ISSN (Online) 2394-6849

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

Parallel Computing of Fractional Integral Operators

[1] Sameer S. Chikane, [2] Mukesh D. Patil, [3] Vishwesh A. Vyawahare
[1][3] Department of Electronics Engineering, [2] Department of Electronics and Telecommunication Engineering
[1][2][3] Ramrao Adik Institute of Technology, Nerul, Navi-Mumbai

***Abstract* - Fractional calculus a field dealing with mathematical analysis has its applications in various domains such as power transmission units, image processing, financial system design, automobiles and various control system. There are many advantages of fractional calculus in analytical world. But, the computational cost accompanied with it has prevented software implementations to achieve real-time performance for large and complex computations. This paper exhibits the parallel computing power of the Graphics Processing Unit (GPU) in the area of fractional-order integration. Numerical methods for implementing different fractional-order derivatives and integrations are available. By using MATLAB Parallel Computing Toolbox, GPU computational power can be easily accessed with minimum knowledge of GPU architecture and MATLAB code can be executed on the GPU. The fractional-order integration by Trapezoidal formula using NVIDIA GPU with support of MATLAB Parallel Computing Toolbox is implemented in order to achieve faster execution. Performance comparison of the algorithm for sequential implementation on CPU and parallel implementation on GPU is carried out. This new algorithm produces significant speedup in the computations of fractional-order integration and provide required result in much less time as compared to execution on CPU.**

Index Terms— fractional calculus, fractional derivatives, fractional integrals, Graphics Processing Unit.

INTRODUCTION

On commencement of learning calculus, one starts with defining the differentiation and integration of a function followed by calculating the derivatives and integrals. Subsequently one extends the ideas to higher derivatives and multiple integrals. In addition, one questions how to define and compute differentiation and integration in higher dimensions. Moreover, in calculus one is limited to the concept of taking derivatives and integrals of integer order. It is trivial, at this point when studying mathematics, to ruminate generalizing derivatives and integrals to a greater extent. Nevertheless one may ponder that 'Can we generalize the concept of differentation and integration to a more comprehensive idea than what has been elaborated in calculus ?'[1] Rather than defining derivatives and integrals to merely integer order, can one specify derivatives and integrals to arbitrary order and yet be consistent with the traditional integer order derivatives and integrals that we have been habituate to see in calculus? This is a very interesting question that many people have come across with over time: Can we stretch the idea to make it more generalised and interlock it with what has been formulated till this point? The origin of fractional calculus is dated back to the Lebnitz's letter to L'Hopital in the year 1695, where the notation for derivative of non integer order $^1/_2$ is deliberated. In addition to it Lebnitz writes : "Thus it follows that $d^{1/2}$ will be equal to $\sqrt[x]{dx:x}$. This an apparent paradox from which, one day useful consequences will be drawn."[2]

The study of fractional calculus has blossomed in the past two decades, and a lot of advancement has been accomplished in the theory and analysis of fractional-order systems. The generalization of fractional derivatives and integrals is not simply a mathematical curiosity, but it has pioneered applications in diverse fields of physical sciences, also it caters to various engineering problems. Fractional-order derivatives and integrals are necessary for solving most of the differential, integral and difference equations. The computations of these fractional order derivatives and integrals are nowadays widely governed in academia and research centers but they demand huge amount of computer time. Recent drastic progress in hardware and software computing have promoted the computation of these mathematical equations. Parallel computing has been noteworthy among these advancements [3]. Large-scale engineering problems can be simulated extensively with the help of parallel computers. The advancement in hardware such as multicore processors have further boosted the performance speed of each compute node in the network. This has facilitated multiplicative increases in the computing speed without the necessity to make similar gains in the individual chip speed. Other paradigm in scientific computing that is emerging is the use of multi-threaded Graphics Processing Units (GPUs), which act as co-processors for Central Processing Units (CPUs). Recently, driven by the necessity for fast graphics and games, GPU's have become quite powerful, as well becoming notably cheaper than CPU's of same computing power. GPUs with the capacity to conduct one teraflop i.e. one trillion floating point operations per second have been developed in the past few years. Simultaneously

ISSN (Online) 2394-6849

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

MATLAB (matrix laboratory) is multi-paradigm numerical computing environment developed by MathWorks, allows implementation of algorithms on GPUs easier [4].

The benefit of using GPU for general purpose computation is the execution speedup that can be obtained due to the parallel architecture of GPU [5]. One of the most promising General Purpose GPU (GPGPU) technologies called CUDA SDK, which is developed by NVIDIA. But it is not viable to expect the average programmer to cope with all the complexity applicable in CUDA programming. MATLAB GPU toolbox is available for programming now, and we can grasp the rapid prototyping convenience of MATLAB for GPU computing of programs. With the help of MATLAB GPU toolbox [6], GPU computational power can be easily acheived from MATLAB with minimal GPU knowledge and MATLAB programs can be executed on the GPU [7].

The structure of this paper is as follows.- Section II gives information about GPU computing using MATLAB. The Section III describes the overview of GPU toolboxes for MATLAB. Section IV states the algorithm for fractional Trapezoidal formula. This is followed by Section V which gives information about specifications of the hardware used and the design methodology. In Section VI we presents the results for the Fractional Trapezoidal formula for interation and Section VII concludes the work.

II. GPU COMPUTING USING MATLAB

Multicore machines and hyper-threading technology have enabled scientists, engineers, and financial analysts to speed up computationally intensive applications in a variety of disciplines. Today, another type of hardware promises even higher computational performance: the Graphics Processing Unit (GPU). Although GPUs have been traditionally used only for computer graphics, a recent technique called General purpose computing on graphics processing units (GPGPU) allows the GPUs to perform numerical computations usually handled by CPU. The advantage of using GPU for general purpose computation is the performance speedup that can be achieved due to the parallel architecture of GPU [8]. Unlike a traditional CPU, which includes no more than a handful of cores, a GPU has a massively parallel array of integer and floating-point processors, as well as dedicated, high-speed memory.

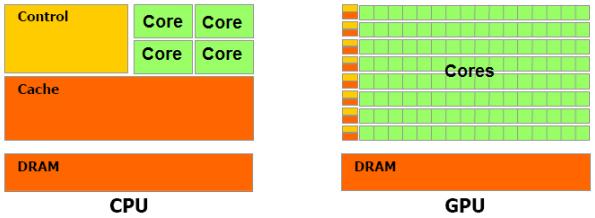

Figure 1. Architecture difference of CPU and GPU [9]

One of the most promising GPGPU technologies is called CUDA SDK, developed by NVIDIA. But it is not realistic to expect the average programmer to deal with all this complexity introduced by CUDA programming. Fortunately, some toolboxes of MATLAB for GPU programming are available now, and you can leverage the rapid prototyping benefits of MATLAB for GPU Programming. With the help of MATLAB and those toolboxes, GPU computational power can be easily accessed with minimum GPU knowledge and MATLAB code can executed on the GPU. Existing MATLAB code can be ported and executed on GPUs with few modifications. And now, there are three toolboxes that are extensively in use, i.e. Jacket, GPUmat, and Parallel Computing Toolbox of MATLAB [6].

GPU is a good choice, because of its high parallel architecture. And there are also some common principles for utilizing GPU more efficient in MATLAB code. More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations where the same program is executed on many data elements in parallel. Because the same program is executed for each data element, there is a lower requirement for sophisticated flow control; and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches. The architecture of GPU is showed in Fig.1, in which the most area of the chip is used for ALU, and only little of it is used for control unit and memory. Therefore, GPU is good at parallel computing and MATLAB is good at matrix operation. There are lots of same command for every elements of a matrix. To take the add operation for example, it is add corresponding elements of two matrices. It is very suitable for computing on GPU which is SIMD architecture [8].

ISSN (Online) 2394-6849

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

III. OVERVIEW OF GPU TOOLBOXES FOR MATLAB

The following is an overview of three toolboxes- Jacket, GPUmat and MATLAB Parallel Computing Toolbox.

1) Jacket toolbox

Jacket toolbox was developed by Accelereye [10]. This toolbox requires the code written in CUDA to accelerate calculations on the NVIDIA GPUs. It is very similar to the NVIDIA CUDA plug-in that allows the writing of MATLAB MEX files using CUDA to implement acceleration. The benefit of Jacket is that the existing functions can be used like any other MATLAB function. The variables and the data for a function should be of Jacket defined GPU data type. Thus the calculations are done on the GPU transparently since the variables used are of GPU type. There is currently a limited set of functions that have been fully ported to the GPU and most of these use the exact same function call as the regular CPU variant.

2) GPUMat toolbox

GPUmat toolbox is developed by GPyou Group [11]. It is fully free under GNU license. The GPyou Group offers support on developing GPU-based software on demand, as well as to customize our already existing products in function of the user requests. GPUmat allows standard MATLAB code to run on GPUs. The execution is transparent to the user. When the data is GPU type, the code will automatically execute on GPU if the operation is supported. In their latest version, there are 57 MATLAB Numerical functions are supported. But the mixed operation is not supported. GPUmat also include a low level Application Programming Interface that are not the standard MATLAB function.

3) Parallel Computing Toolbox (PCT)

PCT is the product of Mathworks which by default comes with MATLAB. From version R2010b all versions of MATLAB support GPU computing. We are using MATLAB 2015a in which we have used this toolbox for converting our MATLAB code to run on GPU. How the GPU computing power of the toolbox can be utilized to run the MATLAB code more efficiently on GPU is discussed in next section.

IV. ALGORITHM FOR FRACTIONAL TRAPEZOIDAL FORMULA

If $f(t)$ is approximated on each subinterval $[tk;\ tk+1]$ by following piecewise polynomial with degree of order one [12]

$$f(t)\left|\left[t_k,t_{k+1}\right]=t_k,t_k+1=\frac{t_{k+1}-t}{t_{k+1}-t_k}f(t_k)\right)\frac{t-t_k}{t_{k+1}-t_k}f(t_{k+1})$$

we obtain the fractional trapezoidal formula as follows

$$[D_{0,t}^{-\alpha}f(t)]_{t=t_n}\approx[D_{0,t}^{-\alpha}f(t)]_{t=t_n},=$$

$$\frac{1}{\Gamma(\alpha)}\sum_{k=0}^{n-1}\int_{t_k}^{t_{k+1}}(t_n-t)^{\alpha-1}\times(\frac{t_{k+1}-t}{t_{k+1}-t_k}f(t_k)+$$

$$\frac{t-t_k}{t_{k+1}-t_k}f(t_{k+1})dt,=\sum_{k=0}^{n}a_{k,n}f(t_k)$$

Where

$$a_{k,n}=\frac{1}{\Gamma(\alpha)}\begin{cases}\int_0^{t_1}(t_n-t)^{\alpha-1}\frac{t_1-t}{t_1-t_0}dt & k=0,\\ \int_{t_k}^{t_{k+1}}(t_n-t)^{\alpha-1}\frac{t_{k+1}-t}{t_{k+1}-t_k}dt & \\ \quad+\int_{t_{k+1}}^{t_k}(t_n-t)^{\alpha-1}\frac{t-t_{k-1}}{t_k-t_{k-1}}dt & 1\le k\le n-1,\\ \int_{t_{n-1}}^{t_n}(t_n-t)^{\alpha-1}\frac{t-t_{k-1}}{t_n-t_{n-1}}dt & k=n.\end{cases}$$

This can be written in simple form as follows

$$a_{k,n}=\frac{\Delta t^{\alpha}}{\Gamma(\alpha+2)}\begin{cases}(n-1)^{\alpha+1}-(n-1-\alpha)n^{\alpha} & k=0,\\ (n-k+1)^{\alpha+1}+(n-1-k)^{\alpha+1} & \\ \quad-2(n-k)^{\alpha+1} & 1\le k\le n-1,\\ 1 & k=n.\end{cases}$$

The above algorithm is helpful for finding fractional integration of a function by Fractional Trapezoidal Formula.

V. DESIGN METHODOLOGY

The Trapezoidal Formula for integration is evaluated on GPU and compared its performance with CPU at Leopard Cluster IIT Bombay. The specifications of

ISSN (Online) 2394-6849

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

GPU are listed in Table I and CPU are listed in the Table II.

Table I
GPU SPECIFICATIONS

Model	NVIDIA Tesla K40
Total Graphics Memory	12 GB
No. of cores	2880
Clock Rate	745 MHz

Table II
CPU SPECIFICATIONS

Model	Intel(R) Xenon(R) E5-2620
RAM	32 GB
No. of cores	24
Clock Rate	2.0 GHz

For the GPU implementation of numerical methods for fractional-order integration using MATLAB, the following

steps were followed:
Step 1: Writing the sequential codes for implementing the algorithm on CPU.
eg.:Write a sequential code in MATLAB to square every element of the vector.
a = [1 2 3 4];
for i =1:4
a(i) = a(i)^2;
end

Step 2:Writing vectorized code for implementation of algorithm on CPU.
eg.: Vectorized code for above example:
a = [1 2 3 4];
a = a:^2;
This code will work on all the four elements at the same time. Thus, as a result the vectorized codes are more faster and less time consuming than sequential codes.

Step 3: Finally the vectorized code is transfered on GPU.
eg.: GPU code for above example:
a = gpuArray([1 2 3 4]);
a = a:^2;

Step 4: Analyze the code by comparing of time of execution and speedup.

$$\text{Speed Up} = \frac{Program\ Execution\ Time\ on\ CPU}{Program\ Execution\ Time\ on\ GPU}$$

Profiler: To find the part of code which takes maximum time, we use the Run and Time feature of Matlab. This opens up a Profiler which gives us the analysis of the time taken by each line of the code. The part of the code which takes maximum time can be vectorized for speedup.

Profile Summary Report: The Profile Summary report presents the overall execution time taken by the function and provides summary statistics for each function called. The following Fig.2 shows the Profile Summary report for the Fractional Trapezoidal integration.

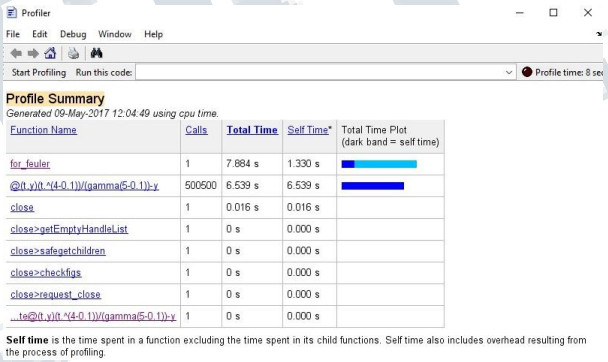

Figure 2. Profile Summary Report

Profile Detail Report: The Profile Detail report shows profiling results for a function that MATLAB called while profiling. The Fig. 3 shows the part which takes maximum time for the execution.

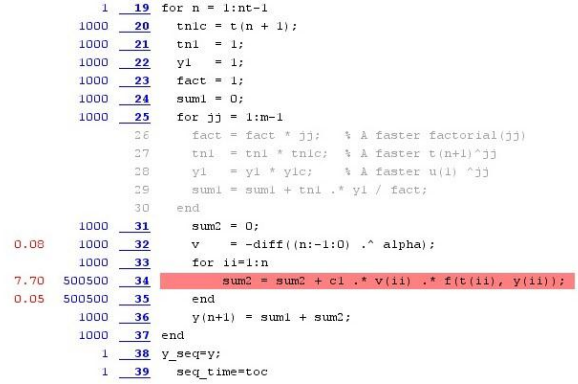

Figure 3. Profile Detail Report

ISSN (Online) 2394-6849

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

The following functions are considered for fractional order integration and the results are elaborated in the next section. The functions used are

$$y = sin(t), \quad (1)$$

$$y = e^{t} sin^{3}(t), \quad (2)$$

$$y = t^{3} + 3tsin(t) - e^{-t}\cos(t), \quad (3)$$

$$y = e^{-4t} sin^{2} t - e^{-t}\cos(t), \quad (4)$$

$$y = \log(e^{6t}) + \sin(2^{t}) + te^{2t}, \quad (5)$$

$$y = sin\sqrt{t^{3} + 3t}, \quad (6)$$

$$y = e^{cos}\left(\sqrt[t]{t}\right).\sin\left(3t(t^{2} + t^{3})\right), \quad (7)$$

VI. RESULTS FOR FRACTIONAL ORDER INTEGRATION BY TRAPEZOIDAL FORMULA

The parameters considered for the integration by Trapezoidal Formula for the eq. 1-7 are as follows:
α = 0:5
step size = h (0.0001, 0.00001, 0.000001)
t = 0 : h : 32π

A. y = sin (t)

The CPU and GPU performance comparison of fractionalorder integration of *y = sin (t)* by Trapezoidal formula is given in table III

Table III
GPU AND CPU PERFORMANCE FOR TRAPEZOIDAL FORMULA ON EQ.1

Step Size	Time (sec)		Speed Up
h	Vectorized	GPU	Vec/GPU
0.0001	0.1640	0.0360	4.5555
0.00001	1.2600	0.2133	5.9071
0.000001	10.3935	1.8363	5.9573

The speedup of around 5.9071 to 5.9573 are obtained.

B. $y = e^{t} sin^{3}(t)$,
The CPU and GPU performance comparison of fractionalorder integration of $y = e^{t} sin^{3}(t)$,by Trapezoidal formula is given in table IV

Table IV
GPU AND CPU PERFORMANCE FOR TRAPEZOIDAL FORMULA ON EQ. 2

Step Size	Time(sec)		Speedup
h	Vectorized	GPU	Vec/GPU
0.0001	0.1543	0.0361	4.2742
0.00001	1.4061	0.2162	6.5037
0.000001	11.9100	1.8804	6.8337

The speedup of around 6.5037 to 6.8337 are obtained.

C. $y = t^{3} + 3tsin(t) - e^{-t}\cos(t)$,
The CPU and GPU performance comparison of fractionalorder integration of $y = t^{3} + 3tsin(t) - e^{-t}\cos(t)$, by Trapezoidal formula is given in table V

Table V
GPU AND CPU PERFORMANCE FOR TRAPEZOIDAL FORMULA ON EQ. 3

Step Size	Time (sec)		Speed Up
h	Vectorized	GPU	Vec/GPU
0.0001	0.1700	0.0346	4.9132
0.00001	1.4315	0.1975	7.2481
0.000001	13.7167	1.8534	7.4008

The speedup of around 7.2481 to 7.4008 are obtained.

D. $y = e^{-4t} sin^{2} t - e^{-t}\cos(t)$,
The CPU and GPU performance comparison of fractionalorder integration of $y = e^{-4t} sin^{2} t - e^{-t}\cos(t)$, by Trapezoidal formula is given in table VI

Table VI
GPU AND CPU PERFORMANCE FOR TRAPEZOIDAL FORMULA ON EQ. 4

Step Size	Time (sec)		Speed Up
h	Vectorized	GPU	Vec/GPU
0.0001	0.1752	0.0339	5.1681
0.00001	1.4974	0.2145	6.9517
0.000001	12.7360	1.9640	6.9847

The speedup of around 6.9517 to 6.9847 are obtained.

E. $y = \log(e^{6t}) + \sin(2^{t}) + te^{2t}$,
The CPU and GPU performance comparison of fractionalorder integration of $y = \log(e^{6t}) + \sin(2^{t}) + te^{2t}$, by

ISSN (Online) 2394-6849

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

Trapezoidal formula is given in table VII

Table VII
GPU AND CPU PERFORMANCE FOR TRAPEZOIDAL FORMULA ON EQ. 5

Step Size	Time (sec)		Speed Up
h	Vectorized	GPU	Vec/GPU
0.0001	0.1913	0.0332	5.7620
0.00001	1.6067	0.2209	7.2734
0.000001	15.7230	2.0007	7.8587

The speedup of around 7.2734 to 7.8587 are obtained.

F. $\mathcal{Y} = sin\sqrt{t^3 + 3t},$

The CPU and GPU performance comparison of fractionalorder integration of $\mathcal{Y} = sin\sqrt{t^3 + 3t},$ by Trapezoidal formula is given in table VIII

Table VIII
GPU AND CPU PERFORMANCE FOR TRAPEZOIDAL FORMULA ON EQ. 6

Step Size	Time (sec)		Speed Up
h	Vectorized	GPU	Vec/GPU
0.0001	0.1744	0.0357	4.8851
0.00001	1.4529	0.2211	6.5712
0.000001	12.2545	1.6670	7.3512

The speedup of around 6.5712 to 7.3512 are obtained.

G. $\mathcal{Y} = e^{cos}\left(\sqrt[t]{t}\right).\sin\left(3t(t^2 + t^3)\right),$

The CPU and GPU performance comparison of fractional order integration of $\mathcal{Y} = e^{cos}\left(\sqrt[t]{t}\right).\sin\left(3t(t^2 + t^3)\right),$by Trapezoidal formula is given in table IX

Table IX
GPU AND CPU PERFORMANCE FOR TRAPEZOIDAL FORMULA ON EQ. 7

Step Size	Time (sec)		Speed Up
h	Vectorized	GPU	Vec/GPU
0.0001	0.2026	0.0346	5.8554
0.00001	1.6238	0.2118	7.6666
0.000001	14.1317	1.9973	7.8754

The speedup of around 7.6666 to 7.8754 are obtained.

VII. CONCLUSION

GPU is a hardware which is very much befitting for parallel applications. The fractional-order derivatives and integrals demand a huge execution time thus they are an excellent nominee to be executed on GPU. The execution or processing time of any function can be reduced by vectorizing it and running it on the GPU cores. Overall performance of the system improved by vectorization of code and reducing CPU load by using GPU. Parallel Computing Toolbox of MATLAB is very useful for modifying your code to run on GPU. From the results, we can conclude that for Trapezoidal integration method as the number of computation increases i.e. number of iteration increases, the GPU speeds up the execution. As step size(h) decreases, the number of computations increases then significant speedup is obtained.

REFERENCES

[1] Keith Oldham and Jerome Spanier. The fractional calculus theory and applications of differentiation and integration to arbitrary order, volume 111. Elsevier, 1974.

[2] Igor Podlubny. Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications, volume 198. Academic press, 1998.

[3] Cristobal A Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. Communications in Computational Physics, 15(02):285– 329, 2014.

[4] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for gpus: stream computing on graphics hardware. In ACM Transactions on Graphics (TOG), volume 23, pages 777–786. ACM, 2004.

[5] Wei Zhang and Xing Cai. Efficient implementations of the adams-bashforth-moulton method for solving fractional differential equations. Proceedings of FDA12, 2012.

[6] Parallel computation toolbox. URL https://in.mathworks. com/help/distcomp/.

[7] Karsten Ahnert, Denis Demidov, and Mario Mulansky. Solving ordinary differential equations on gpus. In Numerical Computations with GPUs, pages 125–157. Springer, 2014.

[8] Baida Zhang, Shuai Xu, Feng Zhang, Yuan Bi, and Linqi Huang. Accelerating matlab code using gpu: A review of tools and strategies. In Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on, pages 1875–1878. IEEE, 2011.

[9] CPU versus GPU architecture. accessed 15 July, 2017. URL http://854320174.r.lightningbase-cdn.com/ wp-content/uploads/2013/03/gputech f2.png.

[10] Jacket - the gpu acceleration engine for matlab, 2017. URL http://www.omatrix.com/jacket.html.

[11] Peter Messmer, Paul J Mullowney, and Brian E Granger. Gpulib: Gpu computing in high-level languages. Computing in Science & Engineering, 10(5):70–73, 2008.

[12] Changpin Li and Fanhai Zeng. Numerical methods for fractional calculus, volume 24. CRC Press, 2015.