Dawn of GPU Era-Potentials of Chaos Theory

Saurabh Maniktala, Anisha Goel, A. B. Patki, and R. C. Meharde

***Abstract*—In the present era Chaos theory has tremendous potential in Computer Science Domain. The true potential of Chaos theory can be realized with the assistance of high performance computing aids such as GPU that have become available in present times. The main purpose is to develop a high performance experimental laboratory in academic institutions, for analysis of chaotic systems. In this paper we present the parallel implementation of One Dimensional Logistic Map on NVIDIA GeForce 9400 GT Graphical Processing Unit (GPU) which is a CUDA enabled GPU as a first step towards this direction. We report that the GPU version of Logistic Map executes 495.08 times faster than the CPU only version.**

***Index Terms*—Chaos theory, CUDA, GPU, logistic map.**

I. Introduction

The field of chaos has fascinated researchers from all over the world since its inception in the 1900s. Traditionally, in any English dictionary the word chaos is defined as complete disorder or anarchy. On the contrary, from a research perspective Chaos has been explored as a discipline which enables us to study the apparently random behavior of nonlinear dynamic systems. By the word apparently we stress the fact that datasets appearing random or chaotic can be generated from a deterministic mathematical equation. The key features of a chaotic system include:

- bounded
- nonlinearity
- mixing
- sensitive dependence on initial conditions (SDIC)

The milestones achieved in this field include electronic circuit by Leon O. Chua, the study of atmospheric convection by Edward N. Lorenz and chemical kinetics by Otto Rössler to name a few. Chaos theory finds applications in various scientific disciplines including computer science, population dynamics, aviation, politics, philosophy, robotics, finance, biology, and psychology.

In the present scenario Chaos theory is playing an even more prominent role for developing computer science based applications. Some of the works in this domain include preemptive queuing models with delay [1] and encryption techniques using Baptista Methodology [2].

In this paper, we have taken the first step towards setting up of an experimental laboratory in academic institutions, including AICTE approved engineering colleges, for analysis of chaotic systems. The systems can be divided into two broad categories of continuous and discrete time. We have presented a parallel implementation of one dimensional logistic map of discrete time system on the Graphical Processing Unit (GPU). The GPU was chosen for the implementation as it has an enormous parallel computational capability that makes analysis faster and efficient. We have achieved a speedup of 495.08 times over the sequential implementation—which we have also presented in this paper for comparison purposes.

We have carried out the experiment using the CUDA enabled NVIDIA GeForce 9400 GT and CUDA Toolkit version 3.0. Other details are mentioned in Table I.

Section II presents a brief introduction to one dimensional logistic map with its sequential C based implementation, section III elaborates the concepts of GPU and CUDA, section IV presents the pseudo-parallel implementation of one dimensional logistic map, section V includes the performance measurement of the parallel implementation and section VI presents conclusion and future work.

II. One Dimensional Logistic Map and its Sequential Implementation

The logistic equation as applied to Chaos models the growth curve of some natural phenomena. As an example, the curve could depict the variation in size of a population of a particular area over time. The size of the population is measured in discrete steps by using (1).

$$x_{t+1} = kx_t(1-x_t) \quad (1)$$

TABLE I: Details of Platform used to Perform Experiment

PLATFORM DETAILS	
CPU	Intel Pentium Dual Core
OS	Windows XP
GPU	NVIDIA GeForce 9400 GT
Compute Capability	1.1
Graphics Bus	PCI-Express
CUDA Driver and Runtime Version	3.0
Clock rate of GPU	1.4GHz
Global Memory	1 GB

The variable x_t denotes the current size of population, x_{t+1} represents future size of population and k is a constant known as the control parameter. The equation (1) is of a feedback nature as the size of population at some time interval is used to generate the size of population at the next time interval. The control parameter scales the size of population. In order to limit the size within realistic bounds, a factor of $(1-x_t)$ is multiplied. Since there is only one variable, (1) is more aptly

Manuscript received July 13, 2012; revised August 15, 2012.

Saurabh Maniktala and Anisha Goel are with Maharaja Surajmal Institute of Technology, GGSIPU, New Delhi, India. (e-mail: sam.maniktala@gmail.com; anishagoel14@gmail.com).

A. B. Patki and R. C. Meharde are with Department of Information Technology, MCIT New Delhi, India (e-mail: abpatki@gmail.com; rmeharde@mit.gov.in).

referred to as one dimensional logistic equation. Equation (1), for simplicity, is normalized such that the size of population can only range from 0 to 1.

The one dimensional logistic map is obtained from the one dimensional logistic equation by plotting the values of x_{t+1} as y coordinates against values of x_t as x coordinates on a graph. Since the map is using two dimensions, namely x and y, to plot the points representing the same variable, the size of population, it is also called pseudo two dimensional.

The initial size of the population is denoted by x_0. As chaotic systems are sensitive to initial conditions x_0 plays an important role to enter into Chaos. Moreover, the control parameter k has to be tuned to the appropriate value for the onset of Chaos. The value of k governs the steepness and the height of the curve which is a parabola. As the values generated from the equation are normalized, the allowable range of k is (0,4).

The system shows varied behavior for different ranges of k as:

- For k<1, the system eventually converges to a zero size population, so in this case the attractor is zero.
- For 1<k<3, the attractor increases from point zero to 0.667. For 0<k<3 the population value sooner or later saturates irrespective of initial value.
- For 3≤ k < 3.57, period doubling starts. At k=3.4, the attractor splits into two point attractor. For greater values of k in this range the attractor keeps on doubling to produce 4, 8, 16… point attractors. In this range the final values keep on oscillating.
- For 3.57≤ k < 4, the system enters in chaotic region. In this case the attractor can be erratic with infinite number of points or stable.

We now present the sequential C based implementation of logistic map. We have developed a C function named *logisitcmap()* having the arguments x_0 as the initial value of population, k as the control parameter, y as the output array containing population sizes for consecutive iterations and N as the number of iterations. The function returns the output array y. Below we present the source code of *logisticmap().*

```
float* logisticmap(float x0, float k, float
y[], int N)
  {
    float temp=x0;
    for(int i=0;i<N;i++)
    {
      temp = k * temp * (1-temp);
      y[i] = temp;
    }
    return(y);}
```

Table II shows the output array for N=10 iterations, k=3.7 and two initial values of x_0 as 0.01 and 0.011. The value of k=3.7 drives the system into chaos and apparently random successive output values are generated. Notice that the output values arising out of the two initial values of x_0 differing by decimal in the third place diverge as the index of iteration increases. This behavior of chaotic systems as referred to as Sensitive Dependence on Initial Conditions.

TABLE II: OUTPUT VALUES FOR TWO INITIAL VALUES IN CHAOS STATE

Index	x_0=0.01	x_0=0.011
1	0.036630	0.040252
2	0.130567	0.142939
3	0.420020	0.453276
4	0.901332	0.916923
5	0.329051	0.281850
6	0.816873	0.748919
7	0.553488	0.695746
8	0.914415	0.783228
9	0.289564	0.628192
10	0.761152	0.864197

III. GPU AND CUDA

The Graphical Processing Unit (GPU) was traditionally designed to offload intensive graphical computations, including scaling, rotating, and rendering, from the Central Processing Unit (CPU). It essentially functions as a coprocessor to the CPU which directs work to it. Hence, the GPU is also known as the Device whereas the CPU is known as the Host.

The traditional GPU architecture consisted of pipelined stages of combination of programmable vertex shader and pixel fragment processors which are used for generating graphics on the screen. Subsequently, the vertex shader and pixel fragment processors were combined into a unified GPU architecture for efficient working.

The advances in the technology of GPU encouraged researchers to step into the direction of General Purpose GPU (GPGPU) programming. The GPGPU concept focuses on using the computational capability of GPU for non-graphical applications. The current GPUs have developed into a programmable parallel processor offering high performance parallel computing capabilities far exceeding that of current multi-core CPUs. The design goal of GPU is to optimize execution of massive number of threads in parallel and hence a larger chip area is dedicated for floating point operations. However, as GPU works as a coprocessor for host CPU, it will not outperform on other jobs that CPU is designed.

The greatest challenge that was faced was to use these numerous cores in an optimized fashion. In order to overcome such difficulties, NVIDIA Corporation developed a software environment named CUDA (Compute Unified Device Architecture) that extends the C language syntax [3], [4].CUDA enables programmers to develop programs that can scale across the cores of a GPU while using the familiar syntax of C language. CUDA package includes a Runtime API and a Driver API. The CUDA runtime API extends C language syntax with function commands dedicated for GPU and is referred to as CUDA C. The Driver API, on the other hand, is a low level procedural API but is totally different from the graphical APIs such as Open GL and DirectX. The OpenGL and DirectX APIs are traditionally used for GPU programming for graphical applications. But the situation gets complicated when a programmer wants to deploy GPU to develop a parallel program for non graphical applications and had to put unnecessary effort of studying the graphic libraries is required. Thus CUDA C fulfills the programmer's needs. We explain the concepts of GPU in the context of CUDA programming.

Fig. 1. shows the architecture of an NVIDIA GeForce 8

series GPU. The basic unit shown is a Texture/Processor Cluster (TPC) which incorporates a Streaming Multiprocessor Controller (SMC), a texture memory unit, and 2 Streaming Multiprocessor (SM) blocks [5]. The Multiprocessor handles continuous flow of data streams and is thus named as a Streaming Multiprocessor. The SMC is responsible for work distribution amongst the two SMs. Each SM is responsible for executing hardware threads that perform computations. The texture memory unit inside the TPC is generally used for storing pixel values. Each SM houses 8 Streaming Processors (SP), 2 Special Function Units (SFUs), a Cache Unit and a 16 KB shared memory unit. The actual floating point computations take place in the SP which is essentially a single core handling a single hardware thread. A hardware thread is the smallest unit of computation. The SP utilizes cache memory for faster lookup of data, while the inter-SP communication in a SM takes place via the shared memory. The GPU actually consists of an array of TPCs. In addition to the above, the GPU also includes global memory unit and constant cache unit on the same chip. The number of TPCs and the size of global and constant cache memory depend on the GPU series.

The concurrent execution of the threads on the SPs of a SM forms the basis of parallel computation. Since a fixed number of threads can run concurrently on a SM, groups of 32 threads are made from the bank of threads and are scheduled to run in an ordered manner. From a hardware view point, these groups of 32 threads are known as Warps. At a particular instance of time only one warp can be active.

From the CUDA programming view point, the program is broken into sequential and parallel modules. For parallel modules, threads are organized into two level of hierarchies namely blocks and grids [6], [7]. The first hierarchy, block is a software programming concept wherein threads are organized in a 3 dimensional layout (as *x*, *y*, and *z* dimensional representation).

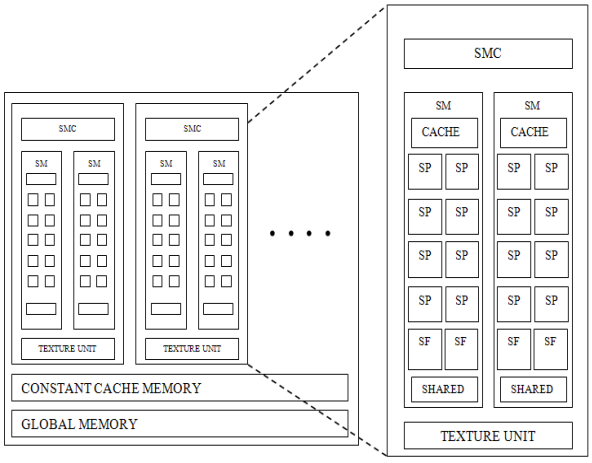

Fig. 1. Basic GPU architecture

The second hierarchy grid is also a software concept wherein the first hierarchy blocks are further organized in a two dimensional layout. The threads in a block execute cooperatively in parallel on the SM, whereas the blocks of given grid execute independent of each other across the GPU. On the contrary, the grids execute sequentially on a GPU. The independent execution of blocks forms the basis of coarse-grained parallelism and cooperative parallel execution of threads in block implements fine-grained parallelism.

Each grid is formed while executing a kernel call. The CUDA kernel is the program module to be executed on the GPU and is typically invoked from host CPU. Multiple copies of the same program module are created and are assigned to each thread at the time of kernel call. We have to specify the dimensions of the grid and the block in the execution configuration at the time of the kernel call. The programmer is free to experiment with the dimension sizes of blocks and grids. The total number of threads in a grid and block is calculated by using (2).

$$T = \text{grid\_dimension} \times \text{block\_dimension} \quad (2)$$

where T is the total number of threads, grid_dimension denotes the total number of blocks in a grid and block_dimension denotes the total number of threads in a block. For example if grid is to have 6 blocks (grid_dimension=(3,2) organized as 3 blocks in x direction and 2 blocks in y direction) and further each block has to have 48 threads (block_dimension=(6,4,2) organized as 6 threads in x direction, 4 in y direction and 2 in z direction) totaling 48*6= 288 threads in all, then the kernel call will be given as:

```
dim3 grid_dimension (3,2)
dim3 block_dimension(6,4,2)
```

Another alternative allotment for same problem could be:

```
dim3 grid_dimension (1,1)
dim3 block_dimension(288,1,1)
kernel_function_name<<<grid_dimension,
block_dimension>>> (list_of_arguments);
```

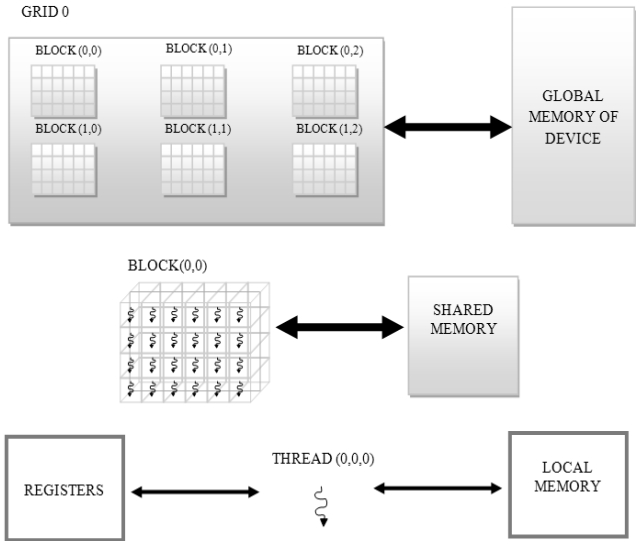

Fig. 2. Kernel execution memory allocation

TABLE III: OUTPUT VALUES FOR TWO INITIAL VALUES IN CHAOS STATE IN PARALLEL IMPLEMENTATION

Index	X_0=0.01	X_0=0.011
1	0.036630	0.040252
2	0.130567	0.142939
3	0.420020	0.453276
4	0.901332	0.916923
5	0.329051	0.281850
6	0.816873	0.748919
7	0.553488	0.695746
8	0.914414	0.783228
9	0.289565	0.628192
10	0.761153	0.864197

TABLE IV: GPU AND CPU EXECUTION TIME

No. of iterations	CPU(MSEC)	GPU(MSEC)
128	50.000	0.115
256	60.000	0.126
384	70.000	0.148
512	90.000	0.168
640	110.000	0.199

The performance issues resulting out of the above two alternative representations for 288 threads are beyond the scope of this paper. Fig. 2. illustrates the first alternative showing the arrangement of threads in blocks and blocks in grid. It also depicts the memory visible to each level of hierarchy. The local memory is only visible by its corresponding thread. Also, the threads in a block can communicate using shared memory. Lastly all the blocks in the grid can see the global memory.

IV. PARALLEL IMPLEMENTATION OF LOGISTIC MAP

In this section we present the parallel GPU based implementation of the one dimensional logistic map.

We have developed a kernel *logisticmap()* having the following arguments x_0 as the initial value of population, k as the control parameter, and y as the output array containing population sizes for consecutive iterations. Below we present the code snippet of *logisticmap()*:

```
__global__ void logisticmap(float *y,
float x0, float k)
  {
    __shared__ float temp;
    temp = x0;
    int i;

       int idx = blockIdx.x*blockDim.x +
       threadIdx.x;

    for(i=0;i<=idx;i++)
    {
      temp = k*temp*(1-temp);
    }

    y[idx] = temp;
  }
```

The execution configuration of the kernel call for a total of N iterations is given below:

```
dim3 block_dimension(2);
dim3 grid_dimension (N/2);
```

The call to the kernel *logisticmap()* is given as:

```
logisticmap<<<grid_dimension,block_dimen
sion>>>(y,x0,k);
```

Table III shows the output array generated by the kernel *logisticmap()* for N=10, k=3.7 and initial values of x_0 as 0.01 and 0.011 for comparison with Table II. There is a difference in the sixth decimal place in last the 3 values for x_0=0.01 from the corresponding values in the sequential implementation due to round off scheme employed by the Device instead of truncation employed by Host. This difference can be overcome using a Device of compute capability 2.0.

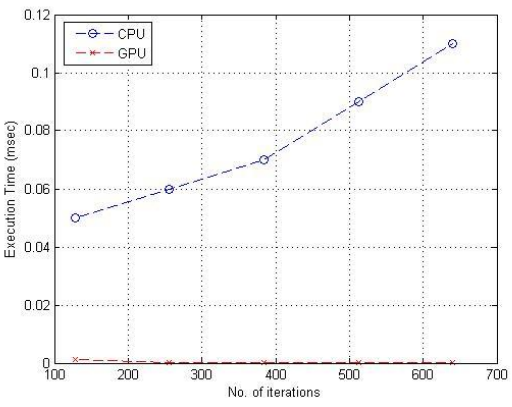

Fig. 3. GPU versus CPU Execution Time

V. PERFORMANCE MEASUREMENT

Now we present the comparison between the execution times of *logisticmap()* on GPU and CPU on the basis of the number of iterations performed. The execution times on GPU (parallel implementation) and CPU (sequential implementation) have been measured in milliseconds with precision of three decimal places and are enlisted in Table IV.

Fig. 3. depicts a graph generated using Table IV. The graph has been interpolated for the ease of understanding. As we observe from the figure the GPU execution time remains nearly constant over the entire range of iterations performed. On the contrary, the CPU execution time increases proportionally with the number of iterations performed. The most remarkable result that we have achieved is that on an average GPU computed the result 495.08 times faster than the CPU.

VI. CONCLUSION AND FUTURE WORK

The objective of this paper is to present a high performance parallel GPU based implementation of One Dimensional Logistic Map using the NVIDIA CUDA environment first step towards setting up of an experimental laboratory in academic institutions, including AICTE approved engineering colleges, for analysis of chaotic systems by developing one of the most fundamental tools for study. We have reported an enormous speedup of 495.08 times than the CPU only version. Towards this direction we have already explored potentials of Chaotic cryptography (Baptista's Methodology). Authors feel that this will be useful in offering an academic course with laboratory hands-on in high performance applications of Chaos.

ACKNOWLEDGMENT

We thank engineering college authorities to encourage student interns in participating in challenging projects at DIT. Authors also thank the DIT student interns for their efforts in pursuing the research and development work in High Performance Computing discipline.

REFERENCES

[1] P. Ranjan, S. Kumara, A. Surana, V. Manikonda, M. Greaves, and W. Peng, "Decision Making in Logistics: A Chaos Theory Based Analysis," in *Proc. AAAI Workshop on Decision Making*, March 2002, Stanford, California.

[2] Q. V. Lawande, B. R. Ivan, and S. D. Dhodapkar, "Chaos Based Cryptography: A New Approach To Secure Communications."
[3] CUDA Technology, NVIDIA. (2010). [Online]. Available: http://www.nvidia.com/object/CUDA.
[4] L. Nyland, M. Harris, and J. Prins, "Fast N-Body Simulation with CUDA, GPU Gems 3," In: Nguyen, H.; ed., Addison-Wesley, 2007, pp. 677-695.
[5] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA TESLA: A Unified Graphics and Computing Architecture," In: Micro, IEEE, 2008, 28, Issue: 2, Identifier: 10.1109/MM.2008.31, pp. 39–55.
[6] CUDA Programming Guide 3.0, NVIDIA. (2010). [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf.
[7] A. Goel and S. Maniktala, "High Performance Computing- GPU CUDA Methodology," Summer Internship Report No. DIT/DIR(RCM)/01/July 2010.

Mr. Saurabh Maniktala completed his Bachelor's in Electronics & Communication Engineering from Maharaja Surajmal Institute of Technology, Guru Gobind Singh Indraprastha University, New Delhi in 2011. He was selected for summer research internship, 2010 at Department of Information Technology, Govt. of India, Delhi under the aegis of Mr. A. B. Patki and Mr. R. C. Meharde. He has been involved in developing parallelized prototypes of algorithms including Data Encryption Standard, Max-Min operations for Type-I fuzzy sets, image processing filters and One-Dimensional Logistic Map in Chaos Theory for the CUDA platform. He is also co-author of the paper – A comparative study of various variants of SPARQL in Semantic Web.

He has developed an introductory course for initiating CUDA/GPU activities at the Bachelor level in engineering colleges. His current research interests include Computer Architecture, Operating Systems, Parallel GPU based computing and Chaos theory based applications.

Ms. Anisha Goel obtained her Bachelor of Technology from Maharaja Surajmal Institute of Technology, Guru Gobind Singh Indraprastha University, New Delhi in 2011 with specialization in Computer Science. She was selected for Summer Research Internship program at Dept of Information Technology, Govt. of India under Mr. A. B. Patki and Mr. R. C. Meharde, during the year 2010. She has been involved in developing parallelized prototypes of algorithms including Data Encryption Standard, Max-Min operations for Type-I fuzzy sets, image processing filters and One-Dimensional Logistic Map in Chaos Theory for the CUDA platform. She has also co-authored a paper — A comparative study of various variants of SPARQL in Semantic Web.

She has been instrumental in the development of an introductory course for initiating CUDA/GPU activities at the B.Tech level in engineering colleges. Her current research interests include Compilers, Operating Systems, Parallel GPU based computing and Chaos theory based applications.

Mr. A. B. Patki obtained his M Tech from Indian Institute of Technology (IIT), Kharagpur in 1975 with specialization in computers. He had worked as faculty in Government College of Engineering, Amravati, Maharashtra, during 1972-73. He also worked as Project Officer at IIT, Kharagpur during 1975-77 on hardware/software R&D projects. Since March 1977, he was employed with Department of Information Technology (erstwhile Department of Electronics), Government of India and superannuated in March 2010 as Senior Director/Scientist-G & HoD. He has worked in various capacities on several projects in the areas of Artificial Intelligence, Software Technology Parks, Reliability Engineering, VLSI Design, Soft Computing and E-Commerce. He was also instrumental in spearheading post-legislation activities of Information Technology (IT) Act, 2000. He has been member of Scientists selection committees in DRDO.

His current research areas include Soft Computing for Information Mining, Evidence Based Software Engineering, Professional Outsourcing, ICT enabled Productivity Enhancement, Cyber Laws, Cyber Ethics including Cyber forensics & Chaos Theory applications information security.

Mr. Patki has been trained in VLSI Design at Lund University, Sweden and Mentor Graphics, USA. He has been a referee for IEEE Transactions on Reliability for over twenty years. He holds a copyright for FUZOS©- Fuzzy Logic Based Operating Software. He has over fifty International publications. He has delivered invited lectures at leading academic institutions. He has developed courseware for VLSI design using VHDL that has been used for training the teaching faculties at leading engineering colleges and microelectronic industry. He has been supervising B.Tech/ ME thesis and also imparting training for Engineering Interns in Computer Science and Information Technology.

Mr. R. C. Meharde obtained his Bachelor of Engineering degree in Electronics with specialization in Control & Instrumentation, from SGS Institute of Technology and Science, Indore in 1978. He had worked with Hindustan Copper Ltd., a PSU from April, 1979 to January, 1981. In January, 1981 he joined the Directorate General of Light Houses and Lightships, Ministry of Shipping and Transport, Government of India. Since February, 1987, he is employed with Department of Information Technology, (erstwhile Department of Electronics), Government of India, which has been recently renamed as Department of Electronics and Information Technology by the Government of India.

He has worked extensively in field of Control & Instrumentation, Coastal Navigation System, Futuristic Navigation Systems. His current areas of interest include promotion of Electronics and IT systems and application in areas of Industrial Electronics, Automation System, Intelligent Transportation System and Power Electronics. He has contributed for evolving several R&D projects in these areas which have been successfully implemented and resulted into technology transfer amongst Indian Industry. Presently, he is Senior Director/Scientist G-HoD Electronics Systems Development & Applications (ESDA).