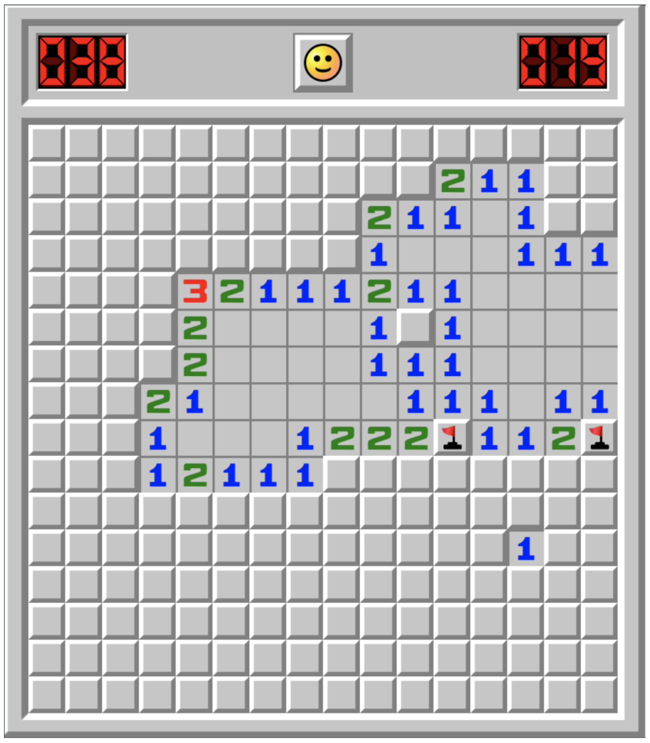
Figure 1: Snapshot of Minesweeper.

3 PROPOSED SOLUTION

In this section, we will describe in detail how we designed our program, including the program flow. We mainly implemented the Pthread method and the CUDA method. In the following, we will introduce the implementation details and some problems encountered in the process.

3.1 Pthread

POSIX Threads, commonly known as pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time. Each flow of work is referred to as a thread, and creation and control over these flows is achieved by making calls to the POSIX Threads API.

For the flowchart of Pthread, you can see figure 2. Before the computation starts, we create threads. At the beginning, a grid will be randomly selected on the entire map. If this grid is not a bomb, you can continue to the next step. But if it is a bomb unfortunately, this time of the minesweeper will fail. The step will only be executed by a specific thread because it is a non-parallel step.

In the second step, detect all mines, it will go through all the grids on the map to determine whether the map can be updated through the existing information, if any grid can be updated, it will do "Open Unknown" or "Set Flag". The workload of this step is shared equally by all threads.

After calculating the complete map, we will first judge whether the map has been updated after the step of detecting all mines, and if so, we will return to the previous step of detecting all mines. If not, it will judge whether the number of bombs we clicked is less than the number of all bombs (n is the bombs we clicked, N is all bombs). If it is false, it means that we have selected all the grids that are bombs, and it will join all the threads. On the contrary, go back to the step of randomly selecting.

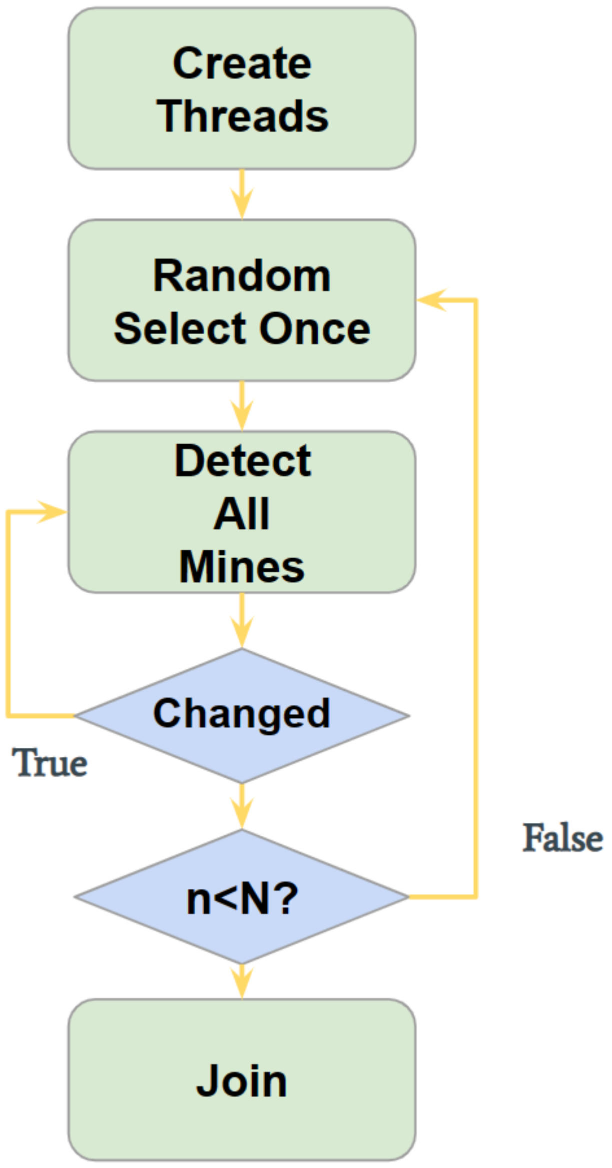

Figure 2: Diagram of Pthread method.

3.2 CUDA

CUDA (Compute Unified Device Architecture) is an integrated technology launched by NVIDIA. With CUDA technology, the GPU can be used for general-purpose processing (not just graphics); this approach is called GPGPU. Unlike a CPU, a GPU parallelizes a large number of threads at a slower rate rather than executing a single thread quickly.

Because of these features we use the CUDA method as our way of parallelizing and shown in figure 2 is a simple flowchart. The flow of the CUDA method is slightly different from that of Pthread.

In figure 2, the green squares represent execution on the host side, while the blue squares represent processing on the GPU. At the beginning, it will judge whether the number of bombs we clicked is less than the number of all bombs (n is the bombs we clicked, N is all bombs). If it is false, it means that we have selected all the grids that are bombs, and do a compare map to confirm whether the answer is correct. On the contrary, do random select to randomly open a grid on the map. Then there is the part of calling CUDA. We will copy some data to the GPU, including the current map as the main map for the game, the truth map as the answer map, and the current bomb count as the total number of bombs we clicked which is shared by all blocks and threads. Moreover, cut all the data to be processed into many blocks, and then distribute the points in the block to different threads.

After all these tasks are finished, it will enter the most important part. All blocks will detect bombs for the grids to be processed. If there is a grid that can open the nearby grid or set it as a bomb, then set the boolean variable "change" to true until all grids have been checked once to determine whether the change is true. If so, recheck all grids. Otherwise, return the current bomb count back to the host. Wait until all blocks are processed and return to the host side, it means that there is no way for all grids to have new actions, so go back to the very beginning to judge whether n is less than N.

The above are some processes and details of our implementation of the CUDA method. More details of tests and experiments will be presented in sections 3 and 4.

4 EXPERIMENTAL METHODOLOGY

In this section, we will introduce the environment and platform our programs test on. Moreover, we also illustrate our test sets and the implementation details.

4.1 Platform and Environment

We test the programs on platform which has GPU GeForce GTX 1060 6GB, Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz processors with 4 cores. Besides, we implement the CUDA parallel method with Nvidia driver which is 510.

Table 1: Execution Time of Serial, Pthread and CUDA.

Method	50x50(200)	100x100 (800)	200x200 (2500)	500x500 (4000)	1000x1000 (7000)
Serial	0.001431	0.00839	0.070506	0.945244	7.36482
2 Thread	0.002797	0.010921	0.078682	1.01815	7.64167
4 Thread	0.003013	0.012759	0.092178	1.09629	8.00158
8 Thread	0.00425	0.014458	0.096284	1.20283	8.06298
CUDA	0.000314	0.000944	0.001627	0.002925	0.006927

Table 2: Speedup of CUDA method.

Method	50x50(200)	100x100 (800)	200x200 (2500)	500x500 (4000)	1000x1000 (7000)
CUDA	4.5573	8.8877	43.3349	323.1603	1063.2048

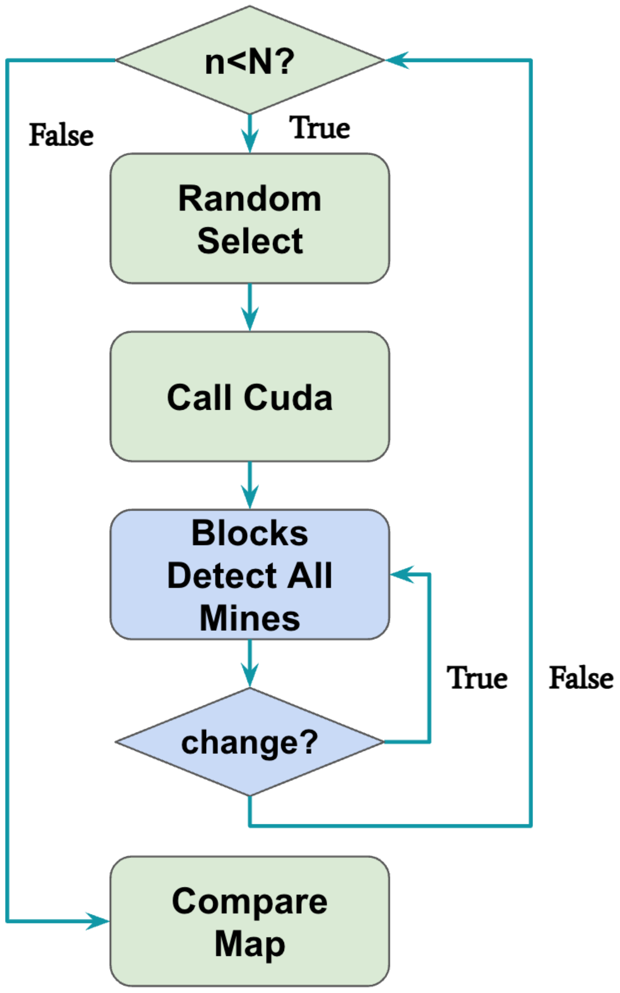

Figure 3: Diagram of CUDA method.

4.2 Input Sets and Arguments

When we design and test our program, we do not use the certain data set. We can decide the size of the map and the number of bombs ourselves.

When executing, the first parameter is rows of map, the second is columns of map and the third is the number of bombs. But in the pthread mode, the third parameter will be added as the number of threads.

4.3 Implementation Details

Our programs are all implemented with C++ language. All functions are the same in the three methods, but pthread method will distribute the work to different threads and CUDA method will copy the data to the GPU and cut it into different blocks. The method is explained in detail in section 5.

In the CUDA method, we set threadsPerBlock to (25, 25) and CHUNK-DIM to 2 (one thread will handle 4 points), so numBlocks will be (numOfCols / (threadsPerBlock.x * CHUNK-DIM) , numOfRows / (threadsPerBlock. y * CHUNK-DIM)).

5 EXPERIMENTAL RESULTS

In this section, we will test three different methods which is mentioned before. We use the execution time of serial method as baseline, compare the time spent by the parallel methods and do analysis.

5.1 Serial method

We use the execution time of serial method as a baseline, the evaluation is shown in Table 1. We tested 5 different map sizes and the number of bombs. There are 200 bombs for map size 50x50, 800 bombs for map size 100x100, 2500 bombs for map size 200x200, 4000 bombs for map size 500x500, and 7000 bombs for map size 1000x1000.

When we tested that the map size 1000x1000 has 7000 bombs, the execution time will come to 7.365 seconds, which is a significant increase from the map size 50x50 with 200 bombs, which also makes it clearer when we observe the results of the parallel method.

5.2 Pthread method

In Pthread method, we test with 2, 4 and 8 threads under same conditions. The evaluation compared with serial method is also shown in Table 1. Through table1, we can find that no matter what the size of the test map is, using Pthread as a parallelization method will take more execution time than the serial method. Moreover, when the number of threads is larger, the execution time will be longer.

We analyzed the design of the Pthread method based on this result. We believe that the pthread barrier wait still needs to wait for threads with multiple workloads because of the inconsistent workload assigned to each thread. In addition, we think synchronization such as mutex takes more time than benefits of parallelization.

5.3 CUDA method

In CUDA method, we test under same conditions and the evaluation compared with serial method is also shown in Table 1. After testing, the CUDA method only takes about 7 milliseconds in a 1000x1000 map, which is much faster than the serial method. Moreover, the speedup table is shown in Table 2 and the visualization is shown in Figure 3, we can see that the speedup can be more than 1000 times when the map size is 1000x1000, and it can also be accelerated by 4.5 times when the map size is 50x50, which is much better than the Pthread method.

We think the reason why the CUDA method is better than the Pthread method is because CUDA is processed on the GPU, and the number of GPU cores is usually thousands or hundreds, and we process the dependency data on the host side, so that It can not only take advantage of the large number of GPU cores but also reduce the overhead of synchronization. Additionally, the speedup increases rapidly as the size of the map increases, because the number of GPU cores is large, so we believe that when the map continues to increase, the speed will continue to increase.

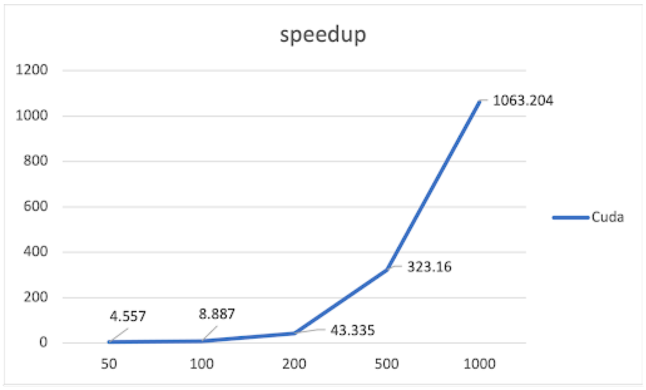

Figure 4: Line Chart of Speedup.

6 RELATED WORK

In the part of writing serial code, we have referred to several minesweeping architectures on the Internet. Basically, the common architectures are similar to what we think. The main difference is the choice of grid when detecting mines.

The first one, such as [1], will go through all the grids on the map in every step of detecting mines. The advantage of this method is that it is simple and does not require too many judgments to maintain, but the disadvantage is that it often checks the grids that have ended or the grids with insufficient information.

The second type, such as [2], will maintain a queue, which will only store useful grids, and will not store those that have been judged or have insufficient information. But the disadvantage is that the overhead is too big

Finally, after implementing two more implementations, we chose to adopt the first method, because it is simpler and faster in parallel implementation.

7 CONCLUSION

To accelerate the serial program, we implement two parallel methods including **Pthread** and **CUDA.**

In Pthread method, we found that synchronization overhead costs much more than the benefits from parallelization.

In CUDA method, we get well performance and have 1000x speedup when map is 1000 x 1000.

8 REFERENCES

Reference sites
[1] https://github.com/meganung/minesweeper-solver
[2] https://github.com/h0rban/minesweeper-online-solver