High performance reconfigurable computing with cellular automata

Murtaza, S.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
CA on FPGA Enabled PC Cluster

A multiple FPGA enabled PC setup, as introduced in the previous chapter, demonstrates a master-slave system configuration where the host PC controls the allocation of job to each of the attached FPGA via a PCI interface. The proposed performance model was implemented and validated for a dual FPGA enabled PC setup. Other than performing the pre- and postprocessing of the CA lattice (which happens to be the only operation performed by the host-PC for a single FPGA based implementation as presented in the Chapter 3), the host-PC also updates the boundary data across the multiple FPGAs over the PCI bus for every single CA iteration. With this master-slave setup, boundary updates across multiple FPGAs by the host-PC is purely a sequential process and with the increase in the number of attached FPGAs, the inter FPGA communication via host-PC ends up as the bottleneck for the overall performance of the CA implementation. As a result, the host-PC time to update the boundary data increases with the increase in the number of FPGAs connected. The host-PC not only takes longer for boundary updates but the FPGAs might also end up waiting for the boundary update completion.

To get around this master-slave setup situation, a fully parallel FPGA based platform is desirable. In this chapter we look into using Maxwell - a 64 FPGA supercomputer [8] for our CA implementations. In addition we discuss in detail a performance model and the results from the test case runs.

7.1 FPGA Enabled PC Cluster

Maxwell - a 64 FPGA supercomputer (see Appendix 10 for details) is a PC cluster where each PC node is a dual-core machine and includes two separate FPGA cards connected via a PCI interface. For the fully parallelised multiple FPGA based CA implementation, the rectangular problem domain is decomposed along the longest dimension into $F$ equal sized chunks (1-D domain decomposition) along the $y$-axis and distributed to $F$ FPGA LBM engines for CA computations. PTK [9]- a thin software layer along with the standard
MPI library allows Maxwell’s FPGAs setup in a ring topology as shown in Figure 7.1 and therefore a one-to-one mapping of \( F \)-chunk CA lattice onto a \( F \) FPGA ring. Using PTK, one can configure the required number of Maxwell’s nodes, where each node is defined as a software process running on the host PC together with a FPGA hardware. And the inter-process (that abstracts inter-FPGA communication) communication is performed over the cluster’s Ethernet network using the MPI library.

### 7.1.1 Execution Time

For the FPGA enabled PC cluster based implementation, consider a rectangular LBM lattice of size \( N \) cells (\( x \)-columns and \( y \)-rows) as shown in Figure 6.2. The master-node (one of the nodes in a cluster) initially performs the 1-D decomposition of the the LBM lattice along the \( y \)-axis into \( F \) chunks (where \( F \) is the number of available FPGA nodes) and downloads all the relevant information (\( \frac{N}{F} \) cells) to each of the FPGA module’s source memory bank. Once the source memory banks of the FPGA modules are loaded, the FPGAs start computing.

Computation within the hardware part, that is, processing of \( \frac{N}{F} \) cells by each of the FPGA can be further categorised into the following three distinct phases which are largely similar to those described for multiple FPGA enabled PC setup in the previous chapter.

#### 7.1.1.1 Boundary Processing

The boundary processing phase is the same as with the multiple FPGA enabled PC setup. Each FPGA starts with processing boundary data and writes out the results to an additional third memory bank available on their respective boards. Once done with the boundary data processing, the third memory bank is released immediately by the FPGA for host DMA. Assuming all the FPGA boards start computing simultaneously, with the
7.1. FPGA Enabled PC Cluster

initial startup time \((\tau_i)\) each board starts with processing boundary cells. This combined initial startup time and processing boundary cells denoted by \(T_b\), can be expressed as

\[
T_b = \tau_i + \left(\frac{2x}{p}\right)\tau_c
\] (7.1)

7.1.1.2 Bulk Processing

Upon completion of the boundary data processing, the compute engines continue processing the bulk lattice cells. Inclusion of the third memory bank for mirroring boundary data on each of the FPGA module is assumed to allow latency hiding, that is, the host machine updates the boundary data across the three FPGA modules (one attached to itself, to the right and to the left neighbour) and the bulk data processing by the compute engines is done in parallel. To update the boundary data across the multiple FPGA boards (processes own FPGA, left and the right neighbour processes FPGA respectively) each host process performs three steps; a) sends out the right boundary column to its right neighbouring process (and as a result it also receives a boundary column from its left neighbouring process), b) updates the two boundary columns and c) sends out the left column (now updated) to its left neighbouring process (again as a result receives one from its right neighbour). As long as the time to process the bulk data \((T_k)\) is larger than the host processes time to update the boundary data \((T_u)\), latency hiding is effective and the overall execution time remains compute bound. Otherwise the host node processing the boundary data across its neighbouring nodes over the PCI bus and the Ethernet dominates the overall execution time. If \(\tau_m\) is the time to send (or receive) a cell from one host process to the other over Ethernet network using MPI library, execution time for bulk cells computation can be expressed as \(\max\{T_u, T_k\}\), where \((T_k)\) and \((T_u)\) are expressed as:

\[
T_k = \frac{1}{p}\left\{\frac{N}{F} - 2x\right\}\tau_c + \left\{\frac{p}{k} - 1\right\}\tau_i + \tau_w
\] (7.2)

\[
T_u = 2x\{\tau_d + \tau_b + \tau_m\}
\] (7.3)

7.1.1.3 Boundary Download

With the completion of a next generation computation, each of the FPGA interrupts the host process. Next the host process downloads the updated boundary data to its connected FPGA modules source bank. The time to download boundary cells \((T_d)\) is \(2x\tau_d\).
The sum of the above mentioned time durations results in an overall execution time for the FPGA enabled PC cluster implementation.

\[ T_F = T_b + \max \{ T_u, T_k \} + T_d. \] (7.4)

The above three steps are repeated in computation of every single generation and repeated until the required number of generations are computed. Once done, each of the host process uploads the data from its attached FPGA modules destination memory banks and finally, the master-node collects and combines all the data chunks together for further postprocessing.

### 7.1.2 Speedup

For \( T_k \gg T_u, T_F \) is expressed as

\[ T_F = \frac{T_1}{F} + \frac{p}{k} \left\{ \frac{F-1}{F} \right\} \tau_r + \left\{ \frac{F-1}{F} \right\} \tau_w + 2x\tau_d \] (7.5)

and the obtained speedup using FPGA enabled PC cluster implementation is

\[ S = \frac{F}{1 + f_{\text{comm}}}, \] (7.6)

where total fractional communication overhead \( f_{\text{comm}} \) is the sum of: fractional boundary data downloading overhead \( (f_b) \), fractional PE completion overhead \( (f_{\text{pe}}) \), and fractional writing memory overhead \( (f_w) \). Each of the fractional overhead is defined as:

\[ f_b = 2x \left\{ \frac{F}{T_1} \right\} \tau_d \] (7.7)

\[ f_{\text{pe}} = \frac{p}{k} \left\{ \frac{F-1}{T_1} \right\} \tau_r \] (7.8)

\[ f_w = \left\{ \frac{F-1}{T_1} \right\} \tau_w \] (7.9)

As long as \( T_1 \) is big enough, the fractional overheads will be very small and a speedup very close to \( F \) may be expected.

For \( T_u \gg T_k, T_F \) is expressed as

\[ T_F = \tau_r + \frac{2x}{p} \tau_c + 2x \left\{ \tau_b + \tau_d + \tau_m \right\} \] (7.10)

and the obtained speedup is

\[ S = \frac{1}{\frac{T_1}{\tau_r} + \frac{2x}{p} \tau_c + 2x \tau_b + 2x \tau_d + 2x \tau_m}. \] (7.11)
7.2 Details of the Test Case

In order to validate our performance model for FPGA enabled PC cluster based CA implementations, rectangular domains with periodic boundary conditions of varying sizes were used as test cases. For each of the test case, the number of PEs per FPGA was fixed to $p = 8$ as these were the maximum number of PEs we could fit into an FPGA chip available on Maxwell (see Appendix 10 for details). The master node determines the problem domain and performs the 1-D domain decomposition of the CA lattice along the $y$-axis into $F$ equal sized chunks, where $F$ is the number of host processes (or FPGAs, as each host process includes one FPGA for acceleration) used for the test runs. The CA lattice once prepared, the master node setups up the required number of processes (that is FPGA nodes) in a ring topology and uses $MPI\_\text{Scatter}$ to distribute the lattice chunks to each of the FPGA nodes. Using the ring topology and the 1-D domain decomposition along the $x$-axis ensures, that the each computing node has the required boundary data along the $y$-axis. However, boundary data along the $x$-axis needs to be exchanged among the neighbouring computing nodes. Once the lattice is distributed each of the host process loads the available source memory bank on its attached FPGA board with the given chunk of lattice. Table 7.1 lists all the system sizes used as the test cases. Each system size was computed for 10-iterations using a varying number of FPGA nodes. During each single iteration, host processes use $MPI\_\text{SendRecv}$ to exchange and update boundary across their respective left and the right neighbour nodes. With the number of specified iterations computed, master node uses $MPI\_\text{Gather}$ to collect the updated lattice chunks from each of the host process for further postprocessing. Note, the execution times presented in the following sections relate to the part between the first $MPI\_\text{Scatter}$ and last $MPI\_\text{Gather}$ and therefore, do not include the system setup time, that is, loading FPGA boards and lattice scatter/gather related operations.

<table>
<thead>
<tr>
<th>$N$</th>
<th>2 FPGAs</th>
<th>4 FPGAs</th>
<th>8 FPGAs</th>
<th>16 FPGAs</th>
<th>20 FPGAs</th>
<th>25 FPGAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000*1000</td>
<td>500*1000</td>
<td>250*1000</td>
<td>125*1000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1200*1000</td>
<td>600*1000</td>
<td>300*1000</td>
<td>150*1000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1600*1000</td>
<td>800*1000</td>
<td>400*1000</td>
<td>200*1000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1800*1000</td>
<td>900*1000</td>
<td>450*1000</td>
<td>225*1000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2000*1000</td>
<td>1000*1000</td>
<td>500*1000</td>
<td>250*1000</td>
<td>125*1000</td>
<td>100*1000</td>
<td>80*1000</td>
</tr>
</tbody>
</table>

Table 7.1: LBM system sizes $N$ used as test cases for the F FPGA enabled PC cluster implementation. Right column represents the total system sizes used as a test case and rest of the columns show the lattice chunk size for the specified number of FPGA enabled setup.
Figure 7.2: System parameter ($\tau_m$): Stars represent the round trip times for the data packets (ranging from 64KB to 8MB) over the gigabit Ethernet from one PC to another using MPI-based ping-pong test runs. The average of the calculated bandwidth from the test runs, as shown by triangles, was used to define $\tau_m$, that is, the time to send an LBM cell from one PC to another over Ethernet using MPI routines.

7.2.1 System Parameters

Several test case runs were performed using MPI Send-Receive routines to determine system parameter $\tau_m$, that is, the time to send (or receive) a LBM cell (each cell includes ten 64-bit words, where nine represent the nine velocities of a cell and an additional one for future system implementations to include complex boundary computations) from host process to the another host process over gigabit Ethernet. Data packets of varying sizes, ranging from 64KB to 8MB, were used. The round trip times were measured as shown in Figure 7.2. The average bandwidth calculated from the experiment was 91MB/sec and accordingly used to define $\tau_m = 835\,\text{ns}$.

7.3 Performance Results

The subsequent sections discuss the test case results performed for the varying number of system sizes (as specified in the Table 7.1) and number of FPGA nodes.

7.3.1 Two Million Cells Lattice

Figure 7.4 shows the execution times for a two million cells 2D LBM lattice with periodic boundary conditions computed for 10 iterations. This system size was also the largest lattice used as a test case (cluster configuration on the Maxwell machine imposed this
7.3. Performance Results

The test case was run for \( F = \{1, 2, 4, 8, 16, 20, 25\} \) number of FPGAs. Dividing the given system size into equal sized chunks limited the maximum number of FPGAs to 25 though the total number of available FPGAs on Maxwell is 32 (see Appendix 10). With the specified set of parameters, latency hiding work as per the model, that is, Equation (7.5) and as represented by a broken line in Figure 7.4. The results from our implementation, as represented by stars, closely follow the given performance model as shown in the Figure 7.4. And as expected with the increase in the number of FPGAs the execution time also goes down. Figure 7.5 shows the speedup for the same system configuration. However, with the increase in FPGAs especially after \( F > 8 \), the measured speedups deviate away from the model predictions. We performed system profiling to further investigate our model and measure the relative errors. During each system iteration computation, each host process along with its attached FPGA accelerator computes a lattice chunk in three distinct phases, as explained in previous sections, where each stage progress is updated across process and its accelerator via interrupts. Using software timers we explicitly monitored these three phases of the computation as shown in Figure 7.3. Time to compute and download the boundary data is around 10\( \text{ms} \) which is negligible compared to the bulk cells computation time, that is, in the order of hundreds of milliseconds as shown in Figure 7.3. Therefore, relative error(s) with the boundary cells compute plus download times are hardly noticeable in the overall execution time. Please note, our performance model is based on the FPGA execution times and the profiling measurements use the software timers within the host processes. It is not surprising to see the errors with boundary compute time which are in the order of a millisecond from FPGA perspective but, registered by host processes operating system within few milliseconds. However, the main contribution to the overall execution time is the bulk computation. As shown in Figure 7.3 (top left), implementation measurements (shown as stars) closely follows the model represented by the broken line. The relative error with the bulk compute time is around seven percent which is consistent as reported in our single FPGA based implementation in Chapter 5. We are aware of this error, and it is work-in-progress. In future we plan to accommodate these relative errors in our improved performance model where system profiling would include: time to switch the roles of the attached source and destination memory banks performed after every iteration, and the time to write out the next state results to the destination memory banks, that is, \( \tau_w \) (the current model assumes it to be equal to \( \tau_r \)).

7.3.2 Other System Size

The other system sizes as specified in the Table 7.1 were also used as test cases and all were computed for 10 iterations using \( F = \{1, 2, 4, 8\} \) number of FPGAs. Figure 7.6
Figure 7.3: Profiling three stages of a single iteration computation: 2 million cells system size computed for a single iteration using specified number of FPGAs where broken line represents the model predictions and stars stand for the measurements. Each iteration starts with FPGA computing boundary cells (shown in top left figure), followed by computing bulk cells (shown in top right figure) and in parallel host process update boundary cells (successful latency hiding assumes time to compute bulk cells by a FPGA to be greater than the boundary update time by the host process). And once the FPGA is done with computing the lattice chunk, the host process downloads the update boundary cells to the respective FPGA’s on-board memory bank (shown in lower left figure). The overall execution time is shown in lower right figure. Bulk cells computation as shown in top right figure also shows the relative error represented by a solid line.
7.3. Performance Results

Figure 7.4: Execution time as a function of FPGAs. Measurements are from the LBM lattice of size 1000x2000 cells computed for 10 iterations where the broken line represents the model and stars represent the measurements.

Figure 7.5: Speedups as a function of FPGAs for a 1000x2000 cells system size. Broken line represents the performance model as specified in Equation (7.6) and stars show the implementation measurements.
Figure 7.6: Execution time as a function of FPGA. Each curve represents the execution times for computing a specified system size (triangle for 2-million, square for 1.8-million, star for 1.6-million, cross for 1.2-million and plus for 1-million LBM cells respectively) for 10 iterations for varying number of FPGAs.

and Figure 7.7 show the execution times as a function of FPGAs and the system sizes respectively. As expected with the increase in the FPGAs, the execution time goes down and with increase in the system size the execution time goes up. Speedup and the efficiency graphs as shown in Figure 7.8 clearly show how the overall system performance improves and converges to the model with an increase in the lattice system size.

7.4 Conclusion and Future Work

As presented in Chapter 5, we started with a single FPGA based implementation, where the design and implementation is more focused towards manipulating the FPGA fabric to perform bit-level operations. On the other side, this chapter demonstrates, how a single FPGA based LBM compute core with additional features and modification, is used in the parallel multiple FPGA system, where the focus of the overall system design is on the transfer and manipulation of chunks of data in the order of mega bytes. The chapter also presented a performance model for a FPGA enabled PC cluster based 2D CA accelerator, with periodic boundary conditions. The system implementation was validated for a number of system sizes, with 2-million cells 2D LBM grid being the largest. Performance model shows how the FPGA enabled PC cluster is the preferred multiple FPGA organisation over the multiple FPGA based PC setup. Latency hiding as suggested by [56] has been the premise throughout our multiple FPGA based systems, and was fully exploited for PC cluster based system as well. To validate the successful implementation of
7.4. Conclusion and Future Work

Figure 7.7: Execution time as a function of system size. Each line represents the execution times for computing the specified system sizes for 10 iterations. Stars represent one, pluses represent two, triangles represent four, and circles represent eight FPGA implementation setups respectively.

Figure 7.8: Speedups and the efficiencies as a function of system size. Stars represent two, triangles represent four, and squares represent eight FPGA based implementations respectively. In the speedup figure (towards left), broken line represents the model.
using latency hiding technique, and to track the interrupt driven interactions between the host processes and the attached FPGAs, a system profiling was performed for every single CA iteration computation using software timers. Firstly, latency hiding being in sync with the performance model and secondly, the time to update and download boundary data by the host processes (over the PCI interface and the Ethernet/MPI network) being negligible compared to the time for computing bulk cells, indicates a two dimensional decomposition system investigation.