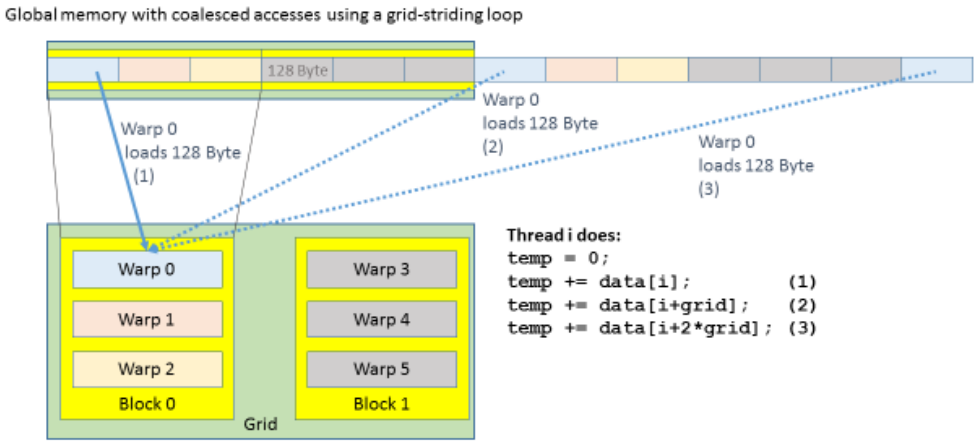
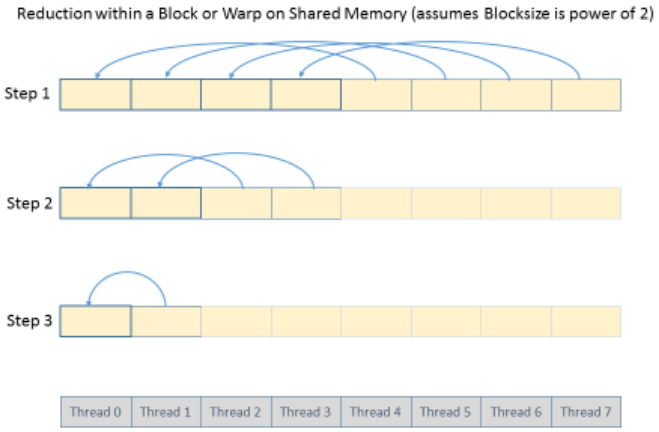
Highly Parallel Programming of GPUs
Lab 3

Reduction

Parallelize the sum $\sum_i^n a\_i$ (reduction) for the GPU.

1. Get the code template by `tar xf lab-reduction.tar.xz`
2. Use global memory and atomics (`atomicAdd(&var, value);`)
3. Implement reduction with Shared Memory and without atomics
   - You can assume $n = 2^k$ and blocksize = 2^p
4. Which loop should be unrolled to improve memory bandwidth?





Game of Live

Story

The Game of Life is a system designed by the mathematician John Conway, based on a two-dimensional cellular automaton. The game field is divided into rows and columns. Each cell of the field can take one of two states which can be considered alive and dead. The next generation of a configuration results from simple rules.

- A dead cell with exactly three living neighbours is reborn in the next generation.
- A living cell with two or three living neighbours remains alive in the next generation.
- Living cells with a different number of living neighbours die in the next generation.

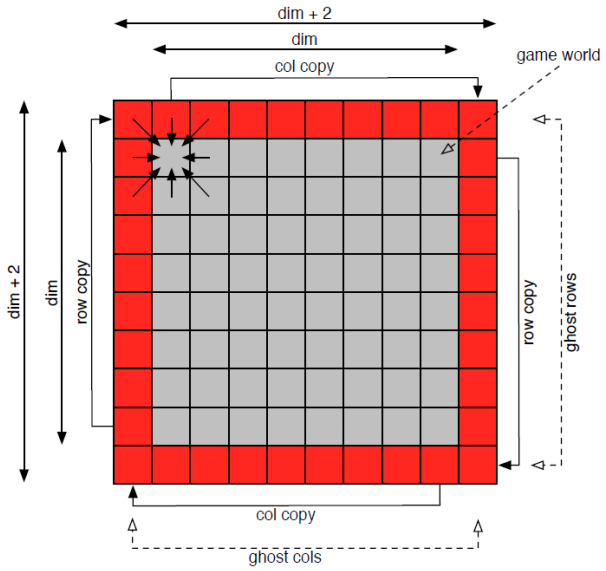
Furthermore it is easily possible to define other rules. Conway's world is a 23/3 world. (One cell stays alive with 2 or 3 living neighbours. A cell is born again with exactly 3 living neighbors.)

Quests

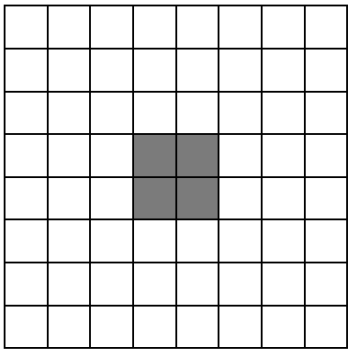
1. Implement a sequential variant of the Game of Life using the predefined code framework. Use the classic rules of Conway and implement a world whose borders are connected to each other.

   The dimension of the square game field should be dynamically changeable. Use the function malloc()/new() to define a linearly addressable memory area and consider a meaningful mapping of the matrices into this memory area.

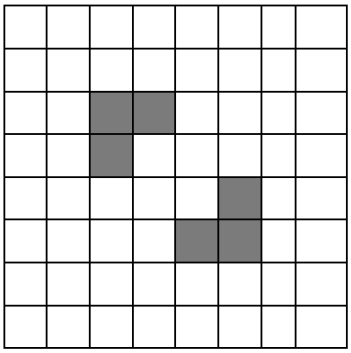
   The following figure helps to understand, how the connected edges using shadow rows and columns can be realized.



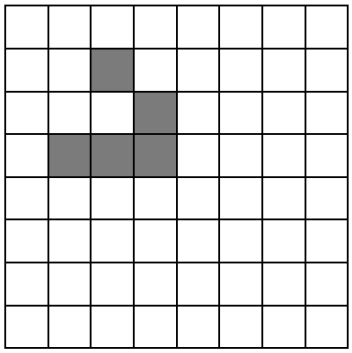
2. Check your implementation against the objects shown below. These should be static, oscillating, and moving.



static block



oscillator



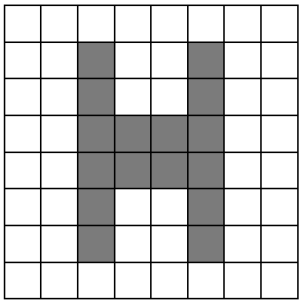
glider

With a correct implementation and a random image neither chaos nor an empty world is produced. Check the correct behaviour at the edges with a glider. When crossing a border the glider should reappear on the opposite side.
Note:
To check the behavior you can use the createPPM(...) function to create a PPM file of the game field at the final state.

3. Extend your algorithm so that the calculation of the next generation of the problem is done on the GPU. Distribute the work among the GPU's multiprocessors.

Side Quest

- Implement the rules of a 1357/1357 world. These provide a copy world that duplicates its contents. Initialize a field of size 1024 x 1024 and initialize it in the middle with an H.



Letter H to initialize the center of the game field

- Run your algorithm and check the correctness of your rules. The H should be duplicated again and again.
- Furthermore, determine the speedup, which results from the calculation on the GPU.
- Determine the number of calculated generations where the GPU starts to have a runtime advantage over the CPU. Adjust the achieved speedup for different problem sizes graphically (e.g. with Gnuplot).