High performance reconfigurable computing with cellular automata

Murtaza, S.

Publication date
2010

Chapter 4

I/O Bound CA on FPGA*

In this chapter CA algorithms like the Game of Life and the Lattice Gas HPP model, as introduced in Chapter 2, are of our interest. Next state computation for these types of CAs are a few bitwise operations, implying each cell update lasts a few hardware clock ticks and the PE implementation requires minimal hardware resources.

Considering these two main factors, their hardware implementation has to ensure that input and the output data streams into the CA engine match up with the engine's processing speed and the optimal utilisation of the FPGA resources. Therefore, once a CA algorithm is categorised as I/O bound using a formulation as presented in Section 3.3, the next step is to formulate a hardware design that ensures the maximum utilisation of FPGA resources, and maximises the number of CA iteration computations per a complete data stream flow from source to destination memory bank via FPGA. Standard techniques like pipelining and memory hierarchy can be employed to improve the overall system organisation.

This chapter presents the FPGA-based implementations for the I/O bound CA test benches, that is, the Game of Life and the HPP lattice gas. Various hardware algorithms employed to improve the overall system performance are discussed in detail. System parameters like depth of pipelining, memory storage, bandwidth etc and how they define an optimal design are also discussed in detail. The chapter concludes with some results and possible future extensions.

*This chapter is based on the following publications:

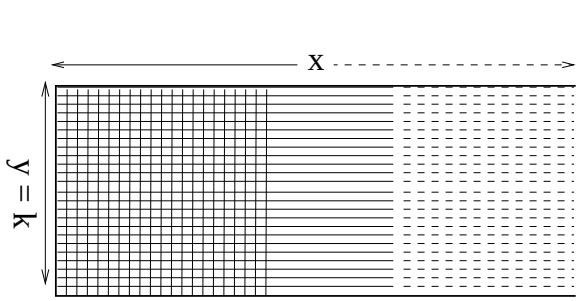
- S. Murtaza, A.G. Hoekstra, and P.M.A. Sloot, "Performance modeling of 2D Cellular Automata on FPGA," in *17th International Conference on Field Programmable Logic and Applications (FPL'07)*, pp. 74–78. IEEE, Amsterdam, August 27–29, 2007.
- S. Murtaza, A.G. Hoekstra, and P.M.A. Sloot, "Performance evaluation of FPGA-based Cellular Automata accelerators," in *Proceedings of the Third Annual Reconfigurable Systems Summer Institute (RSSI'07)*, (on line proceedings) +7. Reconfigurable Systems Summer Institute, National Center for Supercomputing Applications, Urbana, Illinois, July 17–20, 2007.

4.1 I/O Bound 2D CA on FPGA

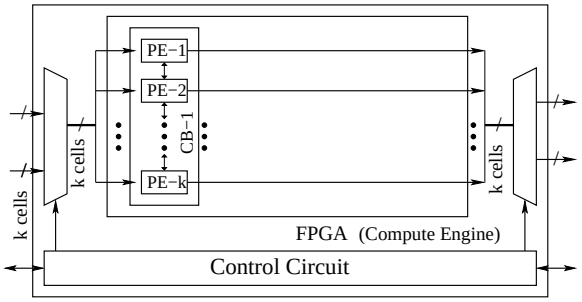
The Game of Life and the Lattice Gas HPP model as introduced in Chapter 2, are both very simple CAs with one-bit and a four-bits state per cell respectively. Since their computational structure is simple, the FPGA resource utilisation per PE is quite low (that is, few logic gates to compute the next state and few memory elements for state storage). Given the fixed bandwidth between the FPGA and the memory banks, and the low FPGA resource utilisation per PE, a simple implementation would read k cells from the source memory into the FPGA, and have equal number of PEs implemented in the FPGA for parallel computations. As shown in Figure 4.1, a k PE implementation to compute k cells in parallel make a compute block (CB). Though this design layout reads, computes and writes k cells in parallel respectively, the FPGA resources are massively under utilised. In order to improve both the computation and the FPGA resource utilisation, a number of CBs are arranged in a pipeline as proposed by [65] and as shown in Figure 4.2. The number of CB implementations also depends on the available FPGA resources as explained in Section 4.1.2.

4.1.1 Basic Computation Model

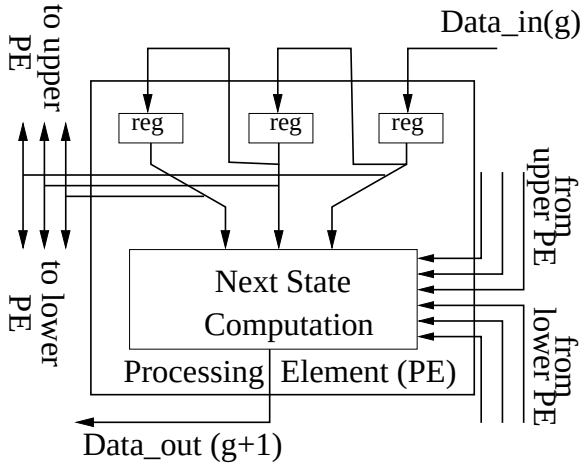
In order to explain the detailed strategy of the computation method for our I/O bound CA accelerator implementation as shown in Figure 4.2, consider a simple scenario with a CA grid consisting of x columns and y ($= k$, FPGA-memory bandwidth) rows as shown in Figure 4.1(a) with periodic boundary conditions (boundary conditions are discussed in section 2.1.1, Figure 2.2). A simple implementation with a single CB as shown in Figure 4.1, reads and computes a column (which is equal to k cells) in parallel. For a CB to compute a column implies, performing a collision and propagation within the FPGA, and to do this each CB has to store three consecutive columns, that is, left, middle, and the right column to compute the next state for the middle column. Therefore, each PE implementing the CA collision and propagation function are as shown in Figure 4.1(c). With each clock tick, a PE updates a single cell. In order to update a cell, a PE: a) stores three consecutive cells of a row, where the one in the middle is to be updated, b) receives the required upper three neighbouring cells from the neighbouring top PE, and c) receives the required lower three neighbouring cells from the neighbouring lower PE. And with all of the information available, the PE updates a cell and flushes out it's updated state. For this setup, lets assume, the FPGA reads the whole column in parallel in time τ_{r}. Since the boundary conditions along the top and bottom edge of the lattice are available within the y cells, the single CB outputs 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 τ_{r}, the



(a) CA lattice.



(b) I/O bound Compute Engine having one Compute block, that is, k-PEs



(c) PE without any buffering

Figure 4.1: *I/O bound FPGA-CA mapping (basic scenario).*

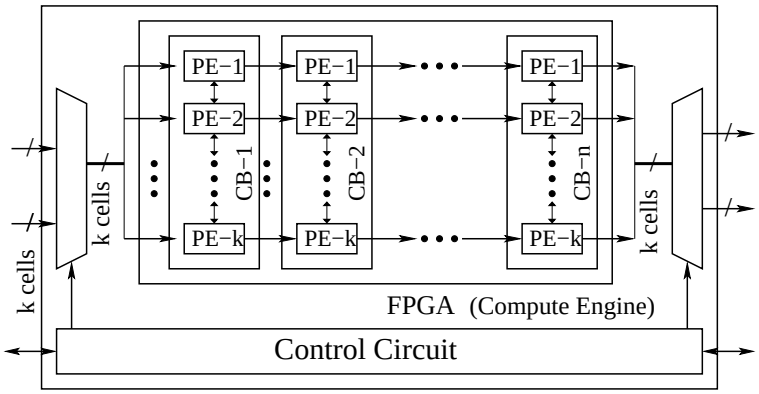


Figure 4.2: *I/O bound system: I/O bound implementation extends the basic computation model with single CB as shown in Figure 3.2 with n CB pipeline.*

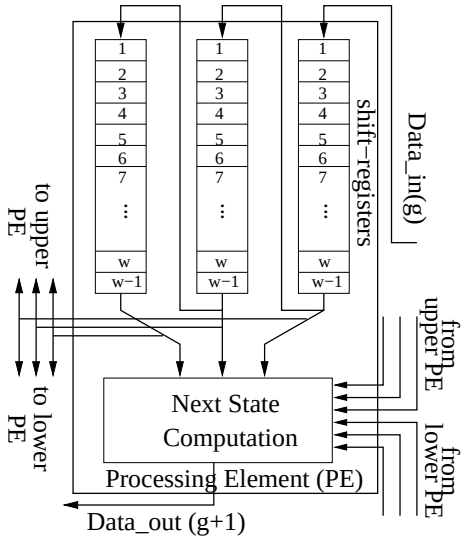


Figure 4.3: *I/O bound influenced PE implementation: PE stores a portion (depending on the available FPGA resources) of the CA lattice's three consecutive columns i.e. left, middle and the right column in order to compute (collision function) the next state for the middle column. With the completion of collision function, PE propagates the update state to its two neighbouring PEs within the same compute block as itself.*

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: $x \times \tau_{\mathrm{r}}$.

However, if the boundary conditions along the sides of the lattice are not available at the beginning of the computation (which is true for periodic boundary conditions), the CE has to re-read the first two columns of the lattice once it is done 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.

4.1.2 Pipelined Computation Model

Based on the computation model presented in the previous section, that is, the CE with a single CB (similar to Figure 3.4, upper panel), 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. Unlike Figure 3.4, lower panel, where the FPGA space is under utilised with some PEs sitting idle, we can further improve the computations and FPGA hardware resources by implementing multiple CBs in the CE. Instead of using only a single stage CE engine where only a single generation is computed, we can use multiple CBs connected in a pipeline as shown in Figure 4.2. This pipelined CE results in the computation of n generations in a single sweep (that is,

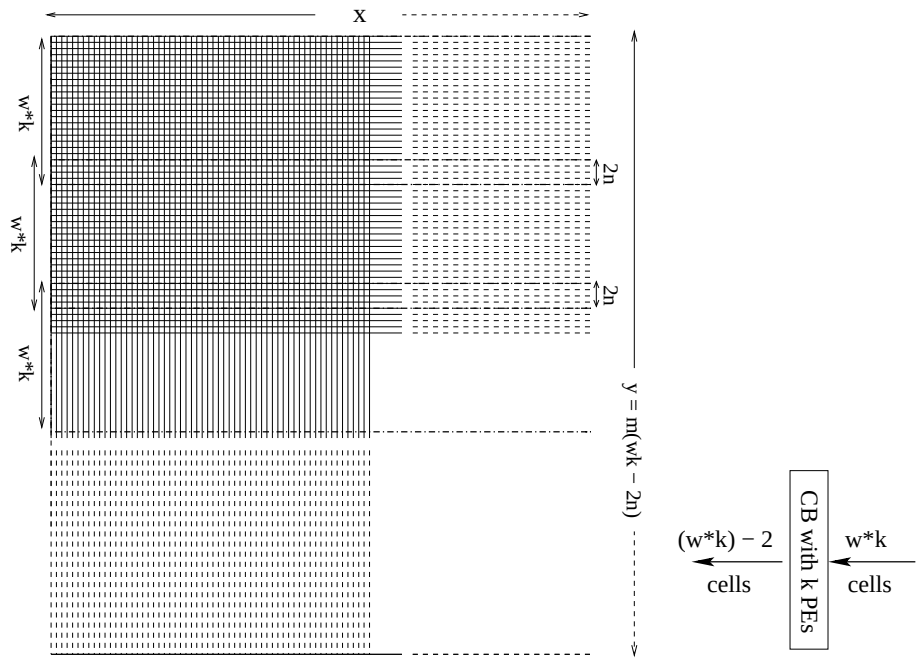


Figure 4.4: *(left) CA lattice with $y \gg k$, (right) Loss due to boundary conditions.*

the whole CA lattice from source memory is passed through the CE for computations, and the computation results are stored in destination memory) where n is the number of CBs connected together. This pipelined model has been successfully implemented by [65] but without alternating the roles of the two memory banks as source, and the destination memory after every single sweep.

With a pipelined CE employing n CBs, we still get y cells with correct output but, the overhead of re-reading columns due to boundary conditions along the sides of the lattice goes up by $2n$. And the overhead time is $2n\tau_{\mathrm{r}}$.

Moreover, with n CBs, it requires some time to fill the pipeline before the CE starts writing out computed data. This startup time is $(n+1)\tau_{\mathrm{r}}$.

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

$$T_{\mathrm{i}} = (x + 3n + 1)\tau_{\mathrm{r}} \tag{4.1}$$

With alternate use of memory banks as source and destination memory, we can compute g generations simply by reusing the CE that has n CBs only by swapping the role of memory banks as required. So to compute generation g, it requires sweeping the whole CA lattice through the CE b times where $b = \dfrac{g}{n}$, then the total execution time is simply $b \times T_{\mathrm{i}}$.

4.1.3 Pipelined Computation Model with Internal Buffers

Next we consider a more realistic scenario with $k <$ FPGA internal memory $\leq y$ as shown in Figure 4.4 (left). Now, we can use the available internal memory within the FPGA to buffer $w \times k$ cells within each CB (each CB has k PEs as shown in Figure 4.2 and the three internal buffers of every PE now have a capacity of w each as shown in Figure 4.3). This enables us to store the data for w computational planes as shown in Figure 4.4 during CA computation.

As $y \gg w$, the 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 Figure 4.4 (right). With the n CB pipelined engine, the CE outputs $2n$ cells with wrong states and this results in the overlap of computation planes along x-axis that need to be recomputed in order to get correct results. As shown in Figure 4.4, to compute n generations, that is, 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

$$m = \frac{y}{wk - 2n}. \tag{4.2}$$

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

$$T_{\mathrm{i}} = bmw(x + 3n + 1)\tau_{\mathrm{r}}. \tag{4.3}$$

Substituting for b and m, we get

$$T_{\mathrm{i}} = \frac{gyw(x + 3n + 1)\tau_{\mathrm{r}}}{n(wk - 2n)}. \tag{4.4}$$

4.1.4 Optimal Design

The number of system parameters like k, n, w, etc are now identified. How these parameters are related, helps to describe the optimal system implementation. Next we try to find an expression for the optimal value of n, that is, the depth of the pipeline. Since we can safely assume that $x >> 3n + 1$, we can reduce Equation (4.4) to

$$T_{\mathrm{i}} = \frac{gwxy\tau_{\mathrm{r}}}{n(wk - 2n)}. \tag{4.5}$$

Equation (4.5) is minimised by taking n equal to

$$n_{\mathrm{optimal}} = \frac{wk}{4}. \tag{4.6}$$

Substituting n_{optimal} for n in Equation (4.5), we get

$$T_{\mathrm{i}} = \frac{8gxy\tau_{\mathrm{r}}}{wk^2}. \tag{4.7}$$

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. Equation (4.7) also states explicitly how the parameters defined by the implementing technology can impact the performance of the CA accelerator.

The CE implementation based on the pipelined model having n CBs, and alternating the roles of the two memory banks as source and destination memory after every single sweep (provided the two on-board memory banks are completely independent), reads k cells from source memory, computes $n \times k$ cells and writes k cells to the destination memory in parallel. With the FPGA configured, the user initiates the process of initialising the source memory, next the CE is run for a specified number of generations. The CE alternatively reads out the data from one of the on-board memory banks and writes out the computed results to the other memory bank. When the CE is done with the computations, it signals the host accordingly.

4.2 Test Cases

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

Consider a 1024x1024 Game of Life lattice that requires to be computed for 512 generations. The performance model for such an implementation as per Equation (4.5) has parameters $x = y = 1024$, $g=512$ and these values are defined by the application. The rest of the parameters for Equation (4.5) are specified by the FPGA implementation technology, that is, the number of bit-wide Game of Life cells the CE can read from source SRAM specifies k= 16 (see Appendix 10 for Spartan-3 FPGA-board details), w and n depend on the available logic space within the FPGA chip and τ_{r} is determined by the hardware clock frequency. With varying values for n and w in Equation (4.5), the respective execution times for the Game of Life are as shown in Figure 4.5.

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 four-bit wide HPP cells CE can read from source SRAM (see Appendix 10 for Spartan-3 FPGA-board details). Again, with varying values for n and w Equation (4.5) for the HPP model, the respective execution times for the model behave somewhat similar to the Game of Life model as shown in Figure 4.5.

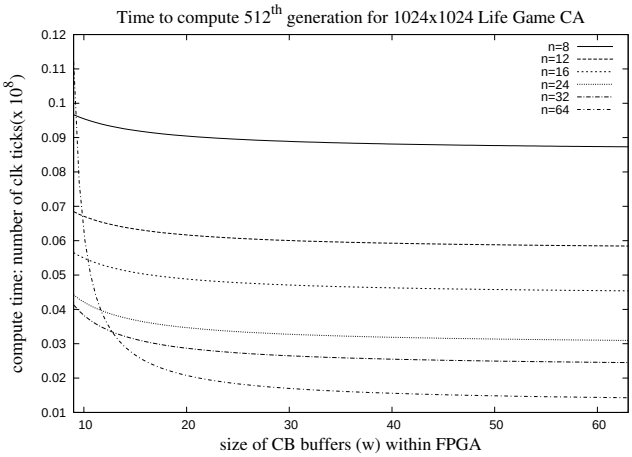


Figure 4.5: *Performance model as specified by Equation (4.5) to compute* 512^{th} *generation of The Game of Life. Each curve represents the execution time for a given number of pipeline stages (that is, number of CBs* $n = 8, 12, 16, 24, 32, 64$ *respectively) as a function of internal buffer size of a pipeline stage or CB.*

4.3 Results

The Spartan-3 board (for details see Appendix 10) 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 permitted us to read, compute and write in two clock cycles, implying that τ_r is 2×20ns 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 Figure 4.6.

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 non-optimal computation time for the given setup. The best computation time was achieved with *n=8* and *w=16* as shown in Figure 4.7.

4.4 Conclusion and Future Work

The two dimensional I/O bound CA implementations presented in this chapter were based on the combination of hardware algorithms as proposed by [25, 65] respectively. Other than combining the best design features from [25, 65] implementations, our work presents a

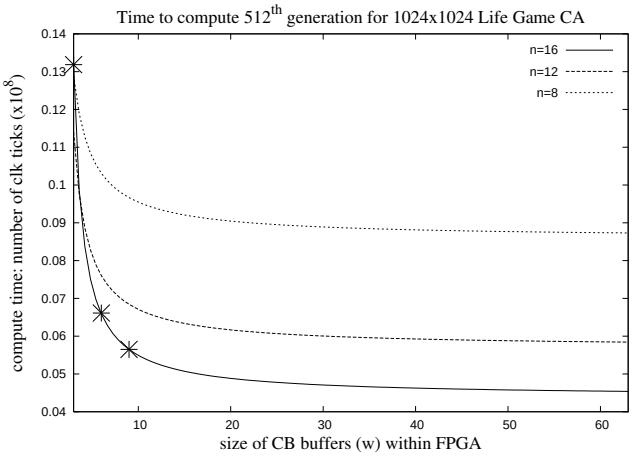


Figure 4.6: *Execution time to compute* 512^{th} *generation for The Game of Life. Curves are the performance model as specified by Equation (4.5)and the points are the measured execution times and specify the best possible values for w and n for implementation using Spartan-3 board (see Appendix 10 for Spartan-3 board details).*

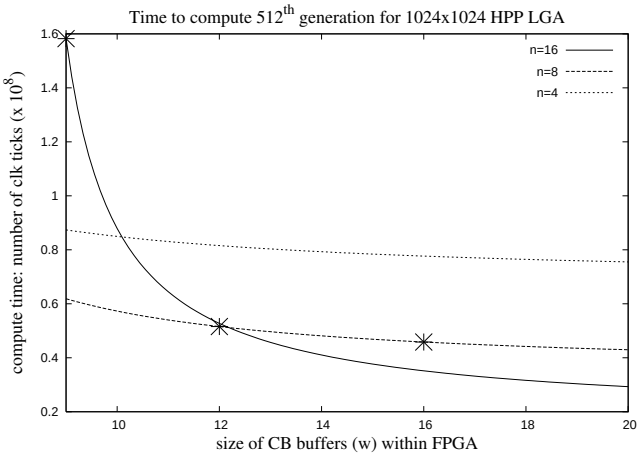


Figure 4.7: *Execution time to compute* 512^{th} *generation for The HPP model. Lines are the performance model as specified by Equation (4.5) and the points are the measured execution times and specify the best possible values for w and n for implementation using Spartan-3 board.*

model to evaluate the performance of a two dimensional CA executed on a typical Spartan-3 FPGA board. The performance model explicitly showed how the various parameters defined by the FPGA technology control the CA accelerator engine implemented in the Game of Life and the HPP model. Using this approach we were able to predict the performance of the algorithm and exploit the parameters optimally for a specific FPGA technology, long before the algorithm was implemented and run in reality. 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. The pipelined CA computation engine as proposed by [65] was further improved by a) alternate uses of memory banks as source and destination memory, b) the parameters that define the number of pipeline stages as specified in Equation (4.6) and categorise the overall hardware design as a scalable algorithm for I/O bound CA implementations.

Possible future extensions are: to look into the data transfer between the host machine and the FPGA, and include the measurements in the given speedup equations; to use advanced hardware platform with V5 Xilinx FPGAs with multiple independent memory banks for implementations; and to further investigate three dimensional I/O bound CAs.