International Journal of Advanced Computer Technology (IJACT)

ISSN:2319-7900

PARALLEL COMPUTING WITH GPU'S FOR HIGH SPEED MONTE CARLO SIMULATIONS USING FPGA

N. Bhavya Raj, M. Tech., CMRIT, Bangalore, India

Abstract

In present day technologies, due to the increase in number of elements present in a device. The performance of the device is also very important. To increase the performance of a device, the computations must be very fast. To achieve high computational time, here we are using Monte Carlo (MC) methods. Monte Carlo simulations are statistical methods, which require particular computation time. It can keep track of multiple physical quantities at a time simultaneously with desired spatial resolution. This is the main advantage, which makes Monte Carlo simulations a powerful method. Monte Carlo methods are considered to be the standard for simulated measurements of processing an image by photon transport for many medical applications. Monte Carlo simulations are now a much-used scientific tool for problems that are analytically intractable and for which experimentation is too time-consuming, costly, or impractical. This paper presents compute architecture for high speed Monte Carlo simulations using parallel processing"

Key Words

Monte Carlo simulations, parallel computing

Introduction

A Graphical Processing Unit (GPU) is an electronic circuit, which is designed for rapidly manipulating and altering the memory for accelerating the creation of images in a frame which is to be displayed as an output. Modern GPUs are most efficient in manipulating computer graphics. GPUs have parallel structures, which makes them more effective than general purpose CPUs (Central Processing Units) for algorithms where processing of large block of data is done in parallel. GPUs are used in embedded systems, mobile phones, personal computers, work stations, and game consoles. FPGAs (Field Programmable Gate Array) are programmable semiconductor devices that are based around a matrix of Configurable Logic Blocks (CLBs) connected through programmable interconnects. FPGAs can be programmed to the desired application or functionality requirements. Although One-Time Programmable (OTP) FPGAs are available, the dominant types are SRAM-based which can be reprogrammed as the design evolves.

Parallel processing is a computational form, in which calculations are carried out simultaneously. It operates on the principle that large instructions can be divided into smaller ones, which can be solved in parallel. Different forms of parallel processing are: bit-level, instruction level, data level, and task level parallelism. Parallelism has been mainly used for high performance computing. Due to the physical constraints preventing frequency scaling, the interest in it has developed very lately. As power consumption by computers has become a concern in recent years, parallel computing has become dominant in computer architecture, mainly in the form of multi-core processors**.**

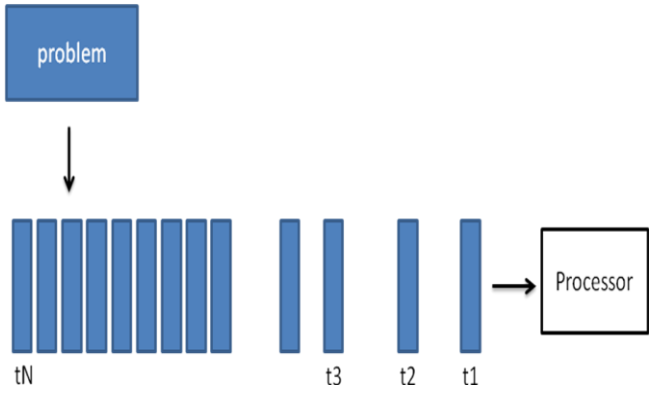

Figure1. Parallel Computation Processes

Monte Carlo simulations

Monte Carlo simulations are a class of computational algorithms which depends on repeated random sampling. Working principle of these simulations is, it runs many times in order to obtain the distribution of an unknown probabilistic entity. Monte Carlo methods are mainly used in three distinct problem classes: optimization, numerical integration and generation of draws from a probability distribution. It is true that MC simulations are good for parallelization, as basically they are comprised of huge numbers of independent experiments. These algorithms are executed by computer programs. Even though computer speed has been increasing drastically, Variance Reduction Techniques (VRTs) are needed. Increased computational power has initialised the

ISSN:2319-7900

analysts to develop more realistic models, such that the final result has not been faster than execution of simulation experiments. For example, some modern simulation models need hours or days for a single program to run. Because of this, modern computer would take more time to execute a single program. Variance reduction techniques can reduce these excessively long runtimes to practical levels.

Algorithmic verification

Algorithmic verification involves three different tasks, which are requirements specification, building executable system models, and developing scalable algorithms. Requirements characterize the expected behavior of a system. Verification methods require techniques for building executable models that faithfully represent a system or an abstraction of it. Algorithm verification can be done in any language such as C, C++, java, MATLAB (MATrix LABoratory). Here, we are verifying the algorithm using MATLAB, which computes in a numerical computing environment. It allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. MATLAB is a high-performance language for technical computing, which integrates computation, visualization, and programming environment. These factors make MATLAB an excellent tool for research.

Image Processing

Image processing is a form of signal processing for which the input is an image. The output of image processing may be either an image or a set of characteristics or parameters related to the image. An image may be considered to contain sub-images sometimes referred to as regions-of-interest, ROIs, or simply regions. This concept reflects the fact that images frequently contain collections of objects each of which can be the basis for a region. Before processing an image, it is converted into a digital form. After converting the image into bit information, processing is performed. This processing technique could be an Image enhancement, Image restoration, and Image compression.

A color image is a digital image that includes color information for each pixel. For visually acceptable results, it is necessary to provide three samples for each pixel. A grayscale image is an image in which the value of each pixel is a single sample, which carries only intensity information. They are mainly composed of gray shades, varying from black at the weakest intensity to white at the strongest. Whereas, black/ white (binary) image is a digital image that has only two possible values for each pixel. An entire class of operations on binary images operates on a 3×3 window of the image. This contains nine pixels, so 512 possible values. Considering only the central pixel, it is possible to define whether it remains set or unset, based on the surrounding pixels.

Project Description

This section describes about the system which is used for increasing the speed of the Monte Carlo simulations using parallel computing.

The Figure 2 shows the network and interconnection of each GPU which are used for parallel processing. Here, an image is taken which is divided equally into 3x3 blocks according to the concept of parallel processing. Each block is sent in parallel such that every data should be accessible by every element of the shared data set of a GPU for processing. Care must be taken such that the data is shared equally among the GPUs to avoid data starvation. To avoid this there should be connection between the Processing Units (PUs). This motivated to pair every PU to a 'switch' that would connect it to the communication fabric.

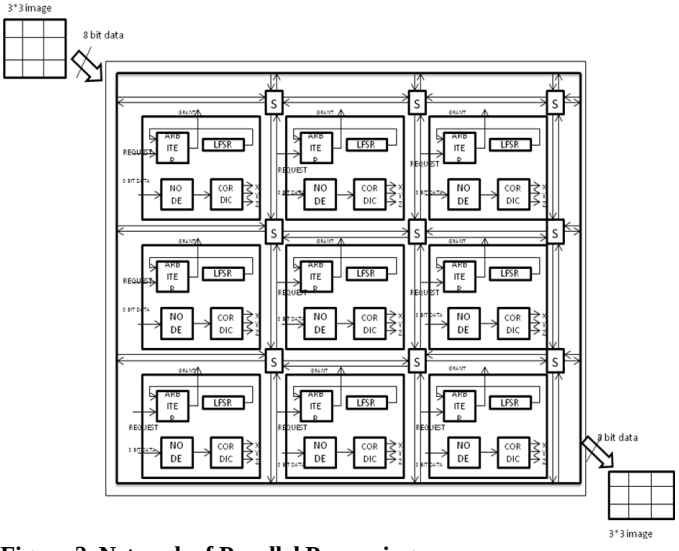

Figure 2. Network of Parallel Processing

ISSN:2319-7900

Block Diagram

The block diagram of the system which is used is shown in the Figure 3. Here, we are taking two images of which one is healthy and other is unhealthy. That is one image is reference image and the other is the image which is to be reconstructed that is unhealthy image. Next is the process of acquiring the image and splitting the whole image into 3x3 blocks. After splitting the image into 3x3 blocks, the image is given to the network node. In this node at a time, 9 pixels will be taken. Along with the node, an internal CORDIC processor is present; this is the network where the processing can be done faster. Here, the node performs the operation of how the data is to be transferred.

The main operation is performed by the CORDIC processor. CORDIC calculates the sin of the angle in terms of radians. For calculation of the angle in terms of sin for given input pixel, we are using CORDIC. The result for the given image will be stored in terms of sin. The same process will be carried by the unhealthy image and, we get the other sin value. By using these two sin values, we are performing Normalised Cross Correlation (NCC). Based on the cross correlated values, it will be calculated how much the healthy and unhealthy images are correlated. Since, we cannot create a 3D graphical image, we are checking how much unhealthy the image is when compared to the healthy image. Implementation of this simulation is called Monte Carlo simulation.

The basic CORDIC computations are expressed by the iterative equations at i^{th} step as follows

$$X_{i+1} = K_i (X_i - m\sigma_i 2^{-s(m,i)} Y_i) \quad (1)$$

$$Y_{i+1} = K_i (Y_i + \sigma_i 2^{-s(m,i)} X_i) \quad (2)$$

$$Z_{i+1} = Z_i - \sigma_i \alpha_{m,i} \quad (3)$$

Where i = 0, 1, …, N-1. The number of iterations 'N' will decide the fractional bit accuracy of the final result obtained, '*m*' parameter stands for one of the three coordinate systems namely linear, circular and hyperbolic (for m = 0, 1 and -1 respectively) and $S(m, i)$ is shift sequence having values 0,1, … ,N-1. Scale factor '*k*' remains constant for a particular computer if all rotations from 0 to N-1are completed. Parameter 'a' is angle, by which a vector is rotated in *i*th step and is given by,

$$\alpha_i = \tan^{-1} 2^{-i} \quad (4)$$

The parameter takes two vales, -1 and 1. If *Z*, the input angle, is left positive in a particular iteration step then its value is 1 otherwise -1.

Architectural Description

This section describes briefly about the internal architecture of the processing units and its interconnection.

The processing unit mainly consists of 3 blocks, which are arbiter, CORDIC and LFSR. There will be a node present in each PU to control the flow of inputs and outputs to and from the other processing elements. A switch is present to access the data between the two processors. To calculate the trigonometric ratio, a trig unit is required which is relatively expensive in terms of hardware resources. The trig unit is based on the well-documented CORDIC algorithm. The processing unit for this dissertation can be as shown in Figure 2.

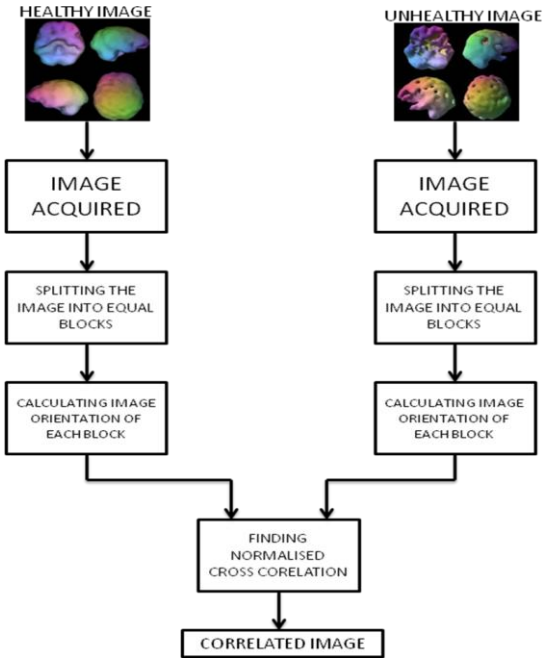

Figure3. Block Diagram of the System Used

CORDIC Processor

CORDIC (Coordinate Rotation Digital Computer), is an efficient algorithm used to calculate the hyperbolic and trigonometric functions. It is commonly used when no hardware multiplier is available as the operations it requires are addition, subtraction, bit shift and lookup table. CORDIC processor computes a number of functions including sine and cosine of angles.

ISSN:2319-7900

Figure 4 shows the basic architecture of the CORDIC processor. Arctangent values are constant and they are stored in ROM. For the iteration to go from ith stage to $(i+1)^{th}$ stage, the sign of Z_i has to be predetermined. Adder/subtracter is the only computational unit in *Z*- data path, latency of this architecture is determined by the latency of the adder/subtracter module. The delay time of the redundant adder is independent of the size of word and is approximately equal to delay time of two full adders. However, use of redundant number increases the hardware overhead and also it makes scale factor variable which has to be computed in each of the iterations.

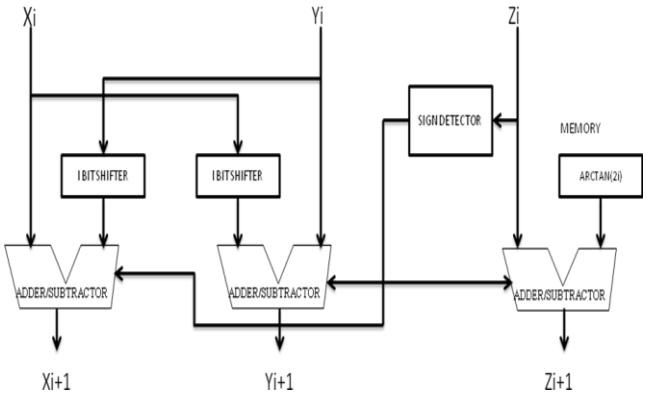

Figure 4. CORDIC Processor Structure

Arbiter

A bus arbiter (BA) is a device used in a multi-master bus system to decide which bus master will be allowed to control the bus for each bus cycle. The most common kind of bus arbiter is the memory arbiter in a system bus system. A memory arbiter is a device used in a shared memory system to decide, for each memory cycle, which CPU will be allowed to access that shared memory. An important form of arbiter is used in asynchronous circuits to select the order of access to a shared resource among asynchronous requests. Its function is to prevent two operations from occurring at once without overlapping. Given only one request, an Arbiter promptly permits the corresponding action, delaying any second request until the first action is completed. When an Arbiter gets two requests at once, it must decide which request to grant first. For example, when two processors request access to a shared memory at approximately the same time, the Arbiter puts the requests into one order or the other, granting access to only one processor at a time. The working principle and structure of the arbiter is sown in the figure below.

The generated BA consists of a D flip-flop, priority logic blocks, an M-bit ring counter and 'M' M-input OR gates as shown in Figure 6, where M=3. A 3x3 priority logic block is implemented in combinational logic implementing the logic function. The priorities of inputs are placed in descending order from in (0) to in (2) in the priority logic blocks (Priority Logic 0 through 2). Thus, in (0) has the highest priority; in (1) has the next priority, and so on. To implement a BA, we employ the token concept from a token ring in a network. The possession of the token allows a priority logic block to be enabled. Since each priority logic block has a different order of inputs (request signals), the priority of request signals varies with the chosen priority logic block.

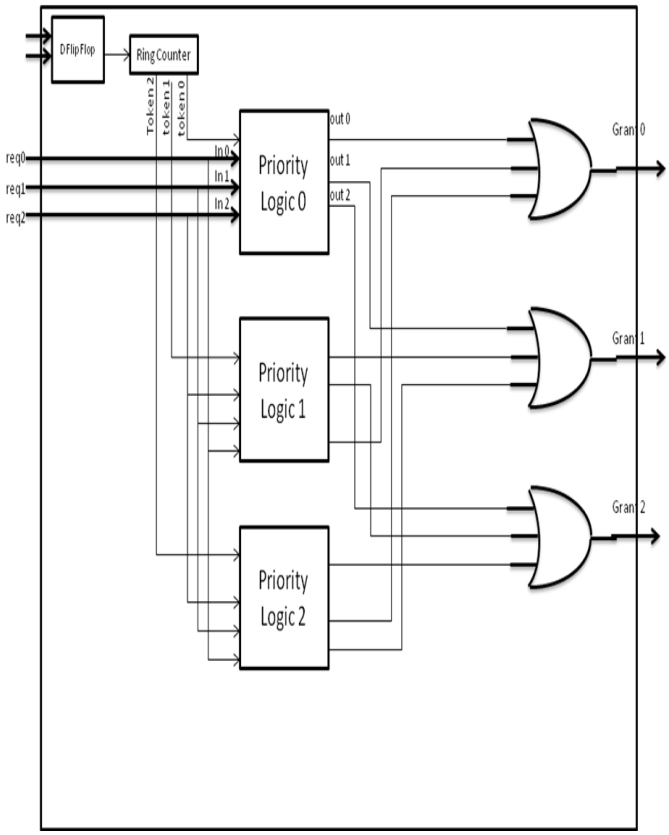

Figure6. Basic Structure of Arbiter

Linear Feedback Shift Register

A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is exclusive-or (XOR). Thus, an LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value. LFSRs sequence through (2^N

ISSN:2319-7900

− 1) states, where N is the number of registers in the LFSR. The contents of the registers are shifted right by one position at each clock cycle.

The linear feedback shift register is made up of two parts: a shift register and a feedback function. The shift register is initialized with n bits (called the key), and each time a key stream bit is required, all of the bits in the register are shifted 1 bit to the right. So the least significant bit is the output bit. The new left-most bit is computed as the XOR of certain bits in the register. This arrangement can potentially produce a 2n-1 bit-long pseudo-random sequence (referred to as the period) before repeating.

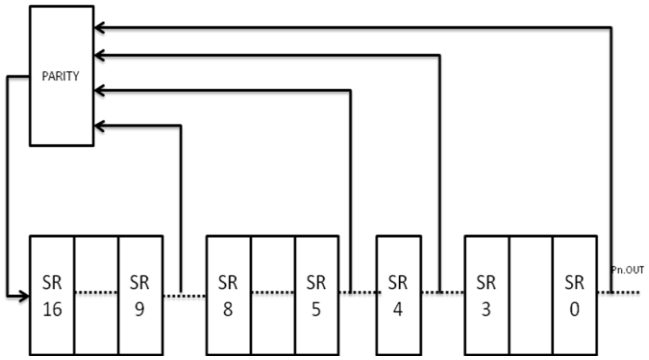

Figure7. Linear Feedback Shift Register

Node and Switch

The size of the network has significant influence on the decision for the routing and switching strategies that were selected. One of the best strategies for guaranteeing no data loss in the network with relatively few resources is the Wormhole (WH) switching technique. We have implemented a slightly modified version of WH that trades some network resources for overall network throughput, thus leaving latency unaffected.

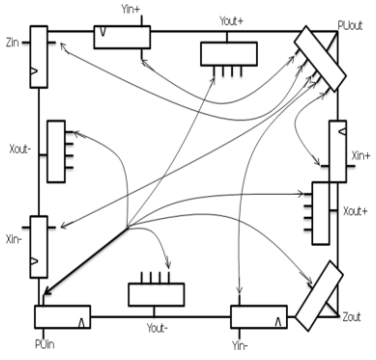

Figure8. Node of the Network

FPGA Implementation and Results

FPGA or Field Programmable Gate Arrays can be programmed or configured by the user or designer after manufacturing and during implementation. Hence they are otherwise known as On-Site programmable. Unlike a Programmable Array Logic (PAL) or other programmable device, their structure is similar to that of a gate-array or an ASIC. Thus, they are used to rapidly prototype ASICs, or as a substitute for places where an ASIC will eventually be used. This is done when it is important to get the design to the market first. Later on, when the ASIC is produced in bulk to reduce the NRE cost, it can replace the FPGA. The programming of the FPGA is done using a logic circuit diagram or a source code using a Hardware Description Language (HDL) to specify how the chip should work.

The process which is described in the block diagram is implemented. After obtaining the values of the healthy and unhealthy images, both the images are cross correlated and the output obtained after correlating the healthy and unhealthy images is as shown in figure below. The black lines represent the dissimilarities in the healthy and unhealthy images.

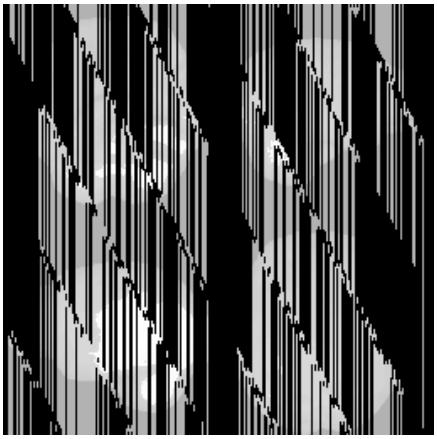

Figure9. Obtained Correlated Image

The following table 1 describes about the utilization of the devices used in this system. Which includes number of devices used available devices and the percentage of the device utilization. Table 2 gives a brief idea about the performance summary.

ISSN:2319-7900

Table1. Device Utilization

	Used devices	Available devices	Utilization
Registers	1,730	7,168	24%
Input LUTs	2,803	7,168	39%
No. of slices	1,459	3,584	40%
Target Device	XC3S400	NA	NA

Table2. Performance Summary

Parameters	present work
Frequency (max.)	167.479MHZ
Minimum Time Period	5.971ns
Delay	2.278ns

Conclusion

In this dissertation, we have implemented architecture for the GPUs to be computed in parallel for increasing the speed of Monte Carlo simulations. Since, we are using parallel processing; the speed of the simulations will increase to the considerable extent. The main process of the proposed system will be carried out by the Cordic processor. This implementation of the CORDIC algorithm allows the device to perform in increased speeds. The significant speedups can be achieved over single core implementations, without compromising the image reconstruction accuracy.

Disadvantages

There is a drawback in this approach. The drawback of this architecture is that the obtained image cannot be realized properly. It means that, it is very hard to analyze the reconstructed image as it is represented in black and white lines, representing the similarities and dissimilarities obtained.

References

[1] F.J. Beekman, H.W.A.M. de Jong and S. Van Geloven, "Efficient Fully 3D Iterative SPECT Reconstruction With Monte Carlo- Based Scatter Compensation," IEEE Trans. Medical Imaging, vol. 21, no. 8, pp. 867-877, Aug. 2002.

[2] C. M. Kao and X. Pan, "Evaluation of Analytical Methods for Fast and Accurate Image Reconstruction in 3D SPECT," Proc. IEEE Nuclear Science Symp. Conf. Record, vol. 3, pp. 1599-1603, 1998.

[3] S. Zhao and C. Zhou, "Accelerating Spatial Clustering Detection of Epidemic Disease with Graphics Processing Unit," Proc. 18th Int'l Conf. Conf. Geoinformatics, pp. 1-6, June 2010.

[4] L. Xu, M. Taufer, S. Collins, and D.G. Vlachos, "Parallelization of Tau-Leap Coarse-Grained Monte Carlo Simulations on GPUs," Proc. IEEE Int'l Symp. Parallel Distributed Processing (IPDPS), pp. 19, Apr. 2010.

[5] F. Angarita, A. Perez-Pascual, T. Sansaloni, and J. Vails, "Efficient FPGA Implementation of Cordic Algorithm for Circular and Linear Coordinates," Proc. Int'l Conf. Field Programmable Logic and Applications, pp. 535-538, Aug. 2005.

[6] Jack E Volder, "The CORDIC Trigonometric Computing Technique" , pp 226,1959.

[7] Phillip J. Kinsman, Student Member, IEEE, and Nicola Nicolici, Senior Member, IEEE "NoC-Based FPGA Acceleration for Monte Carlo Simulations with Applications to SPECT Imaging" IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 3, MARCH 2013