POLITECNICO DI MILANO

μ-LAB

QCAdesigner - CUDA

HPPS project

UIC

Giovanni Paolo Gibilisco

Marconi Francesco

Miglierina Marco

DRESD

WWW.DRESD.ORG

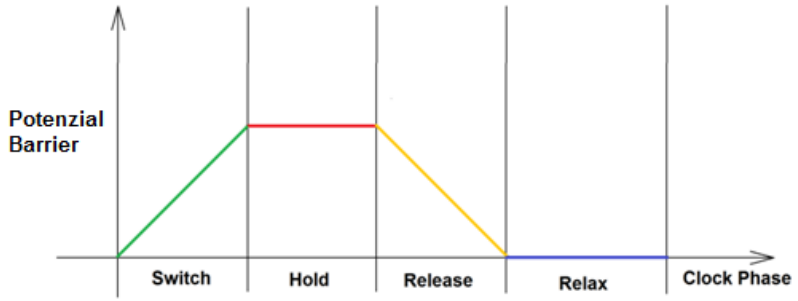
Outline

- Introduction to QCA
- Why CUDA?
- Scope & Motivation
- Implementation
- Results
- Conclusions

QCA

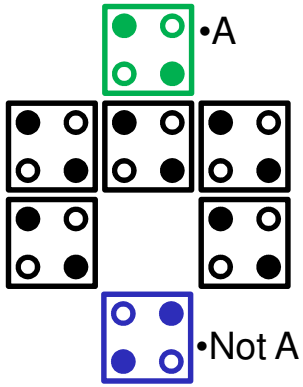
- QCA is a technology based on Quantum Dots.
- Four quantum dots are placed to make a cell, in which are introduced some electrons.

- The cell has 2 possible stable configurations.
- The behaviour of each cell in a circuit is controlled by a clock signal the circuit uses 4 clock shifted of $\pi/4$

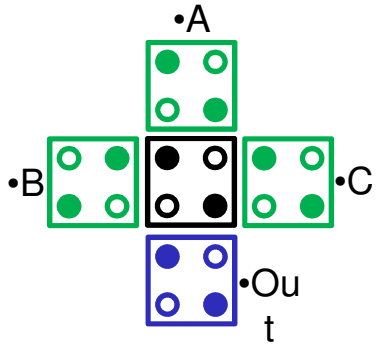


QCA

- Basic blocks can be created aligning different cells



Not

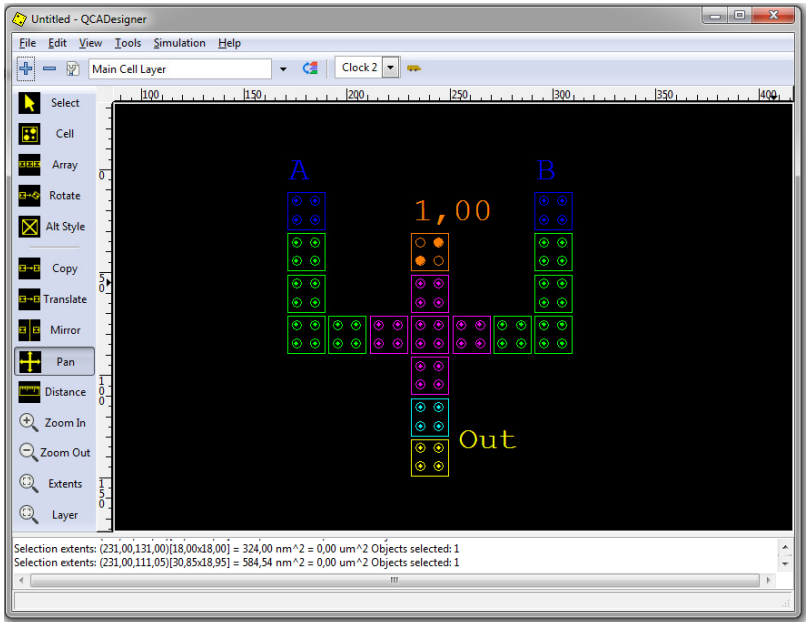


Majority

- Clock phases define the flow of information into the circuit
- QCADesigner is a tool developed in the University of British Columbia to design and simulate this kind of circuits.

QCADesigner

- QCADesigner offers a GUI in which circits can be built



- There are also two simulation engines:
  - Bistable
  - Coherence vector
- We focuse only on the bistable engine.

Scope & Motivation

- The aim of this project is to speedup the execution of the Bistable simulation engine in the QCADesigner tool by paralleizing its execution.
- Bisteble engine verification is used frequently in the development of a circuit as a form of debugging.
- This simulation engine need to be very fast to effectivly support designers.
- Actual execution time for a circuit with 6 input 7000 cells is more than 1h.

Bistable simulation

- Bistable simlation engine characteristics:
  - Fast simulator
  - Simplifies the physics behind QCA
  - Parameters that mostly affect simulation time are number of samples and number of cells
  - For an exhaustive verification the number of sample grows explonentially with the number of inputs

Bistable Simulation

- Simulation algorithm

Algorithm 4.2.1: Original Bistable Engine Pseudocode

```
   Data: NewP :
1  for j ← 0 to numberOfSamples do
2  |  UpdateInputs(Pol)
3  |  RandomizeCells(Pol)
4  |  i ← 0
5  |  stable ← false
6  |  while i <maxIterations and not stable do
7  |  |  for k ← 0 to nCellLayers do
8  |  |  |  for l ← 0 to nCellsPerLayer do
9  |  |  |  |  OldP ← Pol_{k,l}
10 |  |  |  |  NewP ← ComputePolarization(Pol_{k,l})
11 |  |  |  |  stable ← ((|NewP − OldP| < tolerance) and stable)
12 |  |  |  |  Pol_{k,l} ← NewP
13 |  |  |  endfor
14 |  |  endfor
15 |  endw
16 |  CollectOutputs(Pol)
17 endfor
```

- Profiling has shown that the 99% of the time is spent in this loop.

CUDA

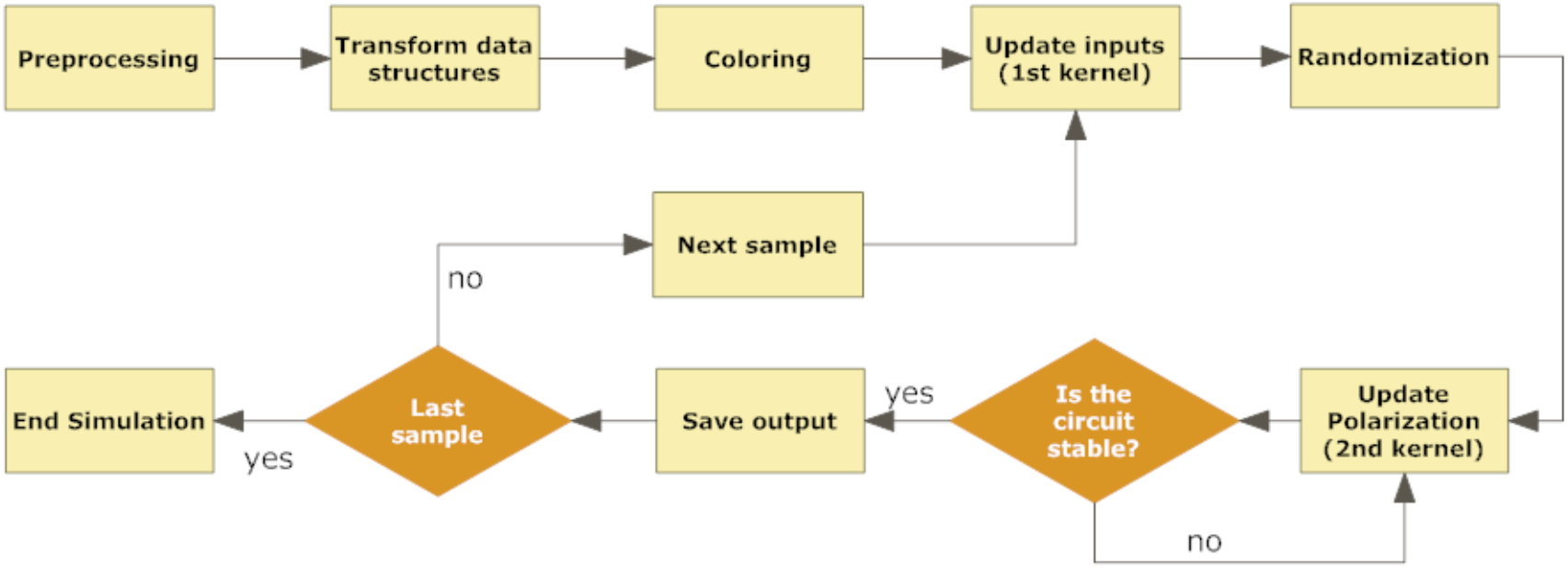
- To speed up the simulation we need to parallelize the computation of the new state of each cell
- As long as the code that update the polarization is the same we need a SIMD architecture to execute the code more efficently
- The great number of cores offered by CUDA architectures makes it a good candidate for this kind of parallelization

CUDA

- CUDA (Compute Unified Device Architecture)
  - Is a programming model developed by Nvidia
  - Allows the use of GPU to execute parallel code
- Cuda abstracts hardware parallelism with the concept of blocks and threads.
- The main structure needed to exploit CUDA parallelism is the Kernel which is a parallel function invoked by the host and executed on the device

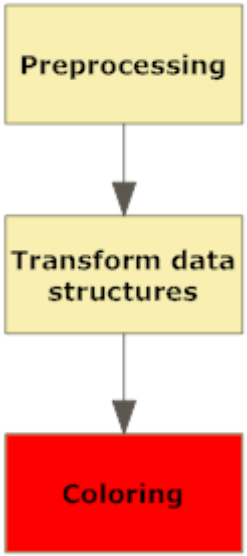
Implementation

- The main challenge in the implementation of the simulator has been preserving the correctness of outputs.
- The parallel simulation code follows these macro-steps:



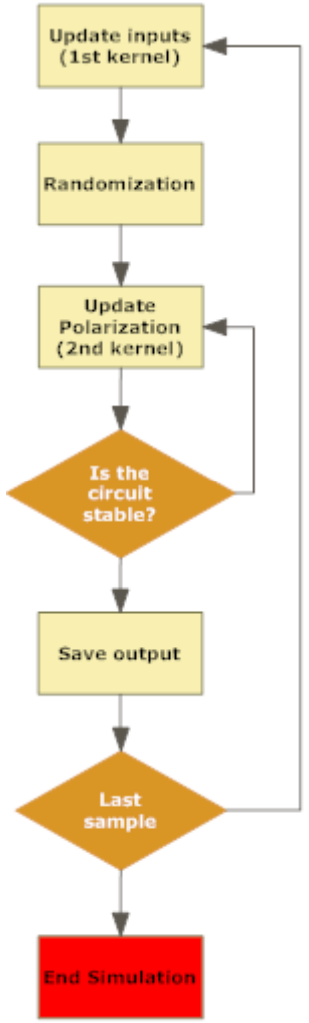
Implementation

- Parallel simulation Algorithm
- Load data file, find neighours, calculate kink energy.
- Original structures are mapped to array, indexes are saved to acces new strucutres in the kernel.
- A coloring algorithm is applied to the graph representing the circuit in order to find non neighboring cells that can evolve simultaneously.



Implementation

- Cells which represent input of the circuit are updated. This is done on the device with an ad-hoc kernel.
- The color order is randomized to minimize numerical errors.
- The main simulation kernel is called once for each color. Each run of the kernel updates only cell's of the current evolving color. At the end of this phase all cells are updated with the new polarization.
- This process is repeated until the circuit converges.
- When the circuit has stabilized the value of output cells are stored and the loop starts again for the next sample.
- When the execution of all samples has ended the simulation output is stored in the form of binary outcome, values of output cells polarization and an image with the shape of the output wave.



Memory Exploitation

Shared memory

• Array with all indexes to inputs and outputs

Constant memory

• Clock value

•Numbr of cells

•Number of sample

•Stability tolerance

- Small structures frequently accessed are stored in shared memory.
- Data which doesn't change during the simulation are stored in constant memory
- Big data accessed by all threads are stored in global memory

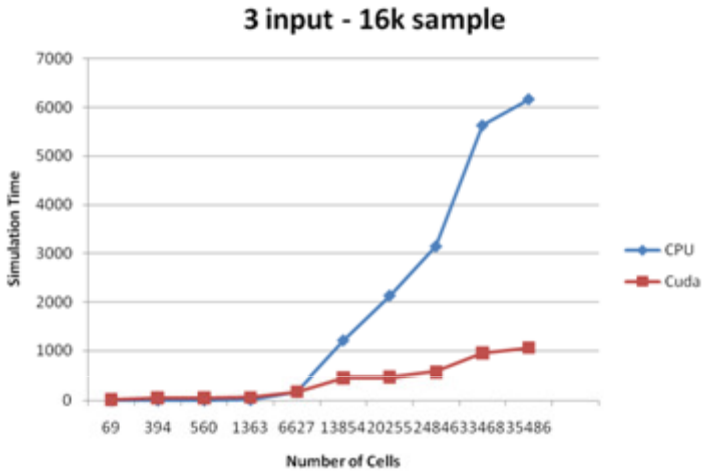
Global memory

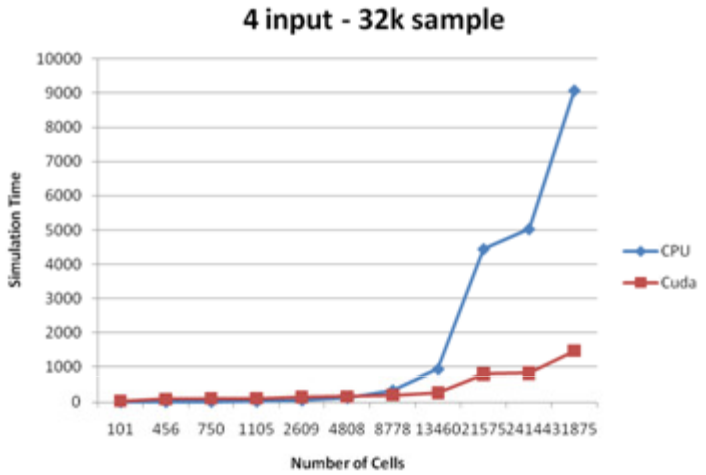
• Array with all cell's polarization

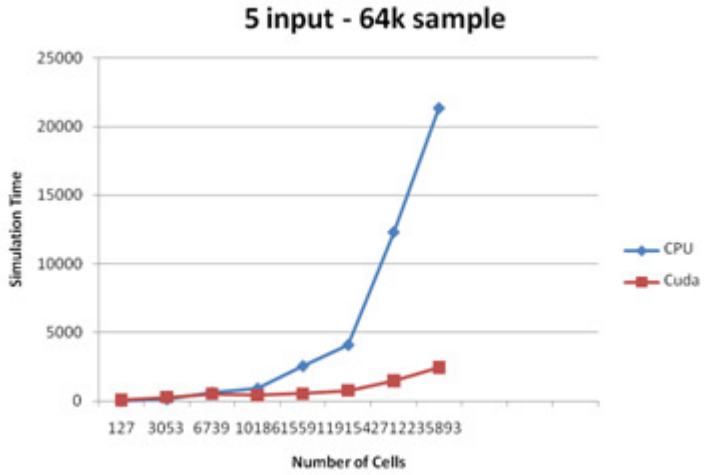
•Array with all kink energy

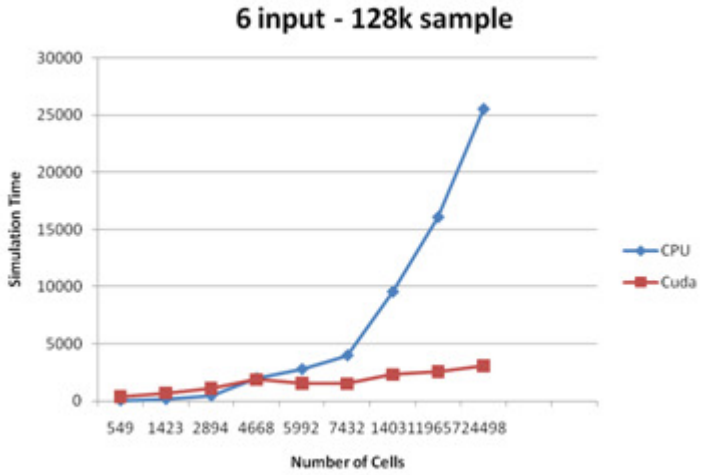
Results

- Comparison between CPU and GPU simulation time.



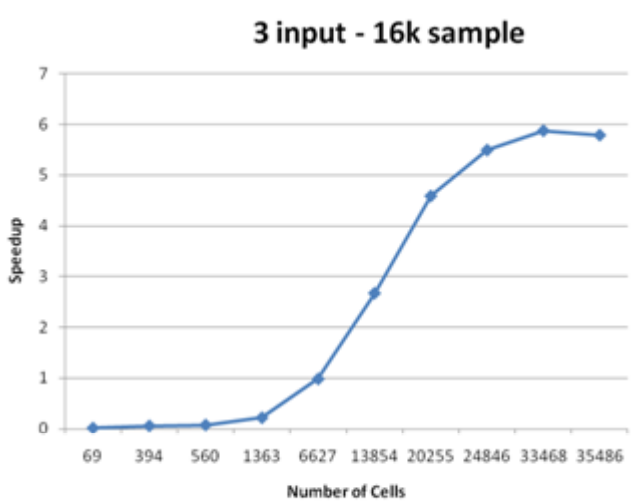


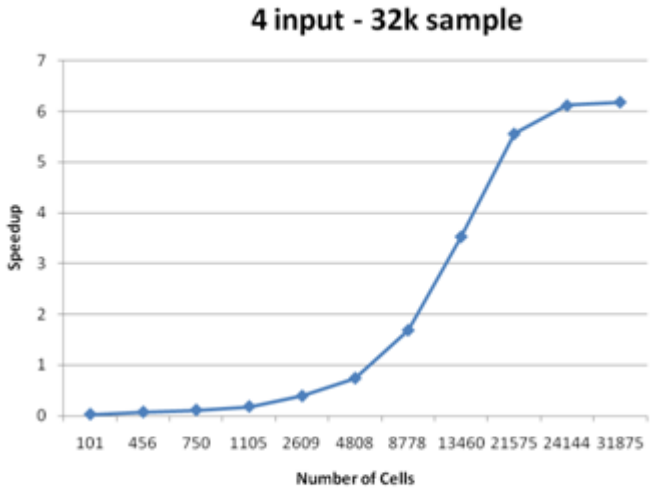


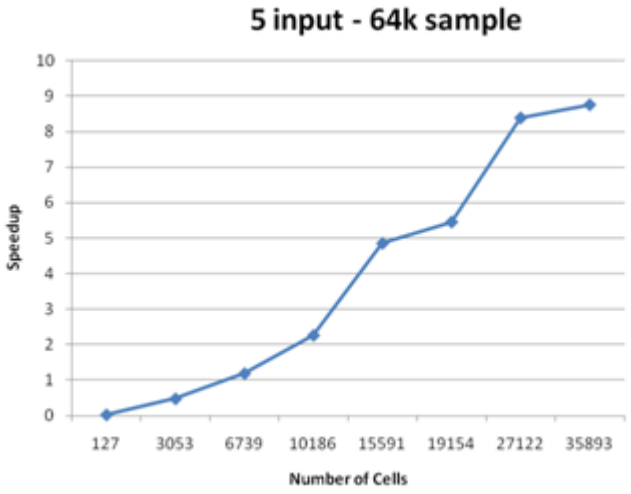


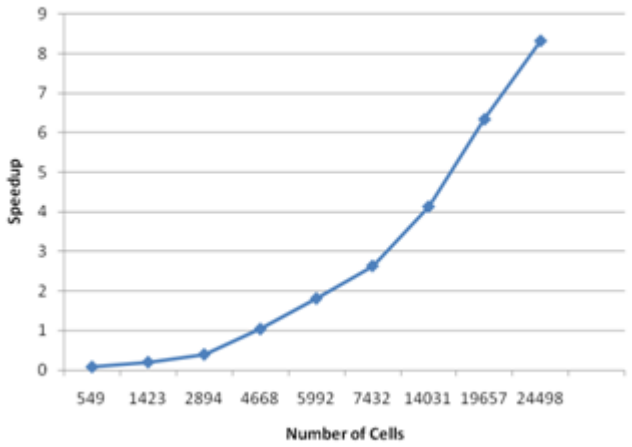
Results

- Speedup.









Conclusions

- The new implementation scales very well with the number of cells.
- Maximum speedup reached over 8x
- Better knowledge of both QCA and CUDA technology

- Future improvement:
  - Better exploitation of memory hierarchy
  - Fixing oscillations in order to avoid coloring
  - Implementing a non exhaustive verification

Questions