Testing for connection between opposite edges of small rectangular bit grids in CUDA

István Borsos

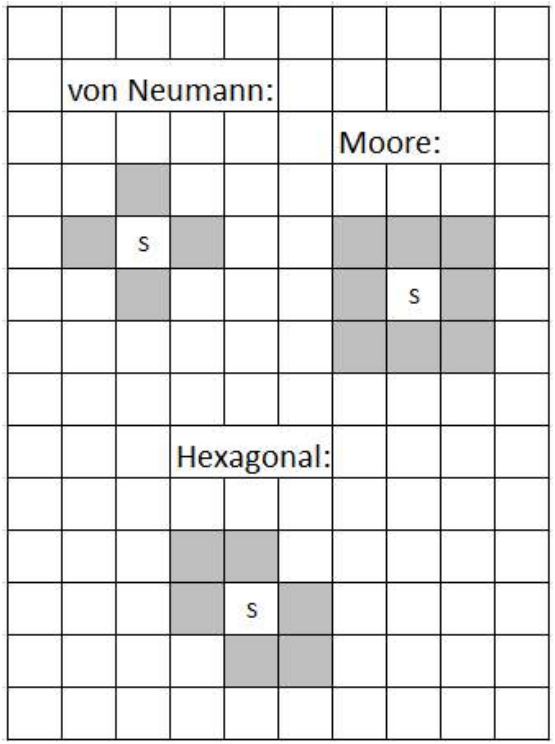
EK-HUN-Ren Centre for Energy Research, MFA, Budapest, Hungary

GPU Day 2024
May 30-31, 2024
Budapest, Hungary

Connection problem

Given a square grid with some sites occupied, the rest is empty. To be decided if there is a path from top to bottom assuming some neighbourhood is defined between the sites.

Some neighbourhood examples



Traditional sequential algorithm

It is a one-pass algorithm with Union-Find data structure (from the "golden age" of sequential algorithms 1965-1985)

- Step 1. Initially the occupied sites are single element subsets of their base set
- Step 2. The sites are visited one by one and if they are connected to some of their neighbours (as deduced by executing two Find operations) the subsets containing them are united by the Union operation.
- Step 3. Once it is found that one of the top row elements is in the same subset as one of the bottom row elements, the connection is found and the algorithm is exited.
- Step 4. Otherwise, if all sites are processed and no such subset is found that means that no connection exists

Weaknesses

- Each site is assigned a unique number in the Union-Find data structure, requiring relatively large storage space
- The Union-Find algorithm has an inherently irregular memory access pattern making it difficult to implement efficiently in SIMD-like machines.
- The elementwise processing has a large time overhead

Bit-parallel algorithm: data

Due to the dense storage and local neighbourhood, bit-parallel algorithms are competitive both in storage space and in execution time even though they perform multiple passes. Furthermore from the CUDA point of view their parallel threads versions are much less divergent.

- We have two bit matrices.
- The rows of these bit matrices are stored in 32-bit unsigned integers
- One matrix, "A", is the input data that remains unchanged, it is used during the search to restrict the reachable sites to the originally occupied sites.
- The other matrix "reached" is used to store intermediate data, it contains those occupied sites that are reached from the top row at a given stage of the processing. Initially it contains only the top row of A.
- The neighbourhood is implicitly handled in the processing

Bit-parallel algorithm: processing

- Top to bottom passes called "sweeps" are executed over the matrices and newly reached sites are booked.
- In each step of a sweep a row is processed, the neighbourhood of each site is checked for possible connection
- These sweeps continue until one of two stopping conditions are met
- Condition 1 No change in the reached sites after executing a sweep: no connection exists
- Condition 2 The bottom row is reached: a connection exists

CUDA code critical part

```
// Test connection
for(int i=0; i<boardsize;i++) reached[i]=0;
reached[0]=A[0];
do {
    changed=0;
    #pragma unroll
    for(int i=1; i<boardsize-1;i++){
        old=reached[i];
        reached[i]=reached[i] | reached[i]<< 1 | reached[i]>> 1 |
                        reached[i-1] | reached[i-1]>>1 |
                        reached[i+1] | reached[i+1]<<1;
        reached[i]&=A[i];
        changed=changed | (old ^ reached[i]);
        };
        reached[last]=reached[last] | reached[last]<< 1 |
                            reached[last]>> 1 |
                            reached[last-1] | reached[last-1]>>1;
        reached[last]&=A[last];
        nsweep++;
    } while (reached[last] == 0 && changed!=0);

    if (reached[last] > 0) nwon++; //connected
```

Average efficiency of the bit-parallel algorithm

- The time consumed by the algorithm is mainly determined by the average number of sweeps.
- In turn, the average number of sweeps is determined by the density of the occupied sites

Example

Grid 32x32 sites, hexagonal neighbourhood

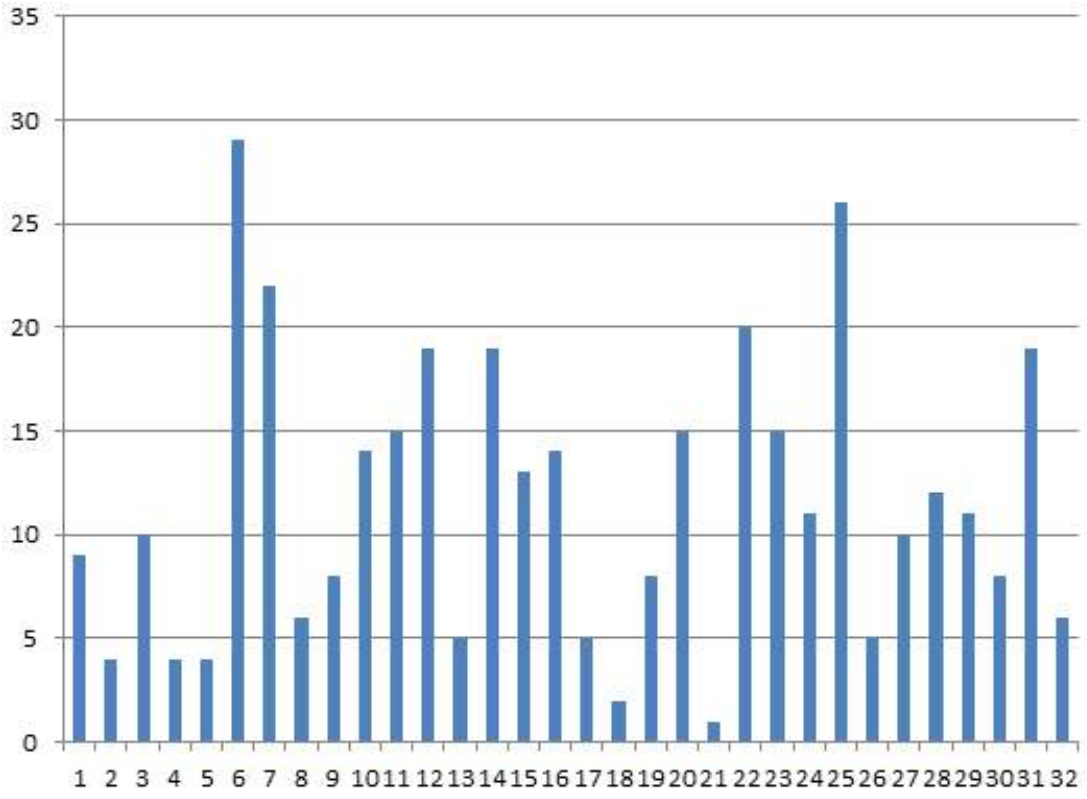
Occupied site density: 0.25
Average number of sweeps: $\sim$ 3 (connection easily refuted because of low density)

Occupied site density: 0.5
Average number of sweeps: $\sim$ 10 (connection neither easily refuted, nor easily found)

Occupied site density: 0.75
Average number of sweeps: $\sim$ 1 (connection easily found because of high density)

Sweep divergence within a warp

There is a loop-divergence in the threads because the number of sweeps depends on the specific site configuration processed by each thread.
An example of the distribution of the number of sweeps in a warp as each thread solves a single connection problem: the average number of sweeps is 11.5, the maximum is 29.



CUDA cycles

GPU used in the test: RTX 4070 Ti Super
Clock: 2340 MHz
Processors: 8448 shaders
Total number of cycles for all processors:
1.98e13 total cycles/sec (19.8 Teracycles/sec)

Test grid: 32x32, hexagonal neighbourhood, p=0.5
Test speed: 1.9e9 total grids/sec (1.9 billion grdis/sec)
Average clock cycles: 10400 cycles/grid
A sweep pass executes about 500 instructions and on the average we execute 10.2 sweeps/grid
So about 5000 instructions are executed for each grid in 10400 cycles, in each thread.
Power consumption 188W (of 285W TDP)

Smaller grid scaling

Grid: 32x32, hexagonal neighbourhood, density=0.5

Speed: **1.9 billion grids/second**
Average number of sweeps: **10.2**

Grid: 16x16, hexagonal neighbourhood, density=0.5

Speed: **8.1 billion grids/second**
Average number of sweeps: **4.6**

Grid: 8x8, hexagonal neighbourhood, density=0.5

Speed: **33.6 billion grids/second**
Average number of sweeps: **2.4**

Reducing thread divergence loss

Simple natural way for a thread program

- for each of N inputs
- generate an input instance
- do
- sweeps
- while connection is not solved

Another way: the same computation is reordered in another conditional structure:

- do
- sweeps
- if an input instance is solved,
- generate a new input instance for the given thread
- replace the old instance in the bit matrices
- while there are inputs to be processed by the thread

Reducing thread divergence loss

- The loop-divergence is passed from the sweeps (which now always work in lockstep for the life of the threads) to the if-divergence of the input generation (which now may diverge).
- But if input generation is faster than the average time to solve an instance or the distribution of the divergence is better, we can expect some speedup
- A some loop-divergence remains in the finishing "while" (not all threads execute the same number of number of total sweeps), but if N is large, the Law of Large Numbers takes care of it.
- For example, for N=10000, this divergence is only about 1 percent.

Measured speed comparison

- Grid: 32x32, hexagonal neighbourhood, density=0.5
- GPU used in the test: RTX 4070 Ti Super
- Natural way: 1.9 billion grids/sec
- Reordered way: 2.9 billion grids/sec

Scaling for other CUDA GPUs

As the program in the critical parts uses almost exclusively registers, the **speed is expected to scale according to the raw computing speed** of the various cards

Conclusion

- A high speed of connection testing is achievable in CUDA for small grids with constant sized bit matrices.
- Thread divergence loss can be reduced by a simple reordering of the computation, resulting in meaningful speedup

Thank you for your attention.