High performance reconfigurable computing with cellular automata

Murtaza, S.

Publication date
2010

Chapter 3

Performance Modeling of FPGA based CA Implementation*

The FPGA based computation engines appear to be very attractive for CA algorithms as CAs consist of a uniform structure composed of many finite state machines, thus matching the inherent design layout of FPGA hardware. The computational structure of CAs in general, resembles the generic stencil based structures also found in, for example, finite difference of FDTD algorithm. As per [29] FPGA based FDTD implementations have shown a significant performance gain.

However, there are many trade-offs to consider when designing and implementing the FPGA based CA accelerator. For the best possible mapping of CA algorithm to the FPGA logic space, the most important aspect is to fully understand the behaviour of the specified CA algorithm in terms of its execution times, which is either compute or I/O bound.

Performance modeling is a highly popular and commonly used technique in high performance computing [46]. In contrast, there is hardly published work that demonstrates the application of performance modeling techniques in high performance computing using FPGAs. This chapters introduces the application of performance modeling techniques for FPGA based CA computations. The chapter starts by explaining the basic FPGA enabled PC organisation for CA computations and the related performance model. It discusses a scalable FPGA enabled PC organisation for CA computations with identified parameters that are defined both by CA algorithm and the FPGA hardware. Finally, a

*This chapter is based on the following publications:

- S. Murtaza and A.G. Hoekstra and P.M.A. Sloot, 'Compute Bound and I/O Bound Cellular Automata Simulations on FPGA logic', *ACM Transactions on Reconfigurable Technology and Systems*, **1**, 1-21 (2009)

simple metric that categorises a CA algorithm in terms of its execution times for the said FPGA-PC system organisation is also presented.

3.1 Basic Organisation

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 generation is uploaded back to the host system where all the required pre- and post-processing is performed. However, this setup is feasible only when the whole CA lattice fits the given FPGA space.

3.1.1 Generic Performance Model

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

$$T_{ft} = T_{fpga} + T_{fo} \tag{3.1}$$

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 Equation (3.1) we assume a serial implementation, that is, the useful computation (T_{fpga}) does not execute at the same time as the overhead work (T_{fo}). However, in many cases an overlap between computation and the overhead work is possible, and then the model needs to be adapted slightly. We can now define the obtained speedup by running on the FPGA as

$$S = \frac{T_{pc}}{T_{ft}} = \frac{T_{pc}/T_{fpga}}{1 + T_{fo}/T_{fpga}} = \frac{S_{max}}{1 + f_o}. \tag{3.2}$$

The maximum speedup S_{max} that can ever be obtained on the FPGA enabled PC 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, Equation (3.2) can be written as

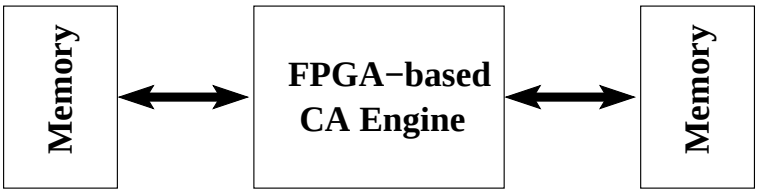

Figure 3.1: *Basic computation model: A scalable FPGA based CA implementation using the two attached on-board memory banks to store the two consecutive CA lattice computations.*

$$S \approx S_{max}(1 - f_o). \tag{3.3}$$

We can also define an efficiency of the FPGA implementation as

$$\varepsilon = \frac{S}{S_{max}}. \tag{3.4}$$

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

$$T_{fo} = T_{pre} + T_s + T_r + T_{pos} \tag{3.5}$$

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

$$f_o = f_{pre} + f_s + f_r + f_{pos}. \tag{3.6}$$

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. However, note that such model is only feasible when the whole CA lattice fits the given FPGA space.

3.2 FPGA with Multiple On-board Memory Banks

For real CA applications the lattice is large (say 128^3 cells). Consequently we can assume that the lattice is larger than the FPGA capacity but still fits the memory banks available on the FPGA board. The resulting computing system is composed of a host machine for pre- and post-processing and a FPGA board with on-board memory banks connected as a co-processor for CA computations. With this FPGA based CA accelerator system, the

host machine initially describes the problem to be solved and downloads all the relevant information (CA lattice data) to one of the on-board memory banks of the FPGA board and then starts the FPGA for computations. The FPGA runs the compute engine (CE) implementing the CA transition function and uses the two attached on-board memory banks to store the two consecutive CA lattice computations as shown in Figure 3.1. At each step t, the CE reads the current state S(t) of k cells (where k is the number of CA cells that the FPGA is able to read from the source memory in parallel) from the source memory bank, computes the new state S(t+1) of k cells using p PEs, and writes them out to the destination memory bank. Figure 3.2 shows the system diagram along with the main parameters like k and p. Once the entire CA lattice data is read from the source memory, updated using CE and stored to the destination memory, the roles of the two memory banks are switched and a new iteration similar to the CAREM processor [25] starts. This system organisation enables a totally parallel computation, that is, reading, computing, and writing all are done in parallel and a scalable organisation where the CA lattice size is not confined to the available FPGA capacity but the on-board memory banks which are often quite huge. With the completion of the computations, the host machine uploads the results from the FPGA's on-board destination memory bank for postprocessing.

3.3 Compute and I/O Bound CA Computations

With the general description of the overall system organisation as presented in the previous section, there are many trade-offs to consider when designing and implementing the FPGA based CA accelerator. For the best possible mapping of CA algorithm to the FPGA logic space, the most important aspect is to fully understand the behaviour of the specified CA algorithm in terms of its execution times, which are either compute bound or I/O bound.

We will now restrict ourselves to two-dimensional CA and apply an algorithm based on the alternate use of memory banks as source and destination memory as proposed by [25], and assume that the FPGA is able to read from the source memory, write to the destination memory, and compute cell states all in parallel.

To categorise a specified CA algorithm as a compute bound or an I/O bound, consider a CA lattice with N cells. Assume, the FPGA running p PEs (each PE updates a single cell at a time) in parallel and reads k cells (where k is the number of CA cells that the FPGA's CE is able to read from the source memory in parallel) in time τ_{r}, and the time

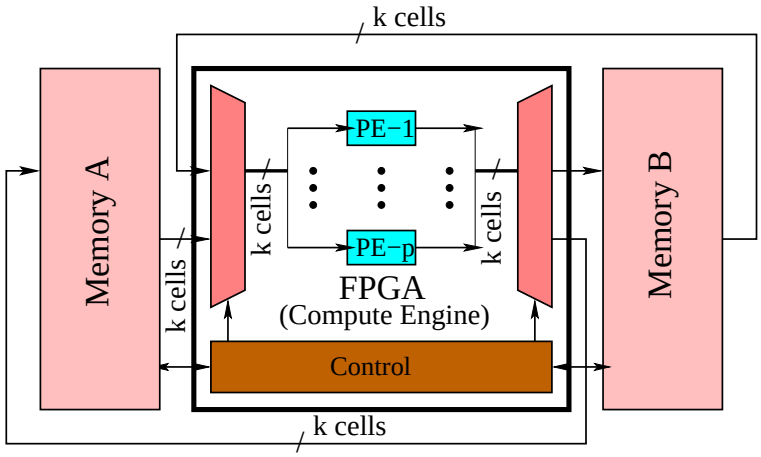

Figure 3.2: *Compute engine implementing the CA transition function with two attached on-board memory banks. Compute engine has p processing elements (PE), where each PE updates a single CA cell at a time. Compute engine reads k CA cells from the source bank and also writes out same to the destination bank.*

to write out k cells from the FPGA into the destination memory is τ_{w} and the time to update a cell is τ_{c}. The total time to compute N cells is $\tau_{\mathrm{c}} \left\lceil \frac{N}{p} \right\rceil$ ($\lceil x \rceil$ is the ceiling of x, i.e., rounding upwards), time to read N cells is $\tau_{\mathrm{r}} \left\lceil \frac{N}{k} \right\rceil$ and time to write N cells is $\tau_{\mathrm{w}} \left\lceil \frac{N}{k} \right\rceil$. Now, we can classify a CA algorithm as *compute bound* computations as long as the following is true:

$$\tau_{\mathrm{c}} \left\lceil \frac{N}{p} \right\rceil \quad > \quad \max(\tau_{\mathrm{r}}, \tau_{\mathrm{w}}) \left\lceil \frac{N}{k} \right\rceil . \tag{3.7}$$

Assuming that $\tau_{\mathrm{r}} \geq \tau_{\mathrm{w}}$ (which for our CA computations is usually true, as the amount of input data per cell is larger then the amount of output data per cell), Equation (3.7) becomes

$$\tau_{\mathrm{r}} \quad \leq \quad \tau_{\mathrm{c}} \left\lceil \frac{N}{p} \right\rceil \left\lceil \frac{N}{k} \right\rceil^{-1} . \tag{3.8}$$

Usually $N \gg p$ and $N \gg k$, and therefore,

$$\tau_{\mathrm{r}} \quad \leq \quad \tau_{\mathrm{c}} \frac{k}{p}. \tag{3.9}$$

Similarly, for I/O bound computations the reverse holds.

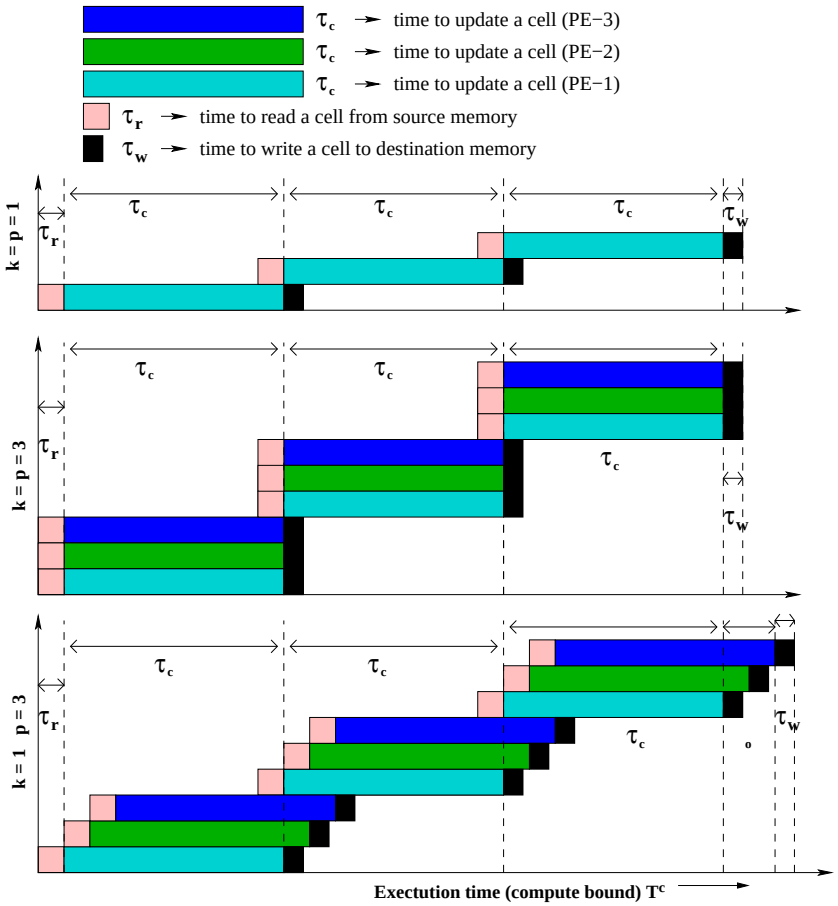

Figure 3.3: *Compute bound scenarios: in upper and the middle panel k (number of CA cells that the FPGA's CE is able to read from the source memory in parallel) is equal to p (number of PEs, each PE updates a single CA cell at a time) and in lower panel $k > p$.*

3.3.1 Execution Time for Compute Bound Computation

For compute bound operations we assume that on start-up the FPGA first reads k cells, after which PEs start to compute. Several scenarios are possible as shown in Figure 3.3. If $p \leq k$ (upper and the middle panel of Figure 3.3) all PEs start to compute, whereas if $p > k$ (lower panel of Figure 3.3) only k PEs will start computing. While computing, a next batch of k cells are read. If PEs still sit idle, they can now start, if not, PEs will immediately start computing on a next cell once they have completed computing on a previous cell. In parallel, the result is written to a destination memory location. Finally, after completion of all computations, remaining data is written to memory. The detailed execution profile depends on k and p, and in Figure 3.3 a number of examples are shown. So, except for start up and final writing of data to memory, all execution time is determined by the computation time.

Define T_{c} as the execution time for compute bound computations. An initial start-up time τ_{r} is required to start the PEs to update cells. The PEs are now busy updating lattice cells for a duration $\tau_{\mathrm{c}} \left\lceil \frac{N}{p} \right\rceil$. At the end of the computations, CE waits for a time $\tau_{\mathrm{r}} \left\{ \left\lceil \frac{p}{k} \right\rceil - 1 \right\}$ for the PEs to complete their remaining computations (this happens only when $p > k$, otherwise this term is 0) followed by a time τ_{w} to write out remaining data to the destination memory bank. The sum of the above mentioned time durations results in the overall execution time for the compute bound scenario as specified below.

$$T_{\mathrm{c}} \;=\; \tau_{\mathrm{r}} + \tau_{\mathrm{c}} \left\lceil \frac{N}{p} \right\rceil + \tau_{\mathrm{r}} \left\{ \left\lceil \frac{p}{k} \right\rceil - 1 \right\} + \tau_{\mathrm{w}}. \tag{3.10}$$

Validating Equation (3.10) for $(k = p = 1)$, we get

$$T_{\mathrm{c}} \;=\; \tau_{\mathrm{r}} + N\tau_{\mathrm{c}} + \tau_{\mathrm{w}}.$$

This is exactly as shown in Figure 3.3 case(a). Similarly, when $(k = p)$ in Equation (3.10) (middle panel in Figure 3.3), we find

$$T_{\mathrm{c}} \;=\; \tau_{\mathrm{r}} + \tau_{\mathrm{c}} \left\lceil \frac{N}{p} \right\rceil + \tau_{\mathrm{w}}.$$

3.3.2 Execution Time for I/O Bound Computation

For I/O bound computations, the CE just starts reading k cells, after which computations start. The reading of cells is repeated $\left\lceil \frac{N}{k} \right\rceil$ times. All computations are done in parallel to this reading. Having read all cells, a final compute and write cycle is required to update the last batch of cells that were read into the FPGA. Figure 3.4 shows a number of possible

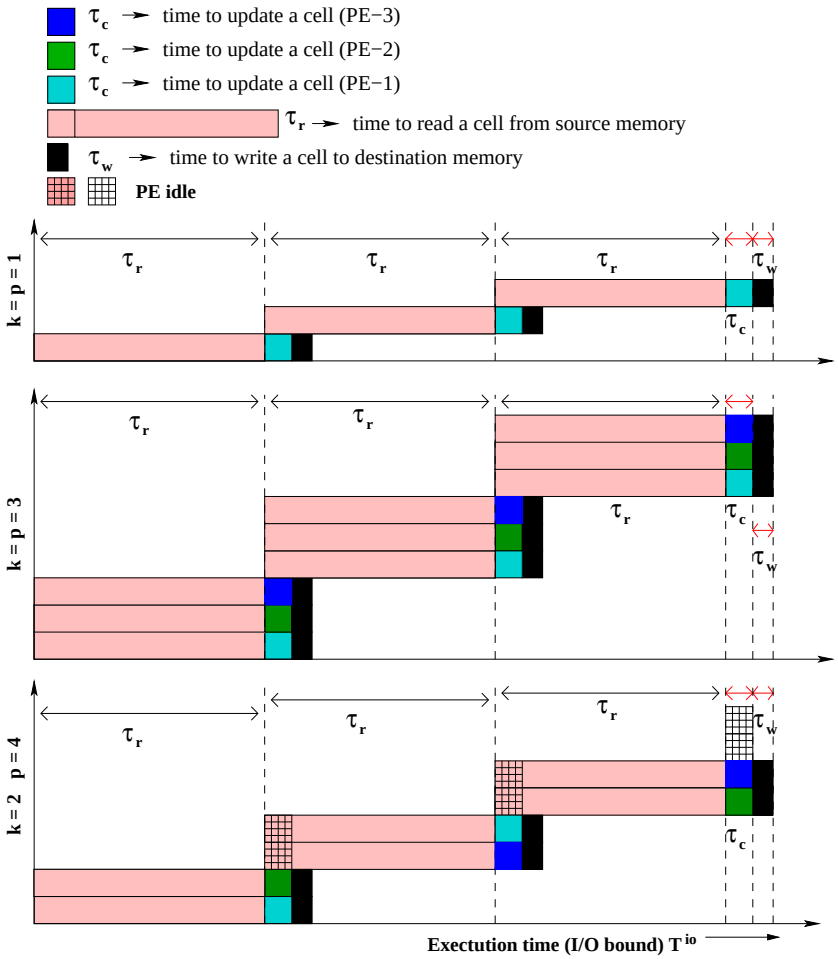

Figure 3.4: *I/O bound scenarios: in upper and the middle panel k (number of CA cells that the FPGA's CE is able to read from the source memory in parallel) is equal to p (number of PEs, each PE updates a single CA cell at a time) and in lower panel, $k > p$ where the mesh of small squares represent idle PEs.*

scenarios, for different values of k and p. Note that when $p > k$ we have the situation that PEs remain idle for the full computation as shown in Figure 3.4 lower panel. In that case one can implement for instance a more advanced CA algorithm, realising a pipeline over CA generations. This is further explained in Section 4.1.2.

The execution time (T_{i}) for I/O bound computations is

$$T_{\mathrm{i}} \;=\; \tau_{\mathrm{r}} \left\lceil \frac{N}{k} \right\rceil + \tau_{\mathrm{c}} + \tau_{\mathrm{w}}. \tag{3.11}$$

Equation (3.10) and Equation (3.11) define T_{c} and T_{i} respectively. They should result in the same execution time when we cross from the Compute bound to the I/O bound case, that is, when (from Equation (3.8))

$$\tau_{\mathrm{r}} \;=\; \tau_{\mathrm{c}} \left\lceil \frac{N}{p} \right\rceil \left\lceil \frac{N}{k} \right\rceil^{-1} \tag{3.12}$$

In substituting Equation (3.12) back in Equation (3.10) and Equation (3.11) respectively, we find a small difference. This has to do with the fact that the expressions for the execution time, take correctly, the start up and end effects into account, whereas these where discarded when deriving Equation (3.8). However, in the limit $N \to \infty$, the small differences disappear, because then the start up and end effects can be neglected.

3.4 Summary

This chapter presents a scalable FPGA based PC organisation where FPGA uses stream processing model for CA computations. Based on FPGA stream processing model, a simple methodology is proposed to categorise a CA algorithm either as I/O bound or as a compute bound algorithm. As presented in the following chapters, this methodology not only hides the details concerning the bit manipulations performed within the FPGA fabric, but also allows accurate prediction of the entire computation performance. Further, I/O and the compute bound categorisation is also used as the foundation for the formulation of the hardware algorithms and their performance model for each of the CA implementations presented in the following chapters.