PERFORMANCE EVALUATION OF FPGA-BASED CELLULAR AUTOMATA ACCELERATORS

S. Murtaza, A.G. Hoekstra, P.M.A Sloot

Section Computational Science,
Institute for Informatics,
University of Amsterdam,
Amsterdam, The Netherlands
email: {murtaza|alfons|sloot}@science.uva.nl

ABSTRACT

The scientific community has been using FPGA-based computation engines as cellular automata (CA) accelerators for some time now. With the recent advent of more advanced FPGA logic it becomes necessary to better understand the mapping of CA to these systems. In this paper, we present a methodology to predict the performance of running such CA on specific FPGA hardware before engineering the design in reality. This will help to determine the optimal values for the various parameters that control the application and the given FPGA hardware specifications. The model is validated for two different types of two-dimensional CA algorithms.

1. INTRODUCTION

CA are decentralized spatially extended systems consisting of large numbers of simple and identical components with local connectivity [1]. Such systems have the potential to perform complex computations with a high degree of efficiency and robustness, as well as to model the behavior of complex systems from nature. CAs have been studied extensively in the natural sciences, mathematics and in 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 behavior in complex systems, see, e.g. [2]. 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, galaxy formation 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, e.g. [3–9].

With the application of CAs in such a wide diversity of fields it is not surprising that both the scientific community and industry have researched many possible implementation approaches, from massively parallel and distributed systems, to reconfigurable platforms such as FPGA-based systems [10–15].

Although the FPGA-based CA accelerator implementations have shown promising results (see e.g.[10, 11]), they have not been able to explicitly identify the various factors or parameters that were exploited to gain performance. A model to predict the performance as a function of parameters related to the FPGA technology and to the CA is needed. Not only does this helps to better understand the mapping process and to exploit the parameters optimally but also defines the limits which are to be expected.

In this paper, we present a model to evaluate the performance of a two dimensional CA executed on a specific FPGA hardware technology before engineering the design in reality. Instead of directly attacking the design and implementation of the accelerators using the traditional engineering mindset, a more scientific approach such as performance modeling as presented in this paper would provide more insight into bridging the gap between the application and the hardware implementation. Not only does this provide pros and cons for the said implementation but it also helps to determine the optimal values for the various parameters that control the application and the given hardware specifications.

The rest of the paper is organised as follows: Section two introduces a simple CA computation model using FPGAs and the efficiency for such a setup is defined. Section three takes the model further and explains the CA computation model for real applications with an assumption that their lattice size are larger than the FPGA capacity. The detailed strategy of the computation method for our CA accelerator implementation is presented in section four. Section five introduces the two specific CA algorithms that are used to validate our model and provides the performance model for their respective implementation. The implementation results with the optimal parameters defined by the application and the hardware technology for each of the CA algorithms are shown in section six. Finally, conclusions and the future work are discussed in section seven.

2. A GENERIC PERFORMANCE MODEL

FPGA based computation engines appear to be very attractive for CA algorithms, which consist of a uniform structure composed of many Finite State Machines, thus matching the inherent design layout of FPGA hardware.

Consider a basic setup where the CA lattice is small enough that each CA cell is processed using a dedicated processing element (PE). As the whole CA lattice completely fits into the FPGA space the resulting computing system is therefore simple. For CA computation, the whole lattice is downloaded to the FPGA from the host machine and is run for the required number of generations. Finally, the resulting CA generation is uploaded back to the host system.

Assuming the host system does all the required pre and post processing, the performance model for such a system is defined as follows. We assume that we want to compute *g* generations of a CA. When executed on a stand alone PC system this would take T_{pc} execution time. Using the PC-FPGA system the same computation takes T_{ft} execution time, and we write

$$T_{ft} = T_{fpga} + T_{fo} \tag{1}$$

where, T_{fpga} is the pure execution time on the FPGA, and T_{fo} is total overhead time to pre and post process the CA lattice and transfer time to move the data to the FPGA and back to the main memory. Note that in Eq. (1) we assume a serial implementation, that is, the useful computations (T_{fpga}) do not execute at the same time as the overhead work (T_{fo}). In many cases however such “latency hiding” is possible and then the model needs to be adapted slightly. We can now define the obtained speedup by running on the FPGA as

$$S = \frac{T_{pc}}{T_{ft}} = \frac{T_{pc}/T_{fpga}}{1 + T_{fo}/T_{fpga}} = \frac{S_{max}}{1 + f_o} \tag{2}$$

The maximum speedup S_{max} that can ever be obtained on the PC-FPGA system is T_{pc}/T_{fpga}. Only if the overhead time is zero this speedup will be obtained. For finite overhead times the decrease of maximum speedup is determined by a dimensionless number, the fractional overhead $f_o = T_{fo}/T_{fpga}$. Note that if f_o is small, Eq. (2) can be written as

$$S \approx S_{max}(1 - f_o) \tag{3}$$

We can also define an efficiency of the FPGA implementation as

$$\varepsilon = \frac{S}{S_{max}} \tag{4}$$

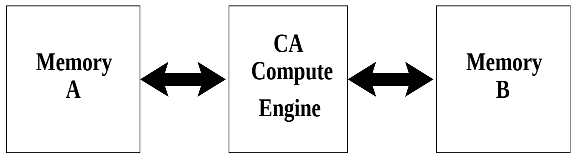

Fig. 1. Basic computation model.

The efficiency is a number between 0 and 1. We assume that the overhead consists of the time to preprocess the CA lattice on the PC (T_{pre}), the time to download the CA lattice from PC to FPGA on-board memory (T_s), the time to upload CA lattice from the FPGA on-board memory to the PC (T_r), and the time to post process the CA lattice on the PC (T_{pos}). With these definitions we can now write

$$T_{fo} = T_{pre} + T_s + T_r + T_{pos} \tag{5}$$

and the fractional overhead f_o can now be written as a summation of 4 different types of fractional overheads, i.e.

$$f_o = f_{pre} + f_s + f_r + f_{pos} \tag{6}$$

with $f_i = T_i/T_{fpga}$ and i $\in \{pre, s, r, pos\}$. The terms above depend on various parameters like clock frequency of the hardware, number of CA cells, number of CA generations computed etc. Note however that such model is only feasible when the whole CA lattice fits the given FPGA space.

For real CA applications the lattice is large (typically 128^3 cells or more). In the sequel we assume that the lattice is larger than the FPGA capacity but still fits the memory banks available on the FPGA board. Therefore, the resulting computing system is composed of a host machine for pre and post CA processing with an FPGA board connected as a co-processor having on-board multiple memory banks. The FPGA runs the CA compute engine and the on-board memory banks store the CA lattice computation. The two (preferable independent) memory banks store two consecutive automaton states similar to the CAREM processor [10] as shown in Fig. 1.

The CA compute engine (CE) has *n* compute blocks (CB) as shown in Fig. 2 and uses memory banks A and B alternatively as source and destination memory to hold the CA lattice. Further, each CB is composed of *k* processing elements (PE) where *k* is the number of the CA cells that the CE is able to read from the source memory in parallel. (Note that each CB has to store 3 columns i.e. left, middle and the right column in order to compute the next state for the middle column). Each PE implements the CA transition function as shown in Fig. 3.

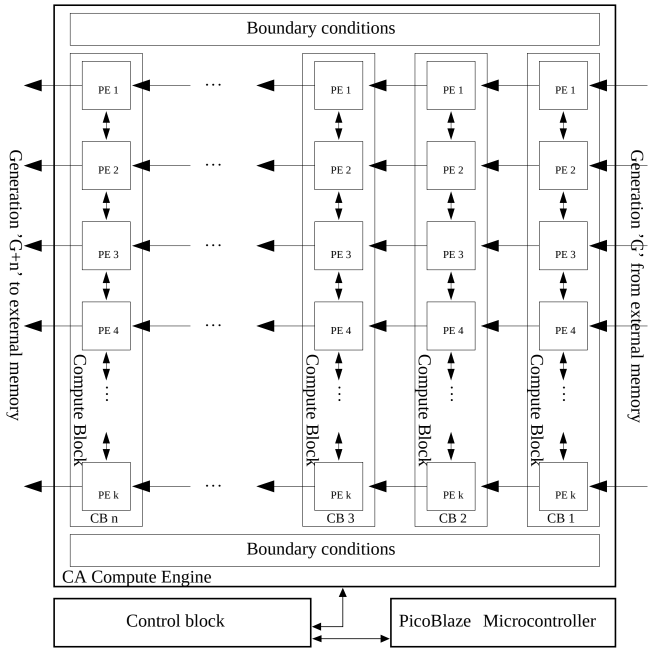

Fig. 2. 2D CA block diagram

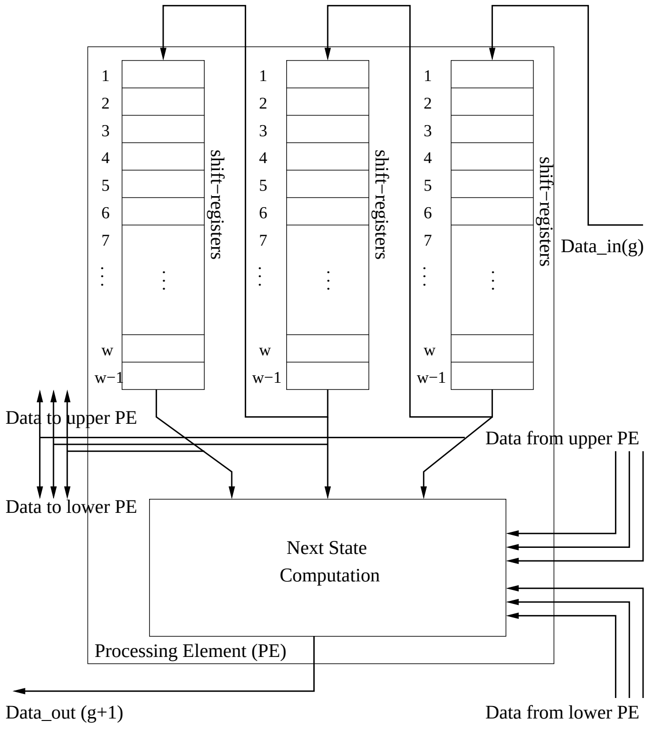

Fig. 3. Processing Element.

3. TWO DIMENSIONAL CA ON FPGA

We will now restrict ourselves to two dimensional CA, and apply an algorithm based on pipelining over CA generations and alternate use of memory banks as source and destination memory, as proposed by Kobori[11] and Cappuccino[10] respectively.

Based on the computation model introduced above, with the CE having only a single CB, the CE first reads the state of *k*-cells from the source memory bank. After the CB computes the next state of *k*-cells, the CE finally writes out this new state into the destination memory bank. To further improve the computations, instead of having only a single stage CE engine where only a single generation is computed, we can have multiple CB's connected in a pipeline as shown in Fig. 2. This pipelined CE results in the computation of *n* generations in a single sweep (i.e. whole CA lattice from source memory is passed through the CE for computations and computation results are stored in destination memory) where *n* is the number of CB's connected together. This pipelined model has been successfully implemented by Kobori [11] but without alternating the roles of the two memory banks as source and destination memory after every single sweep.

Provided the two memory banks are completely independent and with the multiple CB's pipelined model, the CE reads *k* cells from source memory, computes $n \times k$ cells and writes *k* cells to the destination memory in parallel.

The CA hardware implementation was done using the Spartan-3 starter kit board. The CA algorithm implementations in FPGA logic and related performance modeling are based on designs implemented using this board. VHDL was used to describe the behavior and structure of the algorithms. Further, VHDL code was compiled and synthesized using Xilinx design tools.

The Spartan-3 board has one XC3S200 FPGA, and two 512KB SRAM banks. The two SRAM banks are not independent as both of them are addressed using common pins from the FPGA, therefore in this case the CA compute engine is not able to read and write memory banks simultaneously.

The Game of Life and the Lattice Gas HPP model were chosen as the CA algorithms that we implemented for this benchmark. They are both very simple CA's with a one-bit and a four-bits state per cell respectively, but they suit our goal of validation of our performance model. The next step will be to implement more advanced CA's, simulating e.g. hydrodynamics or tumor growth.

The FPGA holds the pipelined CE implementation as well as the Xilinx PicoBlaze softcore processor (PB). The PB is used to provide the necessary interface that is required to (a) initialize the source memory bank, (b) retrieve the computed results from the destination memory bank, (c) start the CE, and (d) provide a serial connection to the hyperterminal running on the host machine. With the FPGA configured, the user initiates the process of initializing source memory via PB, next the CE is started to run for a specified number of generations. When the CE is done with the computations it signals the PB accordingly. The CA computation engine alternatively reads out the data from one of the SRAM banks and writes out the computed results to the other SRAM bank.

4. DETAILED PERFORMANCE MODEL

To explain the detailed strategy of the computation method for our CA accelerator implementation as shown in Fig. 2, consider *x* to be the number of columns and *y* (= *k*, to simplify explanation) is the number of rows of the CA lattice. With this setup, the FPGA will read the whole column (since we assumed *y* = *k*) in parallel say in time R_d. With boundary conditions along the top and bottom edge of the lattice available within the *y*-cells itself, the single CB will output exactly *y*-cells with correct computed next state. With the FPGA engine able to read (and write in parallel as well) *k*-cells per column in time R_d and with C_m being the PE's compute time, as long as $R_d \geq C_m$ (which is true for algorithms like Game of Life that we implemented), the execution times are completely I/O bound. Therefore, the time to compute *x* columns is just the time needed to read them into the FPGA: $T_R = xR_d$.

However, if the boundary condition along the sides of the lattice are not available at the start of the computation, the CE has to re-read the first two columns of the lattice once it is done with reading the *x* columns, in order to correctly compute the next state of the *y*-cells of the last and the first column of the lattice. This results in an overhead of re-reading two columns when the number of CB=1 in the CE. If hard boundary conditions are considered then this overhead is zero.

If we further pipeline the CE with *n* CB's, we still get *y*-cells with correct output but the overhead of re-reading columns due to boundary conditions go up by *2n*. Then, the overhead time is $T_{RO} = 2nR_d$.

Moreover, with *n* CB's, it requires some time to fill the pipeline before the CE starts writing out computed data. This start up time is $T_{Wr} = (n+1)R_d$.

Therefore, for I/O bound CA simulations with *y*=*k*, the total execution time T_E to compute *n* generations is

$$T_E = T_R + T_{RO} + T_{Wr} = (x + 3n + 1)R_d \qquad (7)$$

With alternate use of memory banks as source and destination memory, we can compute *g* generations simply by reusing the CE that has *n* CB's only by swapping the role of memory banks as required. Therefore, if the required generation is *g*, such that to compute it, it requires to sweep the whole CA lattice through the CE *b* times where $b = \frac{g}{n}$, then the total execution time is simply *b* $\times T_E$.

Now, let's consider a more realistic scenario with *k* < *FPGA internal memory* $\leq$ *y* as shown in Fig. 4, we can use

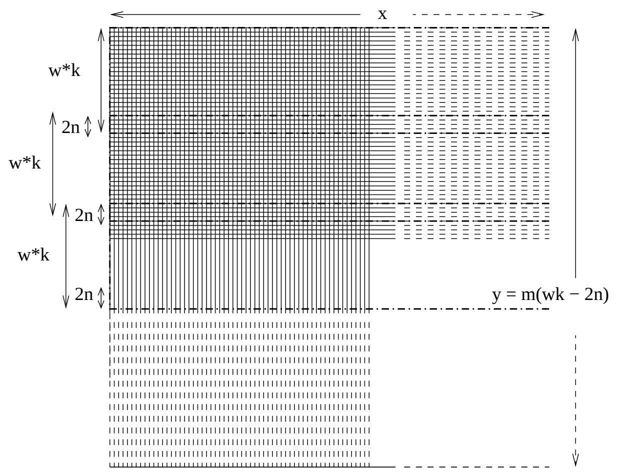

Fig. 4. CA lattice with y >> k .

the available internal memory within the FPGA to buffer *w* $\times$ *k* cells within each CB. This enables us to store the data for *w* computational planes as shown in Fig. 4 during CA computation. (Note: the three internal buffers of every PE, now have a capacity of *w* each).

Since *y* >> *w*, CE no longer has the boundary conditions available along the two edges of the *w*$\times$*k* cells wide computational plane. Therefore, each CB outputs the two cells along the two edges with wrong states as shown in Fig. 5. With the *n* CB pipelined engine, the CE outputs *2n* cells

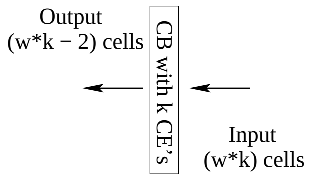

Fig. 5. Loss due to boundary conditions.

with wrong states and this results in the overlap of computation planes along x-axis that require to be recomputed in order to get correct results. Therefore, as shown in Fig. 4, to compute *n* generations i.e. sweeping the CA lattice once through the CE, the CE has to sweep the whole CA lattice from the source to destination memory in *m* computational planes where *m* is

$$m = \frac{y}{wk - 2n} \qquad (8)$$

With this setup, to compute *g* generations, the total execution time is now

$$T_E = bmw(x + 3n + 1)R_d \qquad (9)$$

Substituting for *b* and *m* in (9), we get

$$T_E = \frac{gyw(x + 3n + 1)R_d}{n(wk - 2n)} \qquad (10)$$

Next we try to find an expression for the optimal value of *n*, i.e. the depth of the pipeline. Rewriting Eq. (10), we have

$$T_E = \frac{gwR_d}{n(wk-2n)}[xy + y(3n+1)] \qquad (11)$$

since we can safely assume that $x >> 3n+1$, we can reduce Eq. (11) to

$$T_E = \frac{gwxyR_d}{n(wk-2n)} \qquad (12)$$

Eq. (12) is minimized by taking *n* equal to

$$n_{optimal} = \frac{wk}{4} \qquad (13)$$

Substituting $n_{optimal}$ for *n* in Eq. (12), we get

$$T_E = \frac{8gxyR_d}{wk^2} \qquad (14)$$

Thus confirming the intuitive expectation that a FPGA with larger internal memory or capacity to hold more logic and improved I/O interface between the FPGA and on-board memory banks improves the execution time accordingly. Eq. (14) also states explicitly how the parameters defined by the implementing technology can impact the performance of the CA accelerator.

5. TEST BENCHES

To demonstrate the proposed performance model for two dimensional CA's, we implemented and validated our model for two different CA algorithms: The Game of Life and the HPP lattice gas automata.

5.1. The Game of Life

The Game of Life is the most famous CA algorithm and is a simple model for simulating artificial life. The automata is implemented on a square lattice where each lattice site is either *dead* or *alive*, therefore, a single bit state machine is used to represent each lattice site.

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 neighborhood. The update rules are (1) an alive site with either two or three alive neighbor sites remains alive, (2) a dead site with three alive neighbor sites becomes alive, and (3) the state of the rest of the sites remains unchanged.

Considering a 1024x1024 the Game of Life lattice that requires to be computed for 512 generations. The performance model for such an implementation as per Eq. (12) has parameters *x = y = 1024, g=512* and these values are defined by the application. Rest of the parameters for Eq. (12) are specified by the FPGA implementation technology i.e. *k*= 16, is the number of cells CE can read from source SRAM, *w* and *n* depend on the available logic space within the FPGA chip and R_d is determined by the hardware clock frequency. With varying values for *n* and *w* in Eq. (12), the respective execution times for the Game of Life are as shown in Fig. 6.

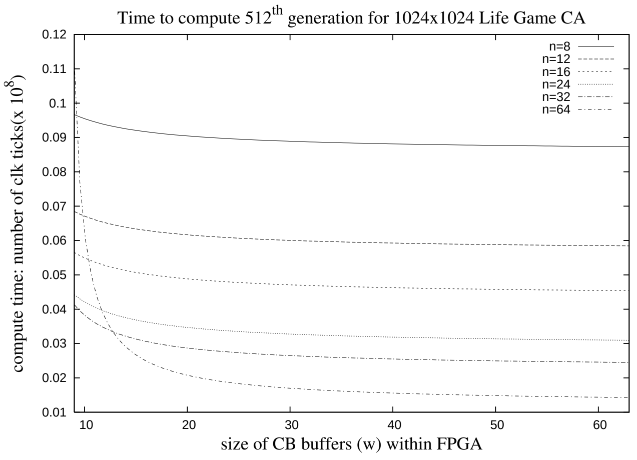

Fig. 6. Performance model to compute 512^{th} generation of The Game of Life.

5.2. Lattice gas automata: The HPP model

Lattice gas automata (LGA) [16] are a relative novel method to simulate the hydrodynamics of incompressible fluids. The flow is modeled by particles which reside on nodes of a regular lattice. The extremely simplified dynamics consists of streaming step where all particles move to a neighboring lattice site in the direction of their velocities, followed by a collision step, where different particles arriving at the same node interact and possibly change their velocity according to collision rules. The main feature of the model are exact conservation laws, unconditional stability, a large number of degrees of freedom, intrinsic spontaneous fluctuations, low memory consumption, and the inherent spatial locality of update rules, making it ideal for parallel processing [1, 17].

The HPP model was the first LGA model invented and introduced by Hardy, Pomeau and de Pazzis in 1973. In this model the particles are restricted to move on the links of the square lattice and the motion is emerged in discrete time steps. Each particle travels at a unit speed i.e. moves from one lattice site to a neighboring site in each time step. 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 Fig. 7(a) along with possible coding of the 4-bit state machine. The particles collide as per the rules as shown in Fig. 7(b) conserving both the mass and momentum for the system.

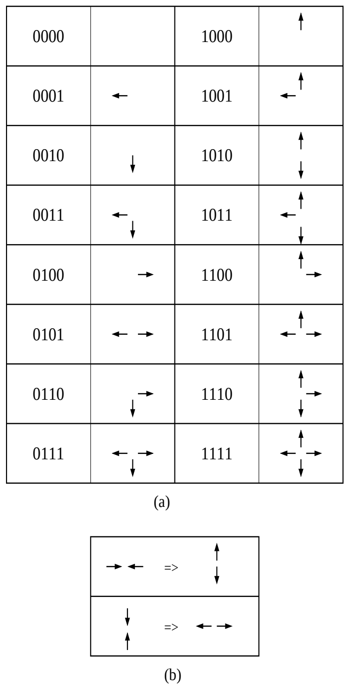

Fig. 7. The HPP (a) configurations, (b) collision rules.

Similar to the Game of Life computation, if we again consider a 1024x1024 square lattice for the HPP model that requires to be computed for 512 generations, all the variables defined by underlying FPGA implementation technology remain the same except *k*=4, which is the number of cells CE can read from source SRAM. Again, with varying values for *n* and *w* Eq. (12) for the HPP model, the respective execution times for the model behave somewhat similar to the Game of Life one as shown in Fig. 6.

6. RESULTS

The Spartan-3 board has two SRAM chips, which are however not completely independent. Therefore, we had to read and write to the source and destination memory alternatively from the CE running on the FPGA. Since the Spartan-3 board runs with a maximum clock frequency of 50MHz and the available asynchronous SRAM chips on board have access time $\leq$ 10ns, this allowed us to read, compute and write in two clock cycles. This implies that R_d is 2 $\times$ 20ns for our system implementation.

Specific to our Spartan-3 board, for the Game of Life implementation, *k=16* and the maximum logic that was possible to fit the underlying FPGA chip was with *n=16* and *w=9*. These values also resulted in the best possible computation time for the available FPGA hardware resources for the given setup as shown in Fig. 8.

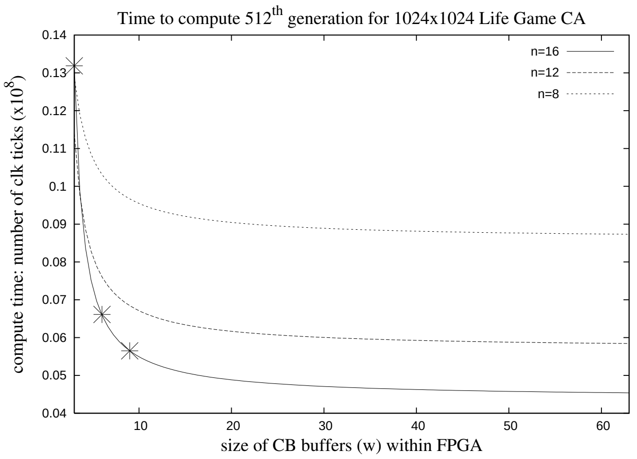

Fig. 8. Performance model to compute 512^{th} generation for The Game of Life. Points are the measured execution times and specify the best possible values for *w* and *n* for implementation using Spartan-3 board

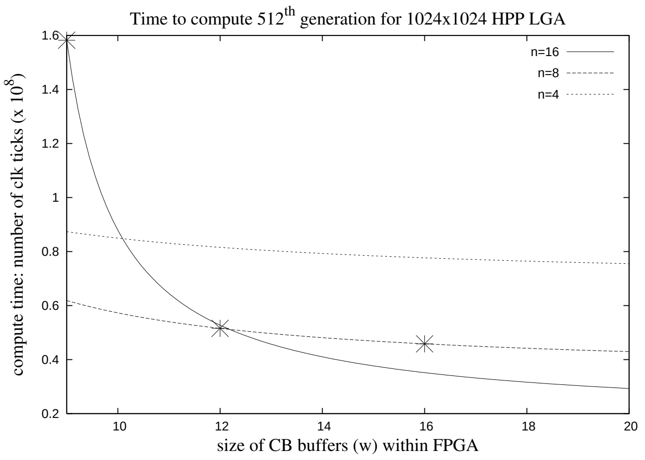

Fig. 9. Performance model to compute 512^{th} generation for The HPP model. Points are the measured execution times and specify the best possible values for *w* and *n* for implementation using Spartan-3 board

For the HPP model implementation, Spartan-3 board defines *k=4* and the maximum logic that was possible to fit the underlying FPGA chip was again with *n=16* and *w=9*. But, for this case these values for *n* and *w* respectively result in the worst computation time for the given setup. The best computation time was achieved with *n=8* and *w=16* as shown in Fig. 9.

7. CONCLUSION AND FUTURE WORK

In this paper, we presented a model to evaluate the performance of a two dimensional CA executed on a typical Spartan-3 FPGA board. We explicitly showed how the various parameters defined by the FPGA technology control the CA accelerator engine that was implemented for the Game of Life and the HPP model. Using this approach we were able to predict the performance of the algorithm, for a specific FPGA technology, before the algorithm was implemented and run in reality. We are now able to exploit the parameters optimally. As a result of this, the FPGA hardware resources are used in a way that enable the implementation of the fastest possible accelerator, for the application, using specific technology. Further, we also improved the pipelined CA computation engine with alternate uses of memory banks as source and destination memory.

In the future, we will look into the data transfer between the host machine and the FPGA, and include the measurements in our speedup equations. We will also investigate into compute bound computations i.e. CA's with complex rules (e.g. using floating points). Advanced hardware platform with V4/V5 Xilinx FPGA's will be used for implementations. We will further investigate the three dimensional CA's, running real CA application like hydrodynamics or tumor growth.

References

[1] P. M. A. Sloot and A. G. Hoekstra, "Modeling Dynamic Systems with Cellular Automata," in *Handbook of Dynamic System Modeling, Editor: Paul A. Fishwick*, 2007.

[2] P. M. A. Sloot, B. J. Overeinder, and A. Schoneveld, "Self-organized criticality in simulated correlated systems," *Computer Physics Communications*, vol. 142, pp. 76–81, 2001.

[3] A. Ilachinski, *Cellular Automata: A Discrete Universe*. World Scientific, 2001.

[4] B. Chopard and M. Droz, *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 1998.

[5] A. Deutsch and S. Dormann, *Cellular Automaton Modeling of Biological Pattern Formation*. Birkhauser, 2004.

[6] H. A. Gutowitz, *Cellular Automata*. MIT Press, 1990.

[7] P. M. A. Sloot, B. Chopard, and A. G. Hoekstra, *Cellular Automata: 6th International Conference on Cellular Automata for Research and Industry, ACRI 2004*. Springer Verlag, Oct. 2004.

[8] P. M. A. Sloot and A. G. Hoekstra, "Cellular Automata as a Mesoscopic Approach to Model and simulate Complex Systems," in *Lecture Notes in Computer Science: Springer Verlag*, 2001, p. 518.

[9] P. M. A. Sloot, J. A. Kaandorp, A. G. Hoekstra, and B. J. Overeinder, "Distributed Cellular Automata: Large Scale Simulation of Natural Phenomena," in *Solutions to Parallel and Distributed Computing Problems, Lessons from Biological Sciences*, 2001, pp. 1–46.

[10] G. Cappuccino and G. Cocorullo, "Custom Reconfigurable Computing Machine for High Performance Cellular Automata Processing," in *TechOnLine Publication*, July 2001.

[11] T. Kobori, T. Maruyama, and T. Hoshino, "A Cellular Automata System with FPGA," in *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001, pp. 120–129.

[12] M. Bubak, J. Moscinski, and R. Slota, "Parallel program for 2D FHP Lattice gas simulation on clusters of workstations," in *10th Summer School on Computing Techniques in Physics: HPC in Science*, 1994.

[13] T. Toffoli and N. Margolus, *Cellular Automata Machines*. MIT Press, 1987.

[14] P. Shaw, P. Cockshott, and P. Barrie, "Implementation of Lattice Gases Using FPGAs."

[15] D. Shand, D. Denning, and R. Chamberlain, "Lattice Gases - Simple Models of Complex Fluid Dynamics," www.Nallatatech.com, 2005, White Paper, NT309-0001.

[16] J. P. Rivet and J. P. Boon, *Lattice Gas Hydrodynamics*. Cambridge University Press, 2001.

[17] D. Dubbeldam, A. G. Hoekstra, and P. M. A. Sloot, "Dynamic structure factor in single- and two-species thermal GBL lattice gas," *Computer Physics Communications*, vol. 129, pp. 13–20, 2000.