High performance reconfigurable computing with cellular automata

Murtaza, S.

Publication date
2010

Chapter 2

Background

This chapter presents the necessary background for the various topics that form the basis of this work. Since one of the main focus of this research has been the implementation of accelerators for cellular automata, an introduction in general is presented for such algorithms followed by the introduction of the various cellular automata models that were implemented as a part of this work. Another dimension has been the design and implementation of cellular automata accelerators using reconfigurable devices like FPGAs, a section is devoted as an introduction to such devices. A brief discussion is presented on the recent emerging trend using reconfigurable device enabled high performance computers also known as high performance reconfigurable computing (HPRC). The chapter concludes with a discussion on the related work in the field of FPGA based CA computations and HPRC in general.

2.1 Cellular Automata

Cellular Automata are decentralised spatially extended systems consisting of large numbers of simple and identical components with local connectivity [92]. Such systems have the potential to perform complex computations with a high degree of efficiency and robustness, as well as to model the behaviour of complex systems from nature. CAs have been studied extensively in natural sciences, mathematics, and computer science. They have been considered as mathematical objects about which formal properties can be proven and have been used as parallel computing devices, both for high-speed simulation of scientific models and for computational tasks such as image processing. CAs have also been used as abstract models for studying emergent cooperative or collective behaviour in complex systems [96]. In addition CAs have been successfully applied to the simulation of a large variety of dynamical systems such as biological processes including pattern formation, earthquakes, urban growth and most notably in studying fluid dynamics. Their implicit

spatial locality allows for very efficient high performance implementations and incorporation into advanced programming environments. For a selection of the numerous papers in all of these areas, see, for instance, [30, 40, 51, 59, 93, 94, 95].

2.1.1 What are Cellular Automata?

CA are dynamical systems where space, time and state are discrete. And in general are characterised as a regular grid of simple finite state machines (often called cells in CA theory) of the same kind with communication between the machines limited to local interactions. Each individual cell changes its state over time depending on the state of the cells in its local neighbourhood. The next state rules are deterministic and the overall structure can be viewed as a parallel processing device with cells updated concurrently with each time step.

Formal definition

Formally, a CA is specified by a quadruplet $(\mathfrak{L}, S, \mathcal{N}, f)$ where:

- $\mathfrak{L}$ represents the lattice.
- S is the finite set of possible states of each of the cell.
- $\mathcal{N}$ is the finite set of neighbourhood.
- f is the local transition rule.

The lattice is a discrete regular grid of cells within a finite dimensional space ($1, 2, 3$ used in practice) where each cell is defined by its discrete position and its discrete state. Each cell can be in one of the finite number of k possible states. Since the time evolution of the CA is also discrete, S_i the state of the i^{th} cell at a new time $t+1$ is a function of the present state of a finite number of cells in its neighbourhood at time t as:

$$S_i(t+1) = f_i\left(S_j(t)\right) \tag{2.1}$$

where $i \in \mathfrak{L}$ and j is the sub lattice neighbourhood about i^{th} cell.

Types of lattice, neighbourhood, and boundary condition

The binary state, nearest neighbour, one dimensional cellular automata is the simplest cellular automata. And a system composed of N uniform cells would look like an array of ones and zeros of width N where the neighbourhood of a cell are the local cells on the either side. The state of all cells form the global configuration of the CA. For two dimensional CAs, various types of lattices are possible, for example, triangular, hexagonal or rectangular. The common types of neighbourhood are Von Neumann (5 cells, consisting

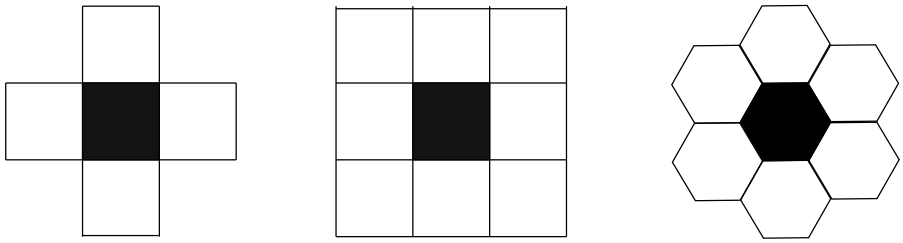

Figure 2.1: *Neighbourhood configurations: (left to right) Von Neumann, Moore and Hexagonal.*

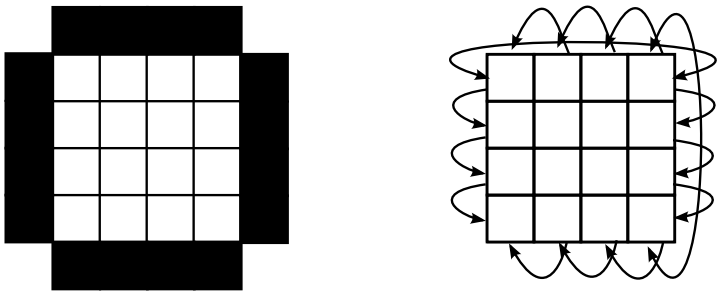

Figure 2.2: *Boundary configurations: (left to right) Fixed and Periodic.*

of the cell along with its 4 neighbours), Moore (9 cells, cell along with its 8 neighbours) and Hexagonal (7 cells, cell along with its 6 neighbours) as shown in Figure 2.1. Various boundary conditions are also possible as shown in Figure 2.2. Fixed boundaries: The boundary cells are assumed to have a *specified* fixed value, for example, in a null boundary configuration the boundary cells are assumed to have a value 0. Periodic boundaries: One in which the lattice is considered to be wrapped as illustrated in Figure 2.2 (right).

According to above given definition for CAs, they are considered to be synchronous and deterministic. However, in some applications it is desired to have some degree of randomness in the rule or to have an asynchronous update scheme.

2.1.2 The Game of Life

The Game of Life, perhaps the most famous CA algorithm, was introduced by the British mathematician John Conway in the beginning of the 70's [115]. The Game of life is a simple model for simulating artificial life. The automata is implemented on two-dimensional square lattice with periodic boundary conditions (Figure 2.2, right one) where each lattice site is either *alive* (represented by value 1) or *dead* (represented by value 0). Therefore, a single bit is used to represent each site and the lattice is a Boolean matrix. The system evolves by updating each of the lattice sites simultaneously based on the update rules that are determined by the state of each cell and their respective neighbourhood (Moore neighbourhood (Figure 2.1, middle one). The exact rules are as shown in Table 2.1. The Game of Life contains many interesting patterns and the most extensively studied pat-

Living neighbours	Next state (if currently alive)	Next state (if currently dead)
0 or 1	dead	dead
2	alive	dead
3	alive	alive
4, 5, 6, 7 or 8	dead	dead

Table 2.1: *Game of Life: The cell update rules.*

tern is known as the *gliders* (see [109] for details), also the first known finite pattern with unbounded growth. And despite its simple architecture, it has been proved that Game of Life with an infinite-sized universum has the same computational power as a *universal Turing machine* [83].

2.1.3 Lattice Gas Cellular Automata

Perhaps the most successful practical application of cellular automata as computing devices has been in the field of fluid dynamics for fluid simulations [92]. Called Lattice Gas Cellular Automata (LGCA), this class of CA mimics a fully discrete fictitious fluid. Both the positions and velocities of the fluid molecules are discrete and tightly coupled to the discrete lattice. All particles perform free streaming from one lattice node to a neighbouring one in a time period. Next, particles arriving at a node collide with each other, thus exchanging momentum in some deterministic or stochastic way. The collisions on the nodes all occur at the same time and the duration of a collision is assumed to take zero time. The enforcement of conservation of mass, momentum, and energy in a collision results in a fully discrete and simplified, yet physically correct microdynamics. The inherent spatial locality of update rules makes it ideal for parallel processing [41, 92].

With this LGCA dynamics we may then investigate macroscopic variables, that is, averaged quantities such as fluid density or momentum, which vary over time and length scales much larger than those of the microdynamics, and for which we can prove that they behave as a real fluid. The most complete account of LGCA is the book by Rivet and Boon [84]. Other influential monographs on LGCA are [30, 85]. [14] provide a through overview of Lattice gases and CA. Finally, [92] explains LGCA in the context of CA modeling, and also discusses issues related to executing LGCA on a computer.

The HPP model

The HPP model was the first LGCA model invented and introduced by Hardy, Pomeau and de Pazzis in 1973 [81]. In this model the particles are restricted to move on the links

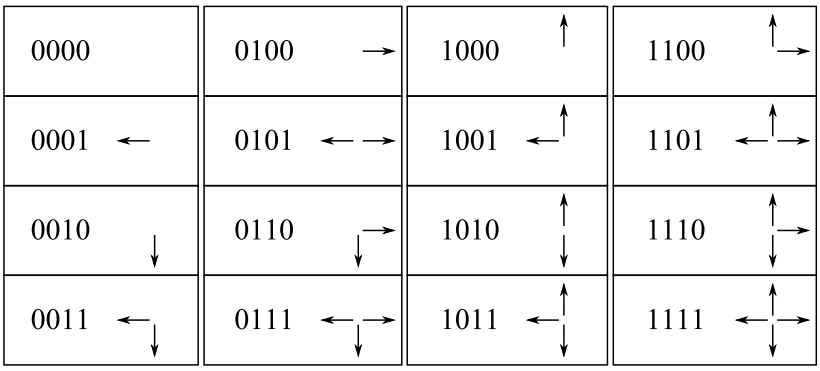

Figure 2.3: *The HPP configurations.*

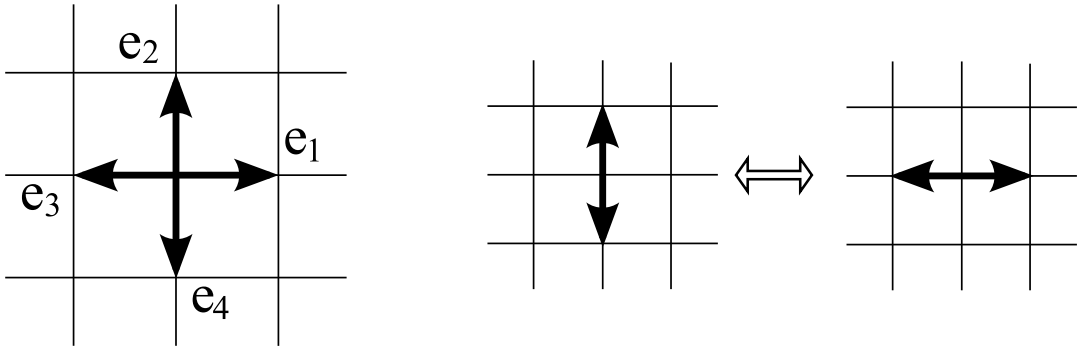

Figure 2.4: *The HPP model: (left to right) Lattice with four velocities and the collision rule.*

of a square lattice (see Figure 2.1 left one). Each particle travels at a unit speed, that is, in each time step it moves from one lattice site to a neighbouring site. A particle can have only one of the four discrete velocities $e_i, i \in 1, 2, 3, 4$ (see also Figure 2.4 left one):

$$e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; e_3 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}; e_4 = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{2.2}$$

As only one particle is allowed to travel in each direction along a link, a maximum of four particles can arrive at any site at any time step. Therefore, a 4-bit state machine is used to represent each of the HPP lattice site. The possible configurations of particles at each site are shown in Figure 2.3 along with possible coding of the 4-bit state machine. The dynamics of the system from one time-step to another takes place in two successive stages, that is, *collision* and the *streaming* stage. At the start of each time-step, the incoming particles at a node collide following the rules as shown in Figure 2.4 (right). After the collision stage, particles perform a free streaming, that is, each particle travels from its node to the neighbouring node in the direction of their velocity vector. It can be easily seen from the collision rules that both the number of particles and the momentum at each site is conserved, and therefore conserving the mass and momentum for the whole system. Figure 2.5 shows an example system dynamics from time-step $t-1$ to $t+1$, lower

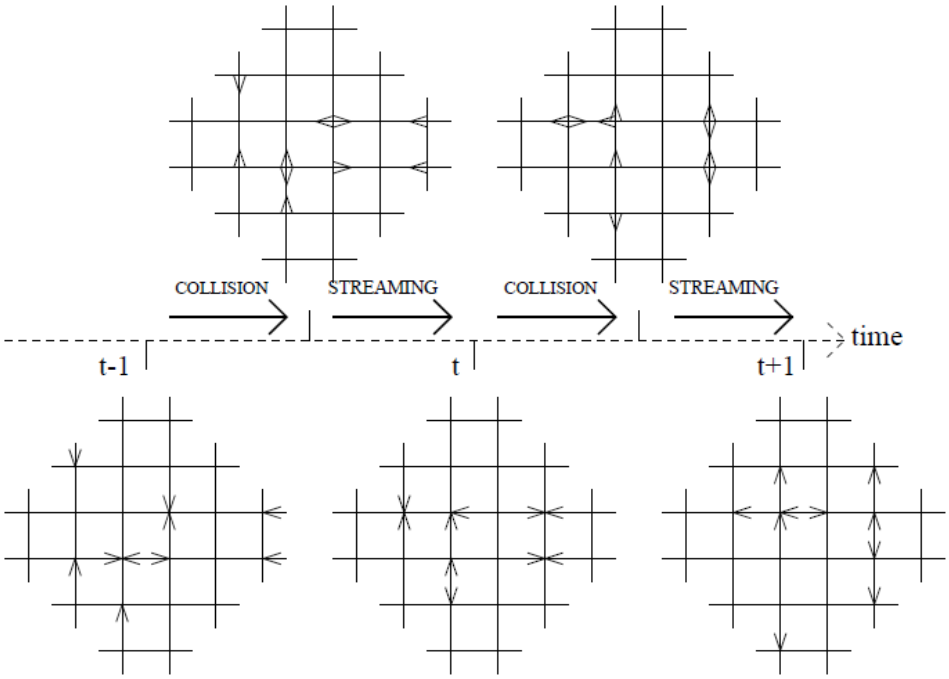

Figure 2.5: *HPP LGA update mechanism. (left to right) Propagation and the collision phases for a portion of a square lattice from time $t-1$ to $t+1$ are shown where arrows represent particles and their directions of motion. Figure taken from [22].*

panel shows the state of the lattice at the three specified time-steps and the top panel shows how system evolves from one time-step to another.

Although the HPP model was the first LGCA, however, it was quickly realised that it was insufficient for correctly simulating flows as governed by the Navier-Stokes equation. Later, it was also realised that this was due to the models underlying square grid and was the main consideration that led to the birth of the FHP model based on a hexagonal lattice where these problems were eliminated (for details see [30, 85]).

2.1.4 Lattice Boltzmann Method

Following the discovery of LGCA as a model for hydrodynamics, in 1988 another model, the lattice Boltzmann method was introduced. This method is reviewed in detail in [100]. The basic idea being that one should not model the individual particles but particle densities, that is, the probability of finding a particle with a certain velocity. This means that particle densities -real numbers- are streamed from cell to cell, and particle densities collide. In a strict sense we no longer have a CA with a Boolean state vector (in fact, the state is now a vector of real numbers, so the state space per node is infinite). However, we can view LBM as a generalized CA, with the same computational structure as explained by [92].

As shown in Figure 2.6, one of the lattice types used for LBM simulations is the D2Q9-lattice where 2 refers to the lattice dimensions and 9 is the number of discrete velocity vectors. For each discrete velocity vector e_0 through e_8 as shown in Figure 2.6 there is a probability density distribution f_0 to f_8 of particles with that velocity. In terms of CA modeling, in every time step, collision and propagation operations are performed at each of the lattice nodes.

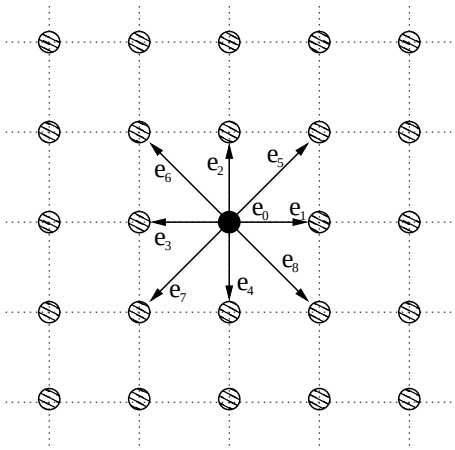

Figure 2.6: *The LBM D2Q9 lattice where $e_i \rightarrow$ discrete velocity.*

The discrete lattice Boltzmann equation with BGK collision approximation (for detailed discussion see [63]) can be described as:

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i\Delta t, t + \Delta t) - f(\boldsymbol{x}, t) = \Omega_i \tag{2.3}$$

where $f_i(\boldsymbol{x}, t)$ is the particle distribution function traveling with discrete velocity $\boldsymbol{e}_i$, with i representing the number of discrete velocities. For a square lattice D2Q9 model as shown in Figure 2.6, particle at rest $\boldsymbol{e}_0$=0, for $i = 1, 2, 3, 4$,

$$\boldsymbol{e}_i = e \begin{pmatrix} cos[(i-1)\pi/2] \\ sin[(i-1)\pi/2] \end{pmatrix} \tag{2.4}$$

and for $i = 5, 6, 7, 8$,

$$\boldsymbol{e}_i = e\sqrt{2} \begin{pmatrix} cos[(i-5)\pi/2 + \pi/4] \\ sin[(i-1)\pi/2 + \pi/4] \end{pmatrix} \tag{2.5}$$

where particle streaming speed $e = \Delta x/\Delta t$. Δx and Δt are the lattice spacing and the step size in time respectively. The right hand side of the Equation (2.3), that is, Ω_i is the so-called BGK collision term that models the collision function of the said model as in Equation (2.6)

$$\Omega_i = -\frac{1}{\tau}[f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)] \tag{2.6}$$

where τ is the dimensionless relaxation parameter and $f_i^{eq}(\boldsymbol{x}, t)$ the equilibruim distribution function. For D2Q9 lattice, using the macroscopic variables such as density ρ and velocity $\boldsymbol{u}$ one obtains the following form

$$f_i^{eq}(\rho, \boldsymbol{u}) = \rho\omega_i \left[1 + \frac{3(\boldsymbol{e}_i.\boldsymbol{u})}{c^2} + \frac{9(\boldsymbol{e}_i.\boldsymbol{u})^2}{2c^4} + \frac{3(\boldsymbol{u}.\boldsymbol{u})}{2c^2}\right] \tag{2.7}$$

where weighing factor ω_i is $\omega_0 = 4/9$, $\omega_i = 1/9$ for $i = 1, 2, 3, 4$ and $\omega_i = 1/36$ for $i = 5, 6, 7, 8$. And the macroscopic density and velocity are defined by sums over the distribution functions f_i :

$$\rho = \sum_i f_i, \qquad \rho\boldsymbol{u} = \sum_i f_i \boldsymbol{e}_i. \tag{2.8}$$

For FPGA implementations as shown in Chapter 5, parameters $\Delta x = \Delta t = 1$ were in lattice units for the above model.

Ignoring the details of the LBM computation being performed at each node as presented above (or see, for example, [92] and [71] for further details) and to simplify the explanation, we assume each lattice site to be nothing but a black-box. Each black-box is initialised with the lattice node's nine probability density distributions. For the computation, the initialised black box runs a number of mathematical operations over the given nine real numbers and returns with a new set of nine real numbers (collision function: in this case using a BGK collision operator [92]) where f_1-f_8 are exchanged with the node's eight neighbours (propagation function) and f_0 represents the particle at rest internal to the node.

2.2 Reconfigurable Computing

Traditionally to perform a computation, one would write a software program implementing the algorithm that would be executed using a microprocessor- a general purpose computer. Since processors execute a set of software instructions to perform a computation and by changing these software instructions one can modify the functionality of the given system without any changes in the processors' hardware. This general purpose nature of a microprocessor makes it the most flexible method of implementing a computation. However, this flexibility comes at the cost of performance, that is, to execute a program, the processor has to read, decode, and execute each of the instruction thus resulting in a high execution overhead for each instruction. Alternatively, for a faster and efficient computation one would design and build a special purpose computer, customised and hardwired to perform the given computation in the hardware directly. However, this high performance is achieved at the cost of flexibility, that is, the hardwired machine cannot

be altered after fabrication. See [21, 47, 57] for an interesting list of articles on special purpose machines in science in general. At the chip level, Application Specific Integrated Circuits or ASICs [43] are designed specifically to perform specified computations faster and more efficiently, a device such as Grape [72] is an example. Again, the chip cannot be altered after fabrication. Reconfigurable computing with FPGA offer a middle ground that combines the flexibility of the microprocessors and the tailor designed circuitry offered by the ASIC. See [7, 31, 32, 39, 101] for an introduction to reconfigurable computing using FPGAs. The fundamental and powerful concept behind using FPGAs is that, the circuitry within the chip is reconfigurable and tailored as per the application requirements.

2.2.1 FPGA

FPGAs are commodity integrated chips that are completely electrically programmable and can be customised almost instantaneously by the designer to create custom digital logic designs after manufacturing (hence the name field programmable). See [106] for an overview of FPGAs and their programming. [19, 64] are also interesting books concerning digital logic design using FPGAs. Current FPGAs include logic density equivalent to millions of gates per chip [74] and can implement very complex computations. Traditionally, used as glue-logic replacement and rapid-prototyping in networking, telecoms, DSP, etc, FPGAs with improved features like flexibility, capacity and high performance have also opened up new avenues in HPC that form the basis of HPRC.

2.2.2 Structure

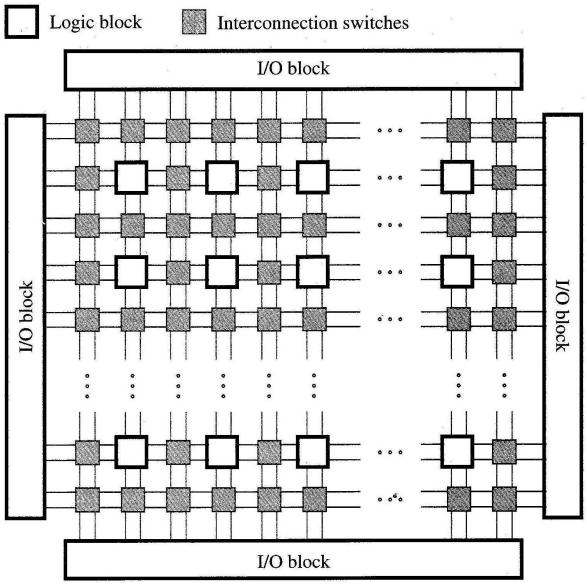

Figure 2.7: *General structure of an FPGA. Image from [19].*

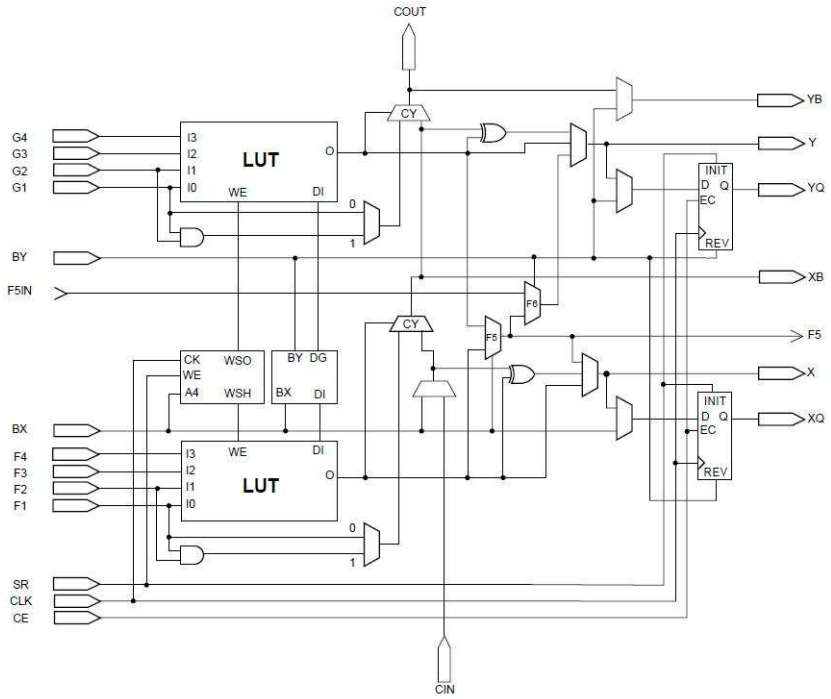

Figure 2.8: *FPGA slice with two 4-input LUT and two Flip-Flops. Image from [43].*

The structure of an FPGA in general as shown in Figure 2.7, is based around a matrix of programmable functional units called configurable logic blocks (CLB) connected via wires that are also programmable. In order to provide connectivity to the outside world, a flexible interface called I/O blocks surrounds this programmable matrix. The majority of the FPGAs are based on SRAM technology [31, 32], that is, the logic blocks and the programmable interconnect use memory cells called static random access memory (SRAM) cells to store programmable information. Hence, the configurable points within the FPGA are SRAM bits and programming these SRAM bits configures the FPGA. Therefore, re-programming these devices is as easy as writing to a standard static RAM in a matter of seconds.

CLB being the functional units for most of the logic implementation include logic cells as their basic building blocks. As shown in Figure 2.8, each logic cell includes one or more lookup tables (LUT), carry logic, and several storage elements called Flip-Flops. LUTs are essentially memories used as arbitrary logic function generators. The basic idea is to store the output value of an n-input function for each input combination in a table, and using current input value to lookup what the value should be by indexing the table. Typically, LUT have either 4 or 6 inputs. Thus using the hierarchical structure, that is, the combination of small scale functions implemented using LUT are interconnected together to form large scale functions. Therefore, using a FPGA chip, a designer can implement a custom digital logic simply by mapping the design's functional units onto the available

logic blocks. The maximum possible clock frequency that can be used to drive the overall circuit depends on the underlying technology and the depth of the computation between the storage elements and the routing wire delays.

Other than the CLB, IOBs and interconnects as shown in the Figure 2.7, FPGAs also include internal memory blocks called block RAMs (BRAM). High-end FPGAs also include many dedicated functional units and peripherals like DSP units, fast I/Os, multiple clock control units and even embedded processors like RISC cores to further boost the performance of the implemented design.

2.2.3 COTS Boards

Typical FPGA based computing resources used are in the form of COTS (commercial off the shelf) boards, see Appendix 10 for the one used for this research. FPGA based COTS boards are widely available and easy to setup. Apart from including high-density, high-performance FPGA(s), they also include on-board memory chips and various other I/O related components. Usually, FPGA chips are connected to several on-board SRAM or DRAM memory chips that can be used for additional data storage requirements.

2.3 HPC using FPGAs

Applications from scientific computing and their ever-increasing demand for computing resources are the two sides of the same coin, and how to further push the performance of HPC applications using HPC systems is always a challenge. In an attempt to boost the performance of HPC applications, currently HPC systems incorporate hardware accelerators like FPGAs chips. Such systems are an extension to the traditional HPC systems with additional reconfigurable devices like FPGAs and are also known as High Performance Reconfigurable Computing (HPRC) [42, 55, 56] or reconfigurable supercomputers [20].

In addition to exploiting the available coarse-grained parallelism across the microprocessors, inclusion of hardware accelerators like FPGAs provides an opportunity to the HPC programmer to further boost the application performance. This is achieved exploiting the massive fine-grained parallelism and pipelining available through direct hardware execution using FPGAs. FPGAs allow a programmer to design and build a custom circuit to compute his explicit computation and therefore, an opportunity to parallalise and pipeline algorithms on the instruction level to achieve tremendous performance gains. Additionally, HPRC systems have also shown orders of magnitude improvement in performance, power, size and cost over conventional HPC systems [42].

Other than proof-of-concept setups like Maxwell [8], RCC [87] or Janus [10], the vendor provided solutions like Cray's XD1 [60], DRC Computers' RPU [34], SGI's RASC [89] and SRC Computers' MAPstations [33], which couple FPGAs with high-end conventional microprocessors, are of increasing interest to the scientists working in computational science and engineering communities.

2.3.1 Exploiting HPRC Resources

Since HPRC combines high performance microprocessors along with reconfigurable devices like FPGAs, all communication is over a high bandwidth and low latency interconnection network [50], both the processing and connecting capabilities can be tailored to suit the needs of the specified algorithm in order to maximise performance. However, to harness the capabilities of the available resources, it is necessary to a) identify kernels with high computational density, b) map possible kernels to hardware accelerators, c) carefully partition the application into software and hardware to reduce overall communication overheads, and d) optimally utilise the available memory hierarchy.

The resources available to the HPRC programmer include programmable logic, on-chip memory, on-chip decided arithmetic blocks, on-board memory, host memory and the high speed interconnects. Therefore, the overall application performance is defined by how efficiently the available resources are configured to maximise the available bandwidth. [55] lists and explains the possible FPGA computing models that can effectively utilise the available computing resources depending on the application requirements. In general, spatial parallelism available within FPGA can be exploited to implement a) pipelined and parallel algorithms (we have implemented pipelining over time strategy for our I/O bound CA implementations as discussed in Chapter 4, shown in Figure 4.2 and multiple processing elements running in parallel implementation as discussed in Chapter 5, shown in Figure 3.2), b) customised data paths (see lattice Boltzmann processing element implementation as shown in Figure 5.2), and c) using internal buffers like FIFOs for temporary data storage to minimise I/O communications. The overall memory hierarchy (on-chip, on-board, and host memory) is exploited to maximise available bandwidth. For example, in our multi-FPGA based lattice Boltzmann implementation as discussed in Chapter 6 and 7, multiple-independent on-board SDRAM memory banks store the lattice and the boundary data, on-chip memory as FIFOs buffering input and output data streams connected to the processing elements, on-chip memory as register file within each processing element storing the cells state during next state computation, and host memory for boundary update over multiple FPGAs. In Chapter 7, we demonstrate using system profiling, a successful implementation of our latency hiding based boundary update operations by

host machines over multiple FPGAs. Such working latency hiding was achieved due to the successful exploitation of the available memory hierarchy.

2.4 Related Work

As CA provide a mathematical representation of a wide range of complex systems [59], it is not surprising to find numerous attempts by researchers both in the past and present to find methods of how to effectively simulate CA systems in an attempt to understand their emergent behaviour, using the available state-of-art computing resources. As an example, one of the famous cellular automata machines has been CAM by Toffoli and Margolus [102]. Margolus states in [73] "our CAM simulations made Pomeau and others realise that lattice gases were not just conceptual models, but might be turned into powerful computational tools". This is a testimony to the success of such machines. See [48] for a list of CA as parallel computing machines. With the availability of programmable chips, such as FPGAs, one can easily emulate a special purpose chip instead of building a special purpose computer using custom VLSI chips [47] that allows virtually every research an opportunity to design his or her special purpose computer using FPGA chips. Consequently, many FPGA-based CA systems have been proposed and implemented. The following section discusses some of the FPGA-based CA systems reported in the current decade.

Researchers at TU Darmstadt developed a series of FPGA-based CA accelerators called CEPRA-machines to answer "How much speedup could be gained by using FPGA technology compared to optimised software?" [52] speculated a thousand times speedup using the then available latest high end FPGA technology.

In 2001, Cappuccino et al. [25] proposed an FPGA-based CA computational engine called CAREM and demonstrated its implementation for an image thinning algorithm and forest fire simulations with a speedup of 65 and 24 respectively. Though the implementation models were simple with a four or less bits per cell state but engine layout used two independent external memory banks to store the current and the next state of the automata respectively. Storing the lattice in the external memory banks ensures that the available FPGA logic resources do not limit the lattice size for simulations. Our implementation design as discussed in Chapter 3 is formulated around this layout which is important for numeric computations based accelerators.

In [92], Sloot et al. specify fluid simulations using cellular automata as one of the most successful practical application of cellular automata as computing devices, and in [97], Smith et al. report FPGA-based accelerators for computational fluid dynamics promise large improvement in sustained performance at better price-performance rations with

lower overall power consumption than conventional processors. Therefore, it is no surprise that number of FPGA-based accelerators for such simulations have been reported, for example, [66, 78, 86, 90, 91].

Kobori et al. [66], reported in a series of publications, results from their FPGA-based implementations for Lattice Gas Models. For a single FPGA implementation of a 3D FCHC Lattice Gas Model they demonstrate a 200 times speedup compared to a software version on a 1.8GHz Athlon processor. They used a parallel and a pipelined processing arrangement to improve both the computation and the FPGA resource utilisation depending on the available FPGA resources. Again, the implementations were non-numerical computations, that is, bitwise operations that demonstrated a speedup for a 3D lattice and an efficient usage of FPGA resources using pipelining over time. Our I/O bound implementations, as presented in Chapter 4, employ this pipelining over time with additional new features.

Nallatech, a commercial vendor for FPGA-based solutions, also reported a FHP-III lattice gas model implementation [91] using their DIME-C environment in 2005, where they reported a performance of 550 times faster than 1GHz Pentium processor which is equivalent to 100 times faster than 4GHz processor. They quote "LG are CA based and as such they are simple models that need to be repeated many hundreds, thousands or millions of times to get a useful result. This would ideally require a great number of relatively simple computers. The question is where such computers exist?" as their motivation behind their work and further explain their approach as "Conventional computers are ideal for the complex models but as inappropriate for the simple models. With RC, we have reverted to the purer ideas of computing where the computer is designed to perform a single task well and then redesign if it needs to perform another task. Thus we have in RC the ability to create multiple, simple, perfectly implemented cells. Each is capable of implementing the simple rules defined by the CA. Their sum total is capable of producing the complexity of results required."

All of the above mentioned works have focused on CA where the state space per cell is finite and therefore these are non-numeric computations. However, the computational structure of cellular automata, in general, matches the generic stencil based structures as also found in well known numerical techniques such as LBM [100] and in the FDTD algorithm [29] [27] that explicitly solve the time-dependent Maxwell equations. We can view LBM as a generalised CA because of their same computational structure as explained by [92], Nallatech also reported their FPGA-based LBM implementations [90] based on their previous FPGA-based CA work [91] in 2005. This work did not demonstrate any speedup, but was most likely the first work to report floating point based FPGA implementation for LBM and a progress from their FHP-III implementation. In the following year, for

2-dimensional time-dependent fluid dynamics problems, [86] reported their streaming accelerator implemented on a Virtex-4 FPGA with PCI Express x8 interface that achieved a speedup of 2.93 and 2.46 compared to a 3.4 GHz Pentium4 processor and a 2.2 GHz Opteron processor respectively. They reported their implementation methodology based on the optimisation of the equations of LBM and then formulating a streaming computation based on their stream compiler for FPGAs [75].

Finite difference or the FDTD algorithm are also numeric and stencil based computations and some of the their noted FPGA-based implementations are [28], [29], [36] and [37]. Using a fixed-point method implementation, [29] reported a 24 times speed-up of a two-dimensional FDTD and as per [36], though this method was not as accurate as floating point but showed tremendous speed-up over the software calculations. Using fixed-point methods for numeric computations like LBM is an interesting idea that could be implemented using an FPGA and validated against a software implementation for the relative errors.

One of the main aims of this project has been scalability, that is, to design a system that can be scaled from a single- to a multi-FPGA based implementation and therefore a capability to simulate large simulation sizes. In Chapter 7 we present a multi-FPGA based implementation using Maxwell- a 64 FPGA supercomputer [8] from EPCC at the University of Edinburgh.

Gokhale et al. define reconfigurable supercomputing as "combines programmable devices like FPGA with high performance microprocessors, all communicating over a high bandwidth, low latency interconnection network" [50] and presents in details the pros and cons of such systems. For the discussion on the feasibility of FPGA supercomputing see for example [35, 50, 87]. [8, 10, 11, 26, 76, 87] are some of the many research groups that have setup FPGA supercomputers. The main idea behind these machines is to run such applications where the control and I/O portion is executed by the microprocessors and the compute and data-intense portion is accelerated using FPGAs. [50] discusses the various commercially available FPGA supercomputing architectures and the more recent trend where the cluster is augmented with FPGA chips for application acceleration are discussed in [35]. Other than having a microprocessor based communication network, machines like Maxwell [8] also include an independent point-to-point link between the adjacent FPGAs thus enabling an FPGA only setup to perform parallel computations. A wide range of applications from the areas of finance [8, 26], medicine [8], petrochemical, cryptography [76], etc have been ported successfully to FPGA supercomputers, however, majority of such applications are trivially parallel as the Monto Carlo simulations [10] with limited data requirements.

2.5 Summary

This chapter introduces cellular automata in general and the various models that were implemented as a part of this work. It also presents reconfigurable platform like FPGA used to implement cellular automata accelerators and how to exploit the available features for overall performance. One of the key focus of exploiting memory hierarchy available in HPRC systems as a necessary feature in order to maximum overall system performance is also covered. Finally, other interesting works in the field of FPGA-based CA implementations and their influence on this work is also presented.