High performance reconfigurable computing with cellular automata

Murtaza, S.

Publication date
2010

Chapter 6

CA on Multiple FPGA Enabled PC*

If the logic resources available within the chip limit the number of possible PEs within our implementation, then could we attempt to push this limit outside the chip, that is, have another chip in parallel? FPGAs operate at a clock speed of more than one order of magnitude slower than microprocessors [50]. Will this provide enough room for the exchange of data between the multiple FPGAs running parallel over the host interface? These questions motivated us to port our implementation to a multiple FPGA setups for further performance investigations. In this chapter we look into multiple FPGA enabled PC based CA implementations where the concept of latency hiding [56] is exploited. It also focuses on a dual FPGA enabled PC setup for performance measurements. A dual FPGA enabled PC should provide a realistic view on the feasibility of implementing multiple FPGA based CA accelerators.

*This chapter is based on: S. Murtaza and A.G. Hoekstra and P.M.A. Sloot, 'Performance of floating-point based Cellular Automata Simulations using a dual FPGA system', *submitted.* Initial results based on this chapter's dual FPGA-based implementation were presented at the Second International Workshop on High-Performance Reconfigurable Computing Technology and Applications (Austin, Texas, 2008) and published in [77].

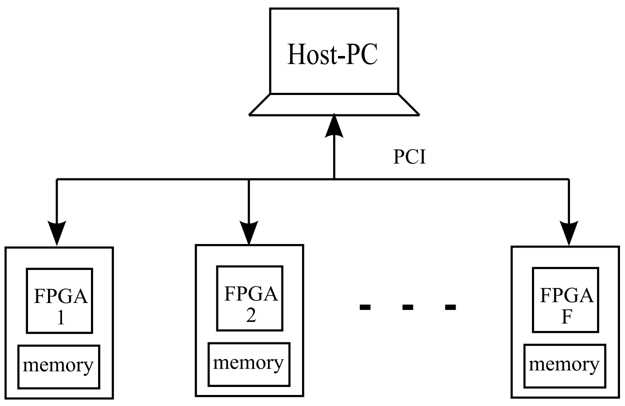

Figure 6.1: *Multiple FPGA enabled PC setup.*

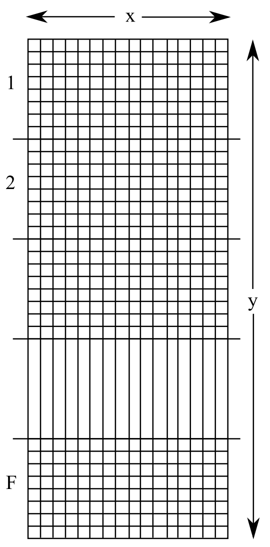

Figure 6.2: *Rectangular lattice (with x columns and y rows) divided along y-axis into F-chunks. Each chunk is processed by a dedicated FPGA based LBM engine.*

6.1 Multiple FPGA Enabled PC

Applying Equation (3.9) to our single FPGA enabled D2Q9 LBM implementation (with system parameters $k = 1/10$, $\tau_\mathrm{c} = 675$, and $\tau_\mathrm{r} \approx 1.1$, see Section 5.3 for details) guarantees our system to be compute bound as long as the system implementation has $p \leq 61$. FPGA logic resources available on our single FPGA enabled system (see Appendix 10 for hardware details) limited our implementation to a maximum of sixteen PEs (i.e. $p = 16$ LBM cores). With these limitations imposed by the available logic resources, we were able to achieve barely 27% of the theoretical possible p PEs for our LBM hardware accelerator implementation. To port a single FPGA based LBM implementation to a multiple FPGA enabled PC system, required modifications both within hardware and the software cores of our existing implementation. The multiple FPGA enabled PC system is composed of a host PC with multiple FPGA boards connected via PCI interface as shown in Figure 6.1. Each FPGA board includes multiple on-board memory banks as shown in Figure 3.1. For CA implementation, the resulting computing system's PC does all the pre- and post-processing and FPGA boards are connected as coprocessors for CA computations. With this multiple FPGA enabled PC based CA accelerator system, the host machine initially describes the problem to be solved, and downloads all the relevant information (chunk of CA lattice data) to each of the FPGA's on-board source memory bank. Once initialised, the FPGAs start computing, and the host machine keeps track of the overall lattice computations (that is, completion of each iteration and boundary exchange across the FPGA boards) via interrupts from the host accessible control registers available on each of the FPGA.

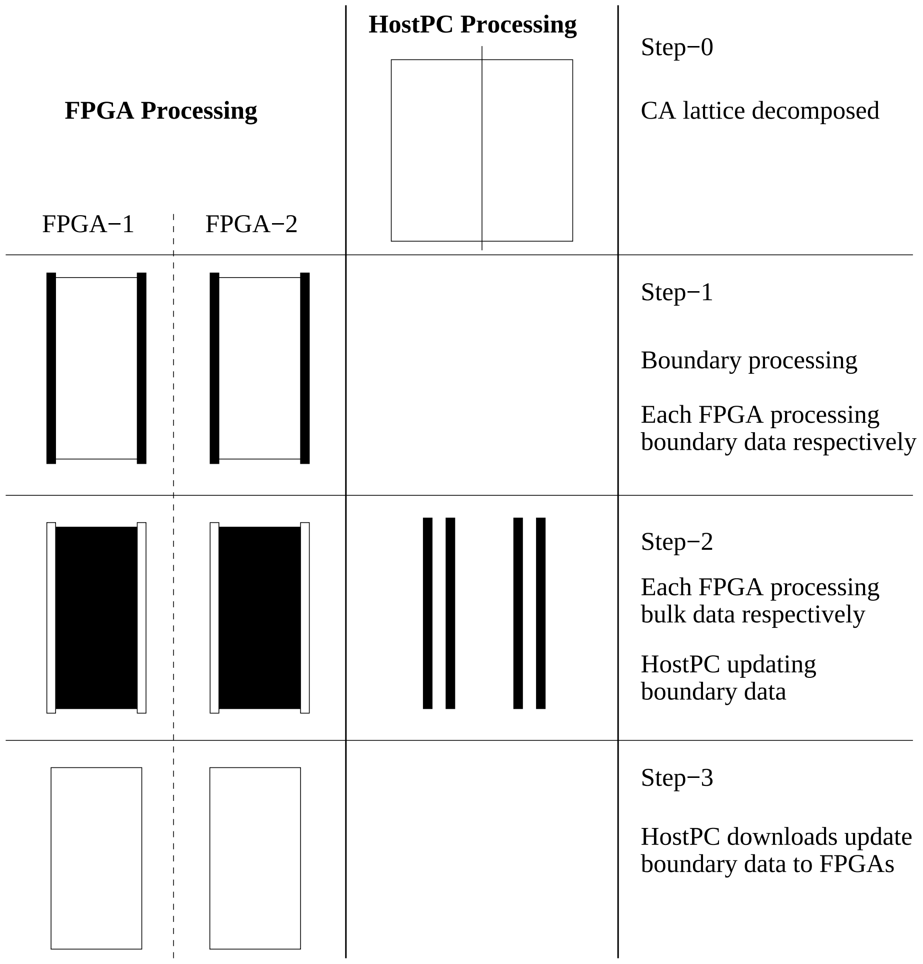
HostPC Processing
FPGA Processing
FPGA−1
FPGA−2
Step−0
CA lattice decomposed
Step−1
Boundary processing
Each FPGA processing boundary data respectively
Step−2
Each FPGA processing bulk data respectively
HostPC updating boundary data
Step−3
HostPC downloads update boundary data to FPGAs

Figure 6.3: *Dual FPGA enabled PC computation model: System initialisation (step-0): host machine decomposes the CA lattice into two halves and downloads each half to the respective FPGA's source memory bank. Boundary processing (step-1): Each FPGA starts with processing boundary data and signals the host via an interrupt. Bulk processing (step-2) FPGA's continue processing bulk data cells and in parallel the host machine updates boundary data across the two FPGA boards. Boundary download (step-3): With the completion of the whole lattice iteration computation, host machine downloads the updated boundary data to each of the FPGA's source memory bank.*

6.1.1 Execution Time

In order to determine the execution time for the compute bound CA computation using multiple FPGA enabled PC, consider the F FPGA enabled PC implementation as shown in the Figure 6.1 and assume the problem domain is a rectangular lattice of size N (with x columns and y rows). The host-PC initially divides the LBM lattice into F-chunks (as shown in Figure 6.2) 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 all 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 further be categorised into the following three distinct phases (see Figure 6.3 for dual FPGA enabled PC system implementation).

6.1.1.1 Boundary Processing

Each FPGA's compute engine starts by processing boundary data. With boundary data being only a small part of the overall lattice (when $N \gg 2xF$), the compute engines write out boundary related results to an additional third memory bank (thus mirroring the boundary data results independently) available on their respective boards. Immediately after the completion of processing boundary cells, the compute engine releases the third memory bank for host DMA. For large system sizes especially when $N \gg p$, we can approximate the ceiling functions behaviour and rewrite Equation (3.10) as:

$$T_1 = \tau_\mathrm{r} + \left\{\frac{N}{p}\right\}\tau_\mathrm{c} + \left\{\frac{p}{k} - 1\right\}\tau_\mathrm{r} + \tau_\mathrm{w}. \tag{6.1}$$

Assume that all FPGAs start computing simultaneously, with the initial startup time (τ_r), each FPGA starts with processing boundary cells. This combined initial startup time and processing boundary cells denoted by T_b, can be expressed as

$$T_\mathrm{b} = \tau_\mathrm{r} + \left\{\frac{2x}{p}\right\}\tau_\mathrm{c} \tag{6.2}$$

6.1.1.2 Bulk Processing

During this phase of computation the system relies on the effective use of latency hiding. Herbordt et al. [56] highlights latency hiding as a basic technique for achieving high performance in parallel applications, and further recommends this as one of the main design techniques to be exploited in HPC/FPGA applications. Upon completion of 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 all of the FPGA modules and the bulk data processing by the compute engines are done in parallel. As long as the time to process the bulk data (T_k) is larger than the host machine's time to update the boundary data (T_u), latency hiding is effective and the overall execution time remains compute bound. Otherwise the host machine processing the boundary data across the FPGA boards over the PCI bus dominates the overall execution time. If τ_b is the time to update a boundary cell by the host machine and τ_d is the time to download a cell from the host machine to a FPGA or vice versa, execution time for bulk cells computation can be expressed as $\max\{T_\mathrm{u}, T_\mathrm{k}\}$, where ($T_\mathrm{k}$) and ($T_\mathrm{u}$) are expressed as:

$$T_\mathrm{k} = \frac{1}{p}\left\{\frac{N}{F} - 2x\right\}\tau_\mathrm{c} + \left\{\frac{p}{k} - 1\right\}\tau_\mathrm{r} + \tau_\mathrm{w} \tag{6.3}$$

$$T_\mathrm{u} = F\left\{2x\tau_\mathrm{d} + 2x\tau_\mathrm{b}\right\} \tag{6.4}$$

6.1.1.3 Boundary Download

With the completion of a next generation computation, each FPGA interrupts the host machine. Following this the host machine downloads the updated boundary data to each of the FPGA modules source bank. The time to download boundary cells is expressed as:

$$T_\mathrm{d} = F\left\{2x\tau_\mathrm{d}\right\} \tag{6.5}$$

The sum of the above mentioned time durations result in an overall execution time for the multiple FPGA enabled PC implementation

$$T_\mathrm{f} = T_\mathrm{b} + \max\left\{T_\mathrm{u}, T_\mathrm{k}\right\} + T_\mathrm{d}. \tag{6.6}$$

The above three steps are repeated in the computation of every single CA iteration and repeated until the required number of iterations are computed. Once done, the host machine uploads the whole data from each of the FPGA module's destination memory bank for further postprocessing.

6.1.2 Speedup

For multiple FPGA enabled PC implementation, when $T_\mathrm{k} \gg T_\mathrm{u}$, T_f is expressed as

$$T_\mathrm{f} = \frac{T_1}{F} + \frac{p}{k}\left\{\frac{F-1}{F}\right\}\tau_\mathrm{r} + \left\{\frac{F-1}{F}\right\}\tau_\mathrm{w} + 2x\left\{F\right\}\tau_\mathrm{d} \tag{6.7}$$

and the obtained speedup is

$$S = \frac{T_1}{T_\mathrm{f}} = \frac{F}{1 + f_{comm}}, \tag{6.8}$$

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

$$f_b = 2x\left\{\frac{F^2}{T_1}\right\}\tau_{\mathrm{d}} \tag{6.9}$$

$$f_{pe} = \frac{p}{k}\left\{\frac{F-1}{T_1}\right\}\tau_{\mathrm{r}} \tag{6.10}$$

$$f_w = \left\{\frac{F-1}{T_1}\right\}\tau_{\mathrm{w}} \tag{6.11}$$

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.

When $T_{\mathrm{u}} \gg T_{\mathrm{k}}$, T_{f} is expressed as

$$T_{\mathrm{f}} = \tau_{\mathrm{r}} + \frac{2x}{p}\tau_{\mathrm{c}} + F\left\{2x\tau_{\mathrm{b}} + 4x\tau_{\mathrm{d}}\right\} \tag{6.12}$$

and the obtained speedup is

$$S = \frac{1}{\frac{1}{T_1}\left\{\tau_{\mathrm{r}} + \frac{2x\tau_{\mathrm{c}}}{p} + 2xF\tau_{\mathrm{b}} + 4xF\tau_{\mathrm{d}}\right\}}. \tag{6.13}$$

6.2 Test Cases and Results

In order to validate multiple FPGA enabled PC based two-dimensional CA accelerator implementation, we implemented and validated our model for the dual FPGA enabled PC setup. For porting our single FPGA to the dual FPGA enabled PC implementation, two of the FPGAs available on Maxwell- a 64 FPGA-based supercomputer (see Appendix 10 for setup details) were used and benchmarked for varying number of square lattice sizes. For square lattice (that is, $x = \sqrt{N}$) CA simulations and dual FPGA enabled PC setup (that is, $F = 2$), the multiple FPGA enabled PC implementation performance model, as specified in the previous section, simplifies as follows

$$T_{\mathrm{b}} = \tau_{\mathrm{r}} + \left\{\frac{2\sqrt{N}}{p}\right\}\tau_{\mathrm{c}} \tag{6.14}$$

$$T_{\mathrm{k}} = \frac{1}{p}\left\{\frac{N}{2} - 2\sqrt{N}\right\}\tau_{\mathrm{c}} + \left\{\frac{p}{k} - 1\right\}\tau_{\mathrm{r}} + \tau_{\mathrm{w}} \tag{6.15}$$

$$T_{\mathrm{u}} = 2\left\{2\sqrt{N}\tau_{\mathrm{d}} + 2\sqrt{N}\tau_{\mathrm{b}}\right\} \tag{6.16}$$

$$T_{\mathrm{d}} = 2\left\{2\sqrt{N}\tau_{\mathrm{d}}\right\} \tag{6.17}$$

The overall execution time for the dual FPGA enabled implementation is the sum of the above mentioned time durations and is expressed as:

$$T_2 = T_{\mathrm{b}} + \max(T_{\mathrm{u}}, T_{\mathrm{k}}) + T_{\mathrm{d}}. \tag{6.18}$$

6.2.1 Minimum Required System Size

For square lattice dual FPGA based implementation, the minimum system size (N^*) required for latency hiding to work successfully is determined when T_{u} and T_{k} are equal.

$$4\sqrt{N}\left\{\tau_{\mathrm{d}} + \tau_{\mathrm{b}}\right\} = \frac{1}{p}\left\{\frac{N}{2} - 4\sqrt{N}\right\}\tau_{\mathrm{c}} + \left\{\frac{p}{k} - 1\right\}\tau_{\mathrm{r}} + \tau_{\mathrm{w}}$$

$$N^* = \left\{4 + \frac{4p\tau_{\mathrm{d}}}{\tau_{\mathrm{c}}} + \frac{4p\tau_{\mathrm{b}}}{\tau_{\mathrm{c}}} + \frac{p}{\tau_{\mathrm{c}}}\sqrt{16\left\{\tau_{\mathrm{d}} + \tau_{\mathrm{b}} + \frac{\tau_{\mathrm{c}}}{p}\right\}^2 - \frac{2\tau_{\mathrm{c}}\tau_{\mathrm{r}}}{k} + \frac{2\tau_{\mathrm{c}}\tau_{\mathrm{r}}}{p} - \frac{2\tau_{\mathrm{c}}\tau_{\mathrm{w}}}{p}}\right\}^2$$

6.2.2 Speedup

For dual FPGA enabled PC implementation, when $T_{\mathrm{k}} \gg T_{\mathrm{u}}$, T_2 is expressed as

$$T_2 = \frac{T_1}{2} + \left\{\frac{p}{2k}\right\}\tau_{\mathrm{r}} + \left\{4\sqrt{N}\right\}\tau_{\mathrm{d}} + \left\{\frac{1}{2}\right\}\tau_{\mathrm{w}} \tag{6.19}$$

and the obtained speedup is

$$S = \frac{2}{1 + f_{comm}}, \tag{6.20}$$

where fractional overheads f_b, f_{pe} and f_w are defined as:

$$f_b = \left\{\frac{8\sqrt{N}}{T_1}\right\}\tau_{\mathrm{d}} \tag{6.21}$$

$$f_{pe} = \left\{\frac{p}{kT_1}\right\}\tau_{\mathrm{r}} \tag{6.22}$$

$$f_w = \left\{\frac{1}{T_1}\right\}\tau_{\mathrm{w}} \tag{6.23}$$

And as long as T_1 is big enough, the fractional overheads will be very small and a speedup very close to *two* may be expected.

When $T_{\mathrm{u}} \gg T_{\mathrm{k}}$, T_2 is expressed as

$$T_2 = \tau_{\mathrm{r}} + \left\{ \frac{2\sqrt{N}}{p} \right\} \tau_{\mathrm{c}} + F \left\{ 2\sqrt{N} \right\} \tau_{\mathrm{b}} + F \left\{ 4\sqrt{N} \right\} \tau_{\mathrm{d}} \tag{6.24}$$

and the obtained speedup is

$$S = \frac{1}{\frac{1}{T_1} \left\{ \tau_{\mathrm{r}} + \frac{2\sqrt{N}\tau_{\mathrm{c}}}{p} + 2\sqrt{N}F\tau_{\mathrm{b}} + 4\sqrt{N}F\tau_{\mathrm{d}} \right\}}. \tag{6.25}$$

From Equation (6.25), in combination with the fact that in this case $N < N^*$, we can conclude that the performance will increasingly deteriorate by the fact that the execution is communication bound, and the speedup will be much smaller than *two*. One can even expect a speed down, that is, and execution time being larger as compared to the single FPGA enabled execution.

6.2.3 System Parameters

Several test case runs were performed to determine system parameter τ_{d}, that is, the time to upload an 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-PC to the DDR2-SDRAM, available on the attached FPGA board or vice versa. Using the software process running on the host-PC, varying number of data packets (ranging from 16KB to 2MB) were first downloaded to the on-board DDR2-SRAM bank followed by uploading them back to the host memory. The round trip timing were measured as shown in Figure 6.6. The average bandwidth calculated from the experiment was 33MB/sec and accordingly used to define $\tau_{\mathrm{d}} = 2.3\mu s$. A number of test case runs for updating 2048 LBM cells were performed to determine τ_{b} time to update an LBM boundary cell, and the average time recorded was $13ns$.

6.2.4 Results

For the dual FPGA enabled PC system, we ported our single FPGA-based implementations with one, two, four, and eight PEs respectively to one of the Maxwell's available nodes (each node includes two FPGAs attached via PCI interface). These implementations were tested for five different square lattice sizes (N equal to 32^2, 64^2, 128^2, 256^2 and 512^2) that were computed for ($g = 256$) number of iterations. The resulting execution times for the corresponding single and the dual FPGA based implementations cases are shown in Figure 6.4. The corresponding speedup is shown in Figure 6.5.

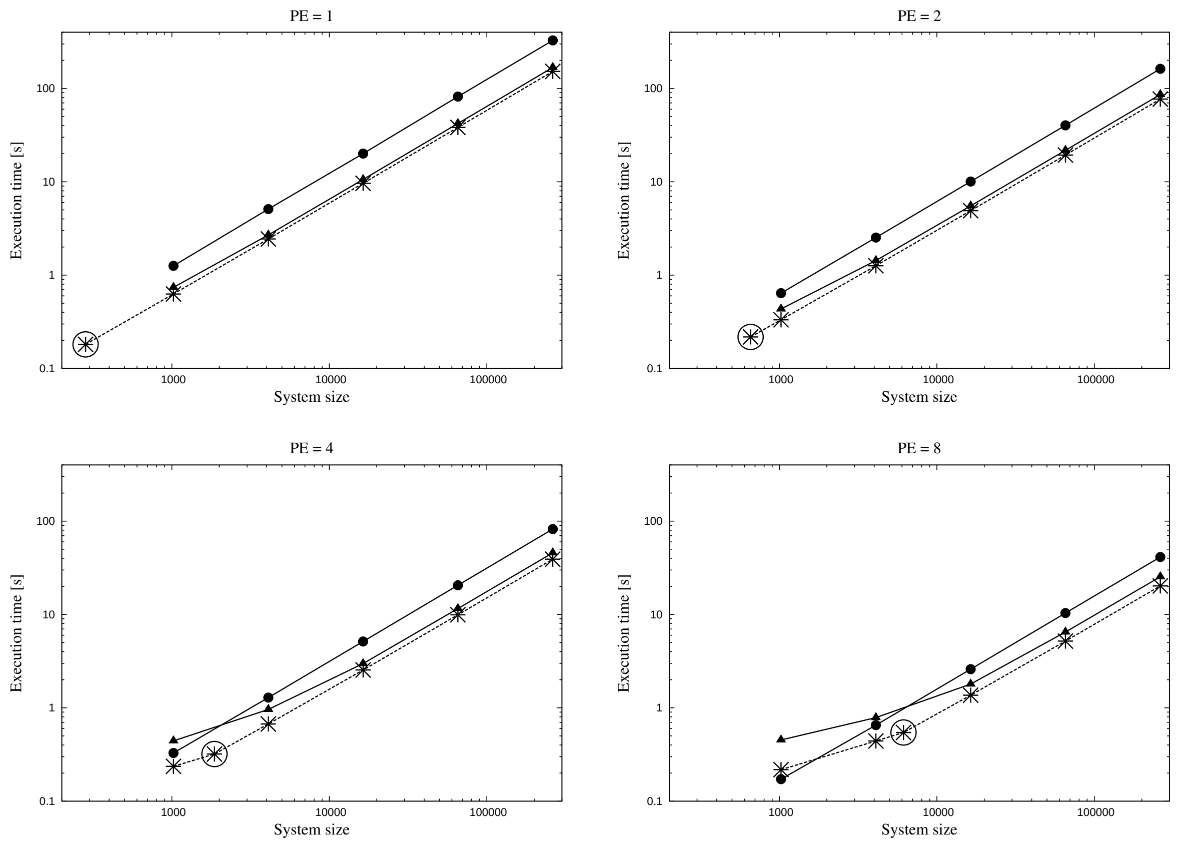

Figure 6.4: *Execution times for computing* 256 *iterations of the square domain LBM D2Q9 grid of size N using a single and dual FPGA enabled system implementation. Broken line represents performance model Equation (6.19) for the dual FPGA enabled execution. Filled circles represent the measured execution times for a single FPGA enabled system implementation, and filled triangles represent a dual FPGA enabled system implementation with (a) 1PE, (b) 2PE, (c) 4PE, and (d) 8PE implementation respectively. Big circles specified on the broken line highlight the minimum system size required for latency hiding to work and for the said implementation to be compute bound.*

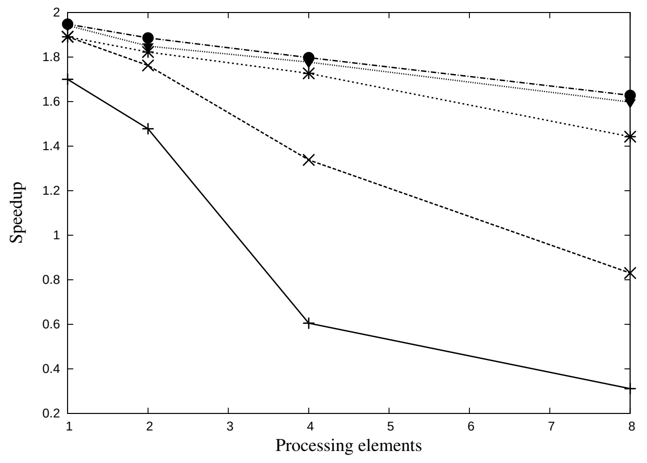

Figure 6.5: *Speedup achieved using dual- over single-FPGA based implementations. Speedup measurements are based on execution times for computing* 256 *iterations of varying square domain LBM D2Q9 system sizes for varying number of PE implementations, using a single and dual FPGA enabled PC implementations respectively. Pluses represent* 32^2, *crosses* 64^2, *stars* 128^2, *triangles* 256^2, *and circles* 512^2 *system sizes respectively.*

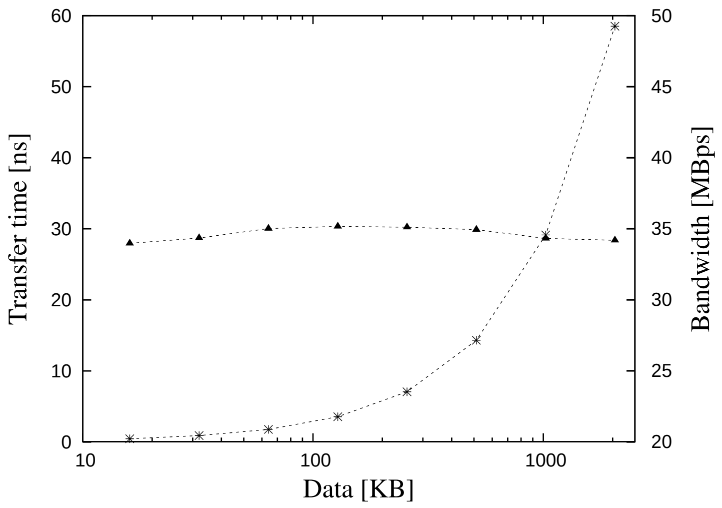

Figure 6.6: *System parameter(τ_d): Stars represent the time to upload the data packets (ranging from 16KB to 2MB) from the DDR2 SDRAM available on the FPGA board to the host-PCs memory over the PCI interface or vice versa. Average of the calculated bandwidth from the test runs, as shown by triangles, was used to define τ_d, that is, the time to upload (or download) an LBM cell from host-PC to the memory banks available on the attached FPGA board.*

With an increasing number of PEs the fractional PE completion overhead (f_{pe}) as shown in Equation (6.10) increases and the overall speedup goes down. And with increase in system size N, the fractional boundary data downloading overhead (f_b) as shown in Equation (6.9) goes down and the overall speedup goes up. All of the three fractional overheads f_b, f_{pe}, and f_w are always larger than zero and overall result in the loss of speedup. However, f_b can be minimised with the inclusion of an extra memory bank for downloading of updated boundary data by the host machine to each of the FPGA board. Moreover, the difference between compute bound speedup (that is, for larger system sizes) and communication bound speedup (or even speed down) is clearly visible due to the fact that latency hiding no longer is effective, for the smaller system sizes.

6.3 Conclusion and Future Work

This chapter presented a detailed discussion on porting a single- to a dual-FPGA based LBM D2Q9 implementation. Based on the single FPGA enabled model as specified in the previous chapters, a model to evaluate the performance of a two dimensional CA executed on multiple FPGA enabled PC system was presented. Further the model was validated for a dual FPGA based setup for the square domain LBM D2Q9 implementations. Results from the dual FPGA based implementations demonstrate how the included latency hiding techniques were a success and the overall execution time was decreased by a factor close to *two*. A working latency hiding also demonstrates what [56] has identified as one of the important HPC/FPGA application design techniques.