ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 8, August 2017

Parallel Computing of Ordinary Differential Equations

[1] Pooja S. Patil, [2] M. D. Patil, [3] Vishwesh A. Vyawahare
[1][2][3] Department of Electronics Engineering
[1][2] Ramrao Adik Institute of Technology , Nerul, Navi Mumbai

***Abstract* - Numerical methods for solving fractional differential equations are computationally heavy due to the need of floating-point operations, the non-local nature of the fractional differential operators and more importantly, the data flow inside the entire memory system of a computer. Hence such systems can be implemented on Graphics Processing Unit (GPU) which has the parallel computing power for quicker simulation. A GPU has a number of threads where each thread can execute different program. MATLAB and Parallel Computing toolboxe can be used to access the computational power of GPU and MATLAB code can be implemented on the GPU. This helps us to achieve significant & faster computation than a normal CPU system. In this paper an attempt is made to implement numerical method for simple fractional ordinary differential equation (FODE) on a Dual Core CPU and NVIDIA GPU. This paper presents the relative performance of a GPU v/s CPU for fractional Euler's method to solve FODE. From the results presented, it is observed that GPU provides two times speed up for fractional Euler's method.**

Index Terms— fractional differential equations, fractional ordinary differential equation, Graphics Processing Unit

I.INTRODUCTION

Fractional calculus is a 300 years old subject which involves the study of ordinary differentiation and integration to non integer order. Over the past three decades, it has gained huge popularity and fractional differential equations (FDEs) has been extensively used for the description various phenomena developing in engineering, physics and science [1]. With the rapid development of technology, the fractional calculus gets involved in more and more areas, especially in control theory, viscoelastic theory, electronic chemicals fractal theory and so on [2]. For FDEs analytical or approximate methods can be difficult or complex to solve and thus numerical methods are required [3]. In the past few decades, several numerical methods have been developed for solving FDEs such as the Legendre wavelet method, the spectral method and quartered shifted Legendre method based on GaussCLabatt [4].

In [8], differential equations are among the most important mathematical tools used in producing models in the field of physical sciences, biological sciences and engineering. Ordinary Differential Equations (ODEs) can be solved by numerical methods such as Euler's method, Runge Kutta methods and the families of Adams Bashforth and Adams Moulton methods where speed of processing is more desirable than the accuracy [5]. Among all numerical methods Euler's method has proved to be efficient solving ODEs [6].

In the field of numerical method such as Euler's method for fractional-order ODE running long computer simulations is a common ordeal. Time spent in computation of such complex dynamics is computationally expensive. Many strategies are being pursued to keep the computation time within reasonable but at the end of the day, the size of problems which can be tackled are ultimately limited by the capabilities of the available hardware. Hence such systems can be implemented on Graphics Processing Unit (GPU) which has the parallel computing power for quicker simulation. A GPU has a number of threads where each thread can execution different program. This helps us to achieve significant & faster computation than a normal CPU system. In this paper, the fractional Euler's method is implemented on GPU to achieve faster computation.
The structure of this paper is as follows.- Section II gives information about Fractional Euler's method to solve FODE. The Section III describes General Purpose Graphics Processing Unit (GPGPU) technology and Parallel Computing toolbox and gives information about specifications of the hardware used and Section IV explains the design steps for the implementation. In Section V we presents the results for the FODE and Section VI concludes the work.

II. FRACTIONAL EULER'S METHOD

A numerical method is approximation of a solution to a problem with a specific numerical process with given initial or boundary value conditions. In this work, various time dependent difference equations are solved by using numerical methods. In numerical method, the differential equation is solved under a

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 8, August 2017

continuous interval [a,T] which is divided into steps of size h. The difference equation having successive approximations yn+j is described by Lambert [7]. Here the values of n such as n = 0,1,...,k are used for the calculation of the successive parameter of the given sequence until we reach to the endpoint (T). In the numerical method, whenever the value of k is less than 1, we can describe it as a multistep method. The one-step method like Forward Euler Method can be represnted for k = 1. This paper is concerned with the numerical solution of following initial value problem of FDE.

$$cD_{\alpha,t}^{-\alpha} y(t) = f\big(t, y(t)\big), m-1 < \alpha < m \quad (1)$$

Where $y^{(j)}(0) = y_j^o, j = 0.1, \dots m-1$

Fractional Euler's Method is one of the integration method same as the classical Euler's method that convert the original fractional equation into the fractional integral equation. Then to solve the initial value problem we have to apply some numerical method to that equation [8]. By applying $D_{o,t}^{-\alpha}$ on the both sides of eq. 1, we will get equivalent Volterra integral equation which is given below [9]:

$$y(t) = \sum_{j=0}^{m-1} \frac{t^j}{j!} y_0^j + \frac{1}{r\alpha} \int_0^t (t-s)^{(\alpha-1)} f\big(s, y(s)\big) ds$$

$$= \sum_{j=0}^{m-1} \frac{t^j}{j!} y_0^j + D_{0,t}^{-\alpha} f(t, y(t))$$

Now, we will apply the following Euler's method to solve intitial value problem [8]. The approximation of $[D_{o,t}^{-\alpha} f\big(t, y(t)\big)]$ $t = t_{n+1}$ is done by using left fractional rectangular formula given below [8] :

$$[D_{o,t}^{-\alpha} f(t)]_{t=t_n} \approx \sum_{k=0}^{n-1} b_{n-k-1} f(tk) \quad (3)$$

Where , $b_k = \frac{\Delta t^\alpha}{r(\alpha+1)} [(k+1)^\alpha - (k)^{(\alpha)}$

$$y_{n+1} = \sum_{j=o}^{m-1} \frac{t_j^{n+1}}{j!} y_0^{(j)} + \Delta t^a \sum_{j=0}^{n} b_{j,n=1} f(tj, yj) \quad (4)$$

Where, $b_{k,n+1} = \frac{1}{r(\alpha+1)} [(n-k+1)^{(\alpha)} - (n-k)^{(\alpha)}]$

III. GRAPHICS PROCESSING UNIT (GPU)

The graphics processing unit (GPU) has become most important part of todays computing techniques. Over the past six years, due to the development in the performance and capabilities of GPUs, it has become one of the powerful graphics engine. As GPU is a highly parallel programmable processor, can do faster calculation of computationally intensive problems than normal CPUs. GPU computing is the utilization of the GPU as a general purpose unit for solving a given problem. It is also known as General-Purpose computing on Graphics Processing Units (GPGPU) [14] [15]. The main objective of GPU computing is to achieve the highest performance over the CPU for a given problem by using data parallel algorithm. The CPU is latency optimized (minimal time to complete a task), whereas the GPU is throughput optimized (Maximum tasks per unit of time, also Bandwidth). In the GPU layout, where instructions are carried out in parallel, the largest amount of surface area is spent on arithmetic units (ALUs). This is the reason a GPU has a high computational throughput at the cost of a higher latency. In terms of Flynns taxonomy [10], a CPU is a Single Instruction Single Data (SISD) type processor, whereas a GPU is would be a Single Instruction Multiple Data (SIMD) processor. The term Single Instruction Multiple Thread (SIMT) is also used. Now, the GPU is a highly powerful graphic engine, which is used as a general purpose computing processor for performing parallel computing and big data processing. Now, GPU is preferred over the CPU due to limitations of CPU over speed. Due to the high computation power and advance programmability, GPU has become one of the promising competitor in high performance computing. There are few toolboxes of MATLAB used for GPU computing. By means of these available toolboxes, MATLAB code can be easily executed on GPU with slight conversions in the existing code and least knowledge of GPU. MATLAB Parallel Computing Toolbox, Jacket and GPUmat are the commonly used toolboxes for parallel computing. [11].

A. GPU acceleration

The CPU decides which is the 'computationally intensive' part of the program, that is the part which takes large amount of calculation time and also whose data set can be processed in parallel. As shown in Fig.1, this part is then transferred to the GPU for accelerated parallel processing, while the rest of the program is executed sequentially by the CPU. The output data sets of the part calculated by the GPU is collected and transferred back to the CPU memory for either display or some other small calculation.

The GPU contains hundreds or even thousand of discrete processing units (cores), which make it extremely efficient and speedy to handle compute

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 8, August 2017

intensive code [14]. Therefore, GPU computing has huge potential in areas requiring processing of enormous amounts of data, also called Big Data [15].

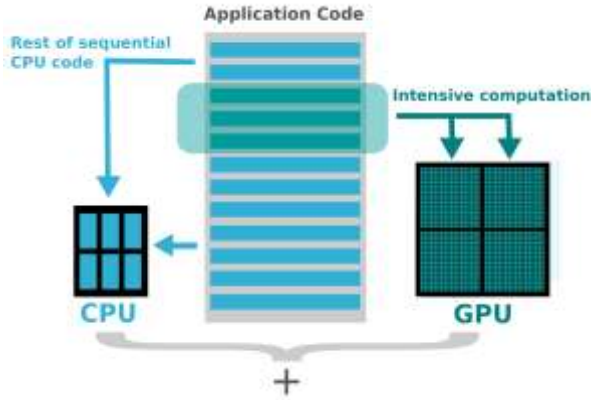

Fig. 1. Acceleration by GPU [14]

B. Parallel Computing Toolbox (PCT)

Parallel Computing Toolbox (PCT) allows us to solve computationally heavy and big data problems using multicore processors, GPUs and computer clusters. Using PCT, we can developed parallel for-loops, special array types and parallelized numerical algorithms which allows us to parallelize MATLAB applications without CUDA or MPI programming. Also the simulations of multiple models can be performed in parallel by using the toolbox with Simulink [12]. Few important MATLAB functions which are used in GPU Computing using MATLAB are as follows [12] [13]:

gpuArray():
It is used to build an array on GPU.
For Example: A = gpuArray(B)
gather():
It is used to transfer distributed array or gpuArray to
MATLAB local workspace.
For Example:X = gather(A)
arrayfun():
It apply function to each element of array on GPU.
For Example:A = arrayfun(FUN, B)

tic-toc command
tic and toc functions of MATLAB work together to measure elapsed time. tic, by itself, saves the current time that toc uses later to measure the time elapsed between the two.

C. HARDWARE SPECIFICATIONS

We have evaluated the Fractional Euler's method for fractional order ordinary differential equations on GPU and compared its performance with CPU. The specifications of GPU are listed in Table I and CPU are listed in the Table II.

TABLE I
GPU SPECIFICATIONS

Model	NVIDIA Geforce 940M
Total Graphics Memory	2048 MB
No. of cores	384
Clock Rate	1071 MHz

TABLE II
CPU SPECIFICATIONS

Model	Intel i5-5200U
Cache	3 MB
No. of cores	2
Clock Rate	2.2 GHz

IV. DESIGN STEPS FOR IMPLEMENTATION

Design steps for successful implementation of Fractional Euler's method to solve FODE on GPU are described below.
It contains basic 4 major steps:

- The first phase includes choosing an appropriate mathematical function to implement in the MATLAB. Its really crucial that the functions that are picked are analytically clear with the user itself. Knowing the range of outputs plays an important part in rectifying the program as suitable.
- The second phase includes writing the scalar codes for the functions. The addition, subtraction, multiplication or division of a matrix by a number is called as the scalar operation. These scalar operations create a new matrix having same number of rows and columns with each element of the original matrix added to, subtracted from, multiplied by or divided by the

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 8, August 2017

number. These simple methods are used to implement our functions.

- The third phase includes the vectorization of these functions. Vectorization of MATLAB code converts an entire array into "single "command.
- The condensed code to apply an action to every element of an array is transformed to a single line of code by using a "Vector "operation in MATLAB. In this phase, we will vectorize our given functions, that is eliminating the loops that are present in the program without changing the final output.
- Finally the vectorized code is transfered on GPU. The analysis of the code is done by comparing of time of execution when implemented on both GPU and CPU. The Speed up is calculated by using following formula:

$$\textbf{speed up} = \frac{Program\ Execution\ Time\ on\ CPU}{Program\ Execution\ Time\ on\ GPU}$$

V. RESULTS

The Fractional Euler's method is tested for following FODE on CPU and GPU and their performance comparison is done. Consider the following fractional differential equation:

$$cD^{\alpha}_{o,t}y(t) = -y(t) + \frac{t^{(4-\alpha)}}{r(5-a)}, 0 < \alpha < 1 y(0) = 0 t > 0 \quad (5)$$

The eq. 5 has following exact solution

$$y(t) = t^4 E_{\alpha,5}(-t^{\alpha}), \quad (6)$$

Where, $E_{\alpha,\beta}(z) = \sum_{k=0}^{\alpha} \frac{z^k}{r(\alpha k+\beta)}$ is the two-parameter Mittag-Leffler function.

The parameters used for the solution of above equation are
as follows:

α=1

$stepsize = h$

t = 0 : h : 1 Like 0.1, 0.001, etc.

The analytical solution and approximate solution for the above example is given in Table III. Fig. 2 shows the superposition of graphs of exact solution given by eq.6 (shown in blue) with that of the numerical solution of the FODE solved by Forward Euler's Method for h=0.00001 (shown in red for GPU implementation). As the step size of eq.5 becomes 0.00001, approximate solution becomes closer to the exact solution with the speedup of 2.02552 as shown in Table III and IV. If we further the decrease the step size, GPU will run faster than CPU.

TABLE III
EXACT AND APPROXIMATE SOLUTION FOR EQUATION 5

t	Exact Solution	Approximate Solution		
		h=0.1	h=0.0001	h=0.00001
0	0	0	0	0
0.1	2.4755e-06	0	2.47124e-06	2.4751e-06
0.2	3.8487e-05	5.08596e-06	3.84553e-05	3.84841e-05
0.3	0.00019	7.20445e-05	1.91387e-04	0.000191
0.4	0.000597	0.00031	5.97418e-04	0.00059
0.5	0.00144	0.00089	0.00144	0.00144
0.6	0.00297	0.00203	0.00297	0.00297
0.7	0.005467	0.00400	0.00546	0.00546
0.8	0.00926	0.00710	0.00926	0.00926
0.9	0.01476	0.01172	0.01476	0.01476
1	0.0224	0.01827	0.02239	0.0224

TABLE IV
GPU AND CPU PERFORMANCE FOR FORWARD EULER METHOD

Step Size	Time(sec)		Speedup
	CPU Implementation	GPU Implementation	CPU/GPU
0.1	0.0071	0.0259	0.274131274
0.0001	10.90	15.1251	0.720729119
0.00001	972.9794	480.3588	2.0255263336

VI. CONCLUSION

GPU offers significant computational power for programmers to achieve a desired result. This computational of GPU is particularly effective for utilization of accelerating scientific prototype which has FODE. In this paper, both CPU and GPU implementation of fractional Euler's methods for FODE is presented. Among which GPU implementation has proved to be considerably faster than the serial form run on central processing unit as GPU architecture has huge potential to accelerate parallelizable portions of an algorithm. In this paper, we have used a simple fractional Euler's method for solving fractional differential equations having Caputo derivative, and from the result, it is observed that the

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 8, August 2017

numerical solution is closer to analytical solution as the number of division increase i.e. step size decreases. From the result, we can conclud that Euler's methods can be designed to run on the CPU when the step size is small and on the GPU when the step size is large. In the fractional Euler's method, GPU computation gives good speed up with accurate result as we go on reducing the step size (h).

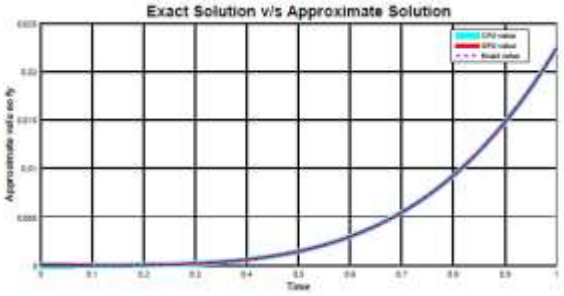

Fig. 2. Forward Euler Method for h=0.00001

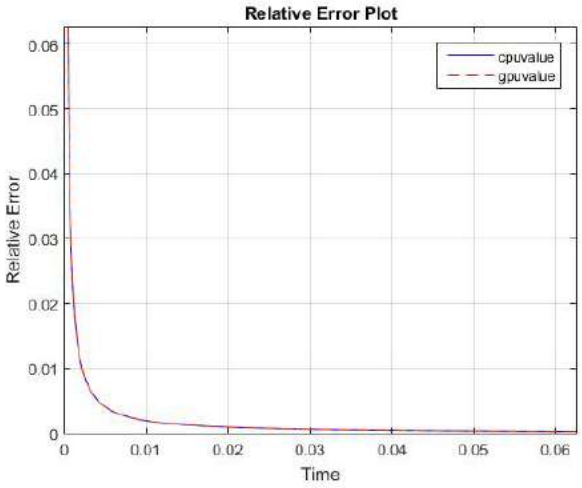

Fig. 3. Forward Euler Method for h=0.00001

REFERENCES

[1] Diethelm, Kai."The analysis of fractional differential equations: An application-oriented exposition using differential operators of Caputo type."Springer, Verlag Berlin Heidelberg, 2010.

[2] Machado, J. Tenreiro, Virginia Kiryakova, and Francesco Mainardi. "Recent history of fractional calculus."Communications in Nonlinear Science and Numerical Simulation Elsevier 16, no. 3 (2011): 1140-1153.

[3] Weilbeer, Marc."Efficient numerical methods for fractional differential equations and their analytical background."Papierflieger, 2005.

[4] ur Rehman, Mujeeb, and Rahmat Ali Khan. "The Legendre wavelet method for solving fractional differential equations."Communications in Nonlinear Science and Numerical Simulation Elsevier 16, no. 11 (2011): 4163-4173.

[5] Suh, Jung W., and Youngmin Kim."Accelerating MATLAB with GPU computing: A primer with examples."Newnes, 2013.

[6] Higham, Desmond J., Xuerong Mao, and Andrew M. Stuart. "Strong convergence of Euler-type methods for nonlinear stochastic differential equations."SIAM Journal on Numerical Analysis 40, no. 3 (2002): 1041-1063.

[7] Lambert, John Denholm. "Numerical methods for ordinary differential systems: the initial value problem."John Wiley & Sons, Inc.New York, NY, USA, 1991.

[8] Li, Changpin, and Fanhai Zeng. "Numerical methods for fractional calculus."Vol. 24. CRC Press, 2015.

[9] Lubich, Ch. "A stability analysis of convolution quadraturea for Abel- Volterra integral equations."IMA journal of numerical analysis 6, no. 1 (1986): 87-101. Oxford University Press

[10] Flynn, Michael J. "Some computer organizations and their effectiveness." IEEE transactions on computers 100, no. 9 : 948-960, IEEE Computer Society Washington, DC, USA, 1972.

[11] Baida Zhang, Shuai Xu, Feng Zhang, Yuan Bi and Linqi Huang, "Accelerating MatLab code using GPU: A review of tools and strategies", Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2nd International Conference on, pp. 1875-1878, IEEE, 2011.

[12] Reese, Jill, and Sarah Zaranek. "Gpu programming in matlab." MathWorks News & Notes. Natick, MA: The MathWorks Inc (2012): 22-5.

[13] Altman, Yair M."Accelerating MATLAB Performance: 1001 tips to speed up MATLAB programs."CRC Press, 2014.

[14] Lippert, Anthony. "Nvidia gpu architecture for general purpose computing. "NVIDIA presentation, April (2009).

[15] Luebke, David, and M. Harris. "General-purpose computation on graphics hardware."In Workshop, SIGGRAPH. 2004.