J.Natn.Sci.Foundation Sri Lanka 2010 **38** (4): 213-218

RESEARCH ARTICLE

Simulation of diffusion limited aggregation in field programmable gate arrays

W.A.S. Wijesinghe[1], M.K. Jayananda[2] and D.U.J. Sonnadara[2*]

[1] *Department of Electronics, Faculty of Applied Sciences, Wayamba University of Sri Lanka, Makandura.*

[2] *Department of Physics, Faculty of Science, University of Colombo, Colombo 03.*

Revised: 04 December 2009 ; Accepted: 17 July 2010

Abstract: This paper presents design considerations and performance in implementation of Diffusion-limited aggregation (DLA) process on a Xilinx Spartan 3 field programmable gate array (FPGA). The DLA cluster algorithm was implemented as a block RAM and two 32-bit Linear Feedback Shift Register random number generators in hardware. The complete design, written in VHDL and synthesized using Xilinx WebPACK 7.2 was downloaded to the Spartan 3 device for speed measurements. A 300% speed improvement compared to a software based implementation of the same algorithm was observed when the design was tested in a XC3S1000 FPGA operated with a 100 MHz clock.

Keywords: Dedicated hardware, FPGA, Monte Carlo simulation, VHDL, Xilinx.

INTRODUCTION

Diffusion-limited aggregation (DLA) is the process of cluster growth by particles undergoing a random walk due to Brownian motion. The theory of DLA, proposed by Witten and Sander in 1981[1], is useful in explaining the aggregation of particles in any system where diffusion is the primary means of transport. It is interesting to see how this very simple algorithm generates highly disordered patterns with nontrivial scaling behaviours, observed in vastly different fields. For example, DLA algorithms have been used to generate patterns very similar to those observed in electro-chemical deposition processes and in dielectric breakdown in solids[2-4]. The DLA approach was found to be extremely useful in the study of the formation of snowflakes[5]. Both qualitative observations and quantitative analysis (such as measurement of fractal dimension) argue in favour of DLA growth in the formation of crystals in magnetic systems[6].

Computer simulations are widely used to study the systems exhibiting DLA[7-9]. Several methods are available to accomplish this. One approach is to place particles in a lattice of any desired geometry and to simulate their aggregation due to sticking together while they perform a random walk. First, a seed particle is placed in the center of the lattice and another particle is released from a random location far away from the seed. The second particle undergoes a random walk until it reaches an adjacent cell of the center seed which either sticks to the seed particle or moves out of the lattice (infinity). Then, a third particle is released to perform the random walk until it sticks to the growing cluster or move to infinity. When this process is repeated many times, a beautiful pattern, which is highly branched and fractal, is formed.

Since most of the time the particle moves around in the lattice without coming to a contact, and rarely comes to the vicinity of the growing cluster, where it will stick, the DLA algorithms are highly time consuming especially when the lattice size is large. This is partly due to the fact that the architecture of conventional computers is ill suited to run DLA models. This supports the idea of using programmable specialized processors such as those based on field Programmable Gate Arrays (FPGAs) attached to general purpose machines to execute time consuming applications.

An FPGA contains an array of logic gates and storage elements in which the functionality of the FPGA can be configured by downloading a bit-stream that defines the interconnections among the logic gates into its configuration memory. A "program" stored in an FPGA is not a chain of commands which are

* Corresponding author (upul@phys.cmb.ac.lk)

executed sequentially, but a description of the hardware configuration for a specific algorithm.

In recent years FPGAs have grown in capacity, improved in performance and decreased in cost, making them a viable solution for performing computationally intensive tasks with the ability to tackle applications for custom chips and programmable digital signal processing (DSP) devices[10]. FPGA technology allows arbitrary precision floating point arithmetic while retaining hardware speed. Recent work on matrix related computations indicate that FPGAs will soon be able to significantly outperform modern microprocessors because of the advantages in memory bandwidth and in floating point performance[11]. Hardware accelerators using reconfigurable logic are already in use in high performance computing (HPC) systems where coarse-grained parallelism of the microprocessors is combined with the fine-grained parallelism provided by FPGAs to achieve higher performance. These emerging hybrid HPC systems allow one to partition the applications such that most critical components are mapped onto hardware while the other parts of the application is executed in a microprocessor as software[12].

Due to these reasons there is a growing interest in converting existing computationally intensive algorithms such as Fast Fourier Transforms and Monte Carlo simulations into FPGA based implementations[10-15]. In this work, the development of a DLA algorithm on a two-dimensional lattice, implemented in a development board based on a Xilinx Spartan 3 FPGA is described.

METHODS AND MATERIALS

The major challenge of the hardware implementation of DLA is porting the algorithm into hardware component blocks. The basic structure of the hardware system can be broken down into four main components: memory module, random number generators, computer interface and control unit. These modules are separately tested and combined to form the entire system.

The memory module is used for the 2-D lattice space or the structure of the DLA where the random walkers will move about. The occupied lattice cell can be represented by setting the relevant memory location to logic 1 and an empty lattice cell can be represented by logic 0 at the relevant memory location.

2-D register array in the programmable device represents the 2-D lattice of the DLA. However, VHDL has features to create 2-D register arrays only for functional simulations. Synthesis tools often fail to compile VHDL codes with a 2-D register array definition.

One alternative to this is to create a 1-D register array as a RAM module and map the 2-D lattice into the RAM module. In this method a particular lattice coordinate should be transformed to the memory address to access the relevant memory location. Equation 1 shows the transformation equation of a lattice cell at (x,y) to the relevant memory address.

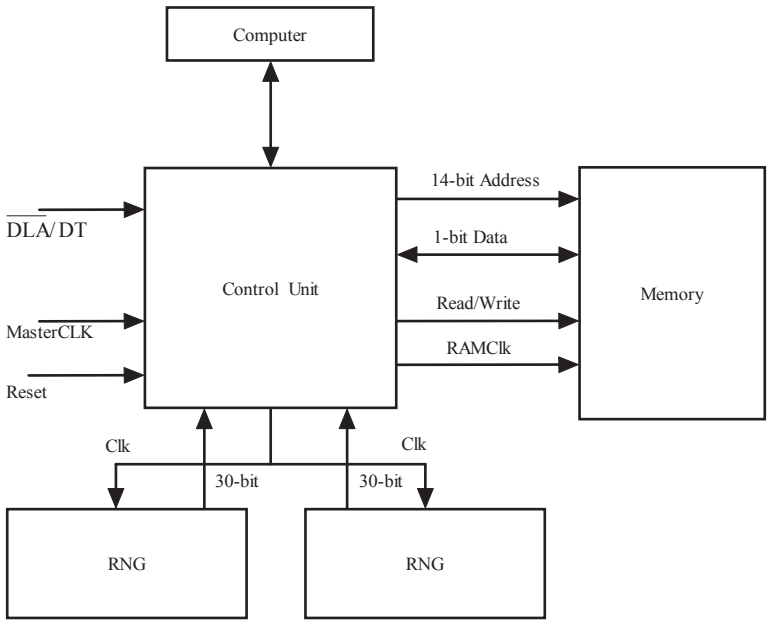

Figure 1: Block diagram of the DLA system

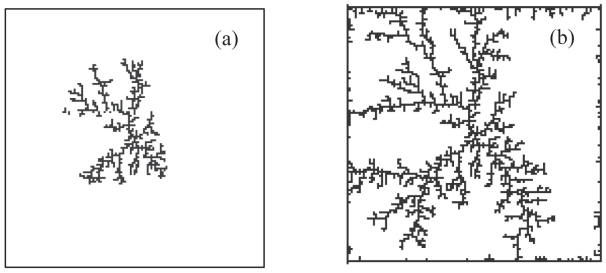

Figure 2: Hardware generated DLA patterns (a) 51 million cycles (b) 80 million cycles

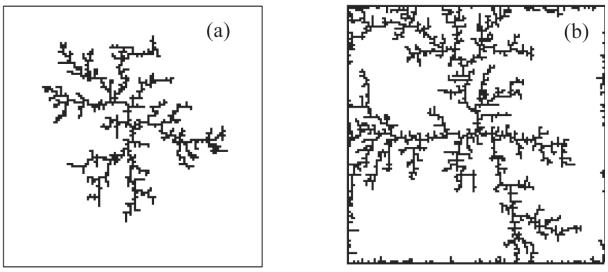

Figure 3: Software generated DLA patterns (a)51 million cycles (b) 80 million cycles

$$\text{Memory address} = y \times \sqrt{D} + x \quad \ldots (1)$$

where D is the capacity of the 2-D square lattice.

The major disadvantage of the above simple transformation is that it contains a multiplication operation. Whenever a memory address is calculated, it is necessary to perform a multiplication operation and an addition operation. Since the DLA processor needs to access memory very frequently, this can affect the overall speed significantly. Although for modern FPGAs, which have embedded multipliers, the use of logic resources would not be a problem, the speed degradation can still be a problem.

In an attempt to find a solution to the above problem, it was necessary to develop an alternative method that does not use any mathematical operation for the transformation. Since a DLA implementation on a 128 × 128 lattice required a total of 16,384 memory locations, a 14-bit address bus is needed to access them. However, we can split the address word into two parts with equal bit lengths. The first 7- least significant bits (LSBs) can be chosen to represent the x-axis of the lattice while the 7- most significant bits (MSBs) can be chosen to represents the y-axis of the lattice. For example, if the system needs to access the cell at (10,5), the memory location that represents that cell is at the memory address 00001010001010.

The advantage of this memory mapping technique is that it has no mathematical operations and is therefore very fast. Further, this method is suitable for any type of device. Embedded RAM modules available in modern FPGA devices are more efficient compared to memory created using VHDL since embedded RAM have been optimized for performance[16]. Therefore, Block RAM available in Spartan 3 FPGA were used to map the DLA lattice.

The random number generator (RNG) is the second main component of this system. RNGs are used to randomly select a position in the boundary when a new walker starts its walk and to select the new direction to move in each step of its random walk. Two random number generators give the x-position and the y-position for the walker. Although there are several methods for hardware based RNGs, the linear feedback shift register (LFSR) RNG is the most simple and the fastest generator[17]. However, the sequence generated by this generator repeats after a certain number of cycles. This means, a n-bit LFSR generator produces a sequence which repeats after 2^n-1 cycles. For example, a 30-bit generator repeats the sequence in 1,073,741,823 cycles. Because n is chosen as a large number, then the period of the sequence becomes very large. Therefore a 30-bit LFSR random number generator was selected for this work.

Figure 1 shows the block diagram of the DLA system. When the $\overline{DLA}/DT$ input is at logic 0, the DLA system starts the process. Initially all memory cells are set to zero. The process starts with setting the centre cell to logic 1. Then, a position on the boundary of the lattice was chosen where the random walker starts its journey. To select this position, two random numbers generated by the two RNGs were used. The system next checks whether the four adjacent cells, top, bottom, right and left, are occupied or not. It reads four memory locations relevant to the four adjacent cells in consecutive clock cycles and adds memory contents to a register called sum. If the sum is greater than zero, then one or more adjacent cells are already occupied and forces the walker to freeze in the current location. Then a new walker starts from the boundary of the structure. If sum is zero the walker moves into one of the four adjacent cells. This move is also selected randomly. Two bits of a 30-bit random number determines the new cell as shown in Table 1.

After the walker moves into the new cell, it checks the neighbours again as mentioned above. This process

Table 1: Two random bits determine random walker's new position

Random bits	New position
00	Top
01	Bottom
10	Left
11	Right

repeats until the walker reaches the growing cluster and sticks or walks out of the boundary of the lattice. When walking out of the boundary, the walker enters the lattice at the opposite side. In parallel to the DLA process there is a counter to track the number of clock cycles being used by the system. The DLA process stops after a certain number of predefined cycles or setting $\overline{DLA}/DT$ input to logic high. A seven segment display is used to indicate the end of execution.

After completing the execution of the DLA algorithm on hardware, the memory contents which contain the generated pattern needs to be transferred to the computer to view the pattern. $\overline{DLA}/DT$ signal is set to logic 1 to feed the processed data to the computer. The frequency of the master clock (MasterCLK) is about 100 MHz. It is not suitable for use in data transfers to the PC through the parallel port since the speed is quite low. Therefore, a separate clock signal is needed for the purpose. A multiplexer was used to select the clock signal of the RAM module for the two events - DLA process which uses MasterCLK and data transfer process which uses a clock signal from the computer (PCClk). When the input is at logic 0, RAM module is clocked by MasterCLK while the $\overline{DLA}/DT$ is at logic 1, the RAM module is clocked by the PCClk.

In the data transfer process, a memory value is sent through the data bus to the computer in every clock pulse. A 14-bit counter generates the address value to read the memory. Therefore, after 16,384 clock cycles, all memory values are fed into the computer. A Visual C++ programme running on the computer sends the clock signal to the system through the CONTROL port and reads the data value through the STATUS port and stores in an array. After completing the read operation, it draws the pattern on the screen.

The same DLA algorithm was implemented using a Visual C++ routine in order to compare the speed performance with the hardware counterpart. The dimensions of the DLA lattice and the number of execution cycles were identical to the hardware implemented system.

Table 2: Resource unitization summary

Resource	Used	Available	Utilization
Number of slice flip flops	160	15,360	1%
Number of 4 input LUTs	271	15,360	1%
Number of occupied slices	180	7,680	2%
Number of block RAMs	1	24	4%

To measure the execution time of the software routine GetTickCount function was used. It is an API (application programming interface) which retrieves the number of milliseconds that have elapsed since the system was started up. This function is called just before and immediately after executing the DLA algorithm. The time difference given by the two instants approximately equals to the execution time of the DLA algorithm in milliseconds.

RESULTS AND DISCUSSION

The DLA simulation algorithm discussed above was implemented on Xilinx Spartan 3 family (XC3S1000) FPGA, which comes with the XSA 3S1000 development board available from XESS Corporation[18]. The XC3S1000 has 1 million equivalent configurable logic gates. Xilinx ISE WebPACK version 7.2[19] was used to synthesize the hardware circuit using VHDL and Modelsim XE III was used to simulate the VHDL codes. Other software programmes were written in Visual C++ 6.0. The size selected for the DLA structure was limited to 128 × 128 for simplicity.

Figure 2 shows patterns obtained from executing the DLA algorithm in hardware. The pattern shown in Figure 2a and Figure 2b were obtained by executing 51 million cycles and 80 million cycles respectively. When the initial seeds of the random number generators were changed, the structure of the DLA pattern was also changed. The two patterns shown in Figure 2a and 2b have the seed particle at the center of the lattice but with different initial seeds of the random number generators.

Since the pattern shown in Figure 2b has been executed over a large number of cycles, the cluster has grown even along the boundary of the lattice. Once the boundary cells are occupied, new random walkers cannot pass through those occupied cells and thus, the pattern tends to grow fast.

The patterns obtained from the software implementation are shown in Figure 3a and 3b for the

same number of cycles (51 million and 80 million) but with different seeds with random number generation. The structure of the patterns are similar to the patterns obtained from hardware implementation.

As expected, similar to the hardware implementation, the software implemented DLA pattern shown in Figure 3b generated with a large number of cycles also tends to generate occupied cells along the boundary.

It should be noted that the total number of occupied cells in the patterns generated by the software implementation could be different to the patterns generated by the hardware implementation due to the two different random number generators used in the hardware implementation and software implementation. In the hardware system, LFSR random number generator was used while in the software system, random number generating function in Visual C++ library was used.

The results shown in Table 2 were obtained from the Xilinx ISE tool's report after synthesizing the whole system. It can be seen that the logic utilization for the DLA implementation is very low.

The dimensions of the implemented DLA system was 128 × 128. To represent the total number of cells of the lattice which is 16,384, only a single Block RAM out of 24 was used. Since other logic resources were mostly unused, the dimension of the DLA lattice can be easily extended.

From the simulation algorithm implemented in hardware, the parameter of interest is the execution time. This was calculated by executing the DLA algorithm for a known number of cycles. Since the maximum frequency of the FPGA board is fixed, it is easy to find the execution time. For the 51,005,100 (51 million) cycles, the average execution time was 510 ms. The average time taken to transfer the processed data (DLA pattern) to the computer was estimated as 265 ms. So the total time for the whole process was 776 ms. For the software simulation the average execution time was recorded as 2,168 ms on a Pentium-IV 3 GHz computer. Hence the speed of the hardware implemented system is better by a factor of 3 (200% speed improvement) than the software implemented system. If the time to transfer the data to the computer is not considered, an improvement of a factor 4 (300% speed improvement) in the execution time was obtained. By using a faster hardware, the performance could be further improved.

CONCLUSION

In this work, the implementation of diffusion limited aggregation algorithm in field programmable logic array is presented. The major challenge was to port the algorithm into digital circuit blocks. For simplicity, the lattice size was limited to 128 × 128. The DLA lattice was mapped to one of the Block RAM units available on the Spartan 3 FPGA. Mathematical operations, specially multiplication operations, were avoided for transformation of lattice cells to memory. This may have added higher speed performance to the hardware system.

The same algorithm was implemented using Visual C++ in order to compare the performance with the hardware based system. The DLA patterns obtained from both methods have similar features. However, hardware system showed about 300% speed performance compared to the software implementation. The work supports the idea of implementing programmable specialized processors attached to general purpose machines in undertaking highly time consuming applications. One of the drawbacks of the implemented system was the dependency on general purpose machines to transfer the memory contents once the execution is over.

The FPGAs have inherent capability of implementing systems optimized for parallel processing which is hardly possible in general purpose computers. Many orders of speed improvements could be easily obtained if the DLA algorithm is implemented on FPGA using the parallel processing capability. In order to do this, the algorithms must be modified so that many threads can be initiated to proceed simultaneously. For example, one could release many random walkers simultaneously from different locations on the boundary and they could walk simultaneously and independently until they stick on the main growing cluster. It would also be interesting to study the difference when the random walkers meet and if they stick together to initiate new growing clusters within the lattice.

Acknowledgement

Financial assistance by the National Science Foundation (research grant number RG/2007/FS/03) and the Wayamba University of Sri Lanka (research grant number RG/2005/07) are acknowledged.

References

1. Witten T.A. & Sandler L.M. (1981). Diffusion-limited aggregation, a kinetic critical phenomenon. *Physical Review Letters* **47**: 1400-1403.
2. Sagués F., Mas F., Vilarrasa M. & Costa J.M. (1990). Fractal electrodeposits of zinc and copper. *Journal of Electroanalytical Chemistry* **278**(1-2): 351-360.
3. Léger C., Servant L., Bruneel J.L. & Argoul F. (2002). Growth patterns in electrodeposition. *Physica A: Statistical Mechanics and its Applications* **263**(1-4): 305-314.
4. Irurzuna I.M., Bergeroa P., Molaa V., Corderoa M.C., Vicentea J.L. & Molaa E.E. (2002). Dielectric breakdown in solids modeled by DBM and DLA. *Chaos, Solitons & Fractals* **13**(6): 1333-1343.
5. Goold N.R., Somfai E. & Ball R.C. (2005). Anisotropic diffusion limited aggregation in three dimensions: universality and nonuniversality. *Physical Review* E **72**: (3):031403.
6. Perugini D. & Poli G. (2007). Tourmaline nodules from Capo Bianco Aplite (Elba Island, Italy): an example of diffusion limited aggregation growth in a magmatic systems. *Contributions to Mineralogy Petrology* **153**: 493–508.
7. Benjamini I. & Yadin A. (2008). Diffusion limited aggregation on a cylinder. *Communications in Mathematical Physics* **279**(1): 187–223.
8. Tang Q. (2008). Scaling behaviour of diffusion limited aggregation in percolation cluster, *Modern Physics Letters B*. **22**(7): 507–513.
9. Ferreira Jr. S.C. (2004). Effects of the screening breakdown in the diffusion-limited aggregation model. *European Physical Journal* B **42**(2): 263–269.
10. Uzun I.S., Amira A. & Bouridane A. (2005). FPGA implementations of fast Fourier transforms for real-time signal and image processing. *IEE Proceedings-Vision Image and Signal Process* **152**(3): 283-296.
11. Ho C.H. (2007). Customizable FPGA platform for accelerating floating point computations. Department of Computing, Imperial College, London.
12. Gothandaraman A., Peterson G.D., Warren G.L., Hinde R.J. & Harrison R.J. (2008). FPGA acceleration of a quantum Monte Carlo Application. *Parallel Computing*. **34**(4-5): 278-291.
13. Mark J., Zahi N., Plassmann P. & Yi Y. (2006). The use of configurable computing for computational kernels in scientific simulations. *Future Generation Computer Systems* **22**(1-2): 67-79.
14. Fragner H. (2007). Usage of a reconfigurable computer to simulate multiparticle systems. *Computer Physics Communications* **176**(5-1): 327-333.
15. Belletti F. *et al.* (2008). Simulating spin systems on IANUS, an FPGA-based computer. *Computer Physics Communications* **178**(3): 208-216.
16. Xilinx (2005). Complete Data sheet: Spartan 3 Family FPGA, DS099.
17. Wijesinghe W.A.S., Jayananda M.K. & Sonnadara D.U.J. (2006). Hardware implementation of random number generators. *Proceedings of the Technical Sessions of the Institute of Physics* **22**: 28-38.
18. X Engineering Software Systems Corporation. *http://www.xess.com*. Accessed on October 27, 2010.
19. ISE WebPACK software. *http://www.xilinx.com/ise/logic_design_prod/webpack.htm*. Accessed on Octobor 27, 2010.