Analysis of GPU Computation of Parabolic, Bessel, Wright and Riemann Zeta Functions

Ashish A. Jadhav[1], Abhijeet D. Kalamkar[2], Pritish A.Gaikwad[3], Vishwesh Vyawahare[4] and Navin Singhaniya[5]
Department of Electronics Engineering,
Ramrao Adik Institute of Technology,
Nerul, Navi Mumbai.
Email : 23jadhavashish@gmail.com[1], abhijeetkalamkar03@gmail.com[2], pritish.gaikwad05@gmail.com[3], vishwesh.vyawahare@rait.ac.in[4] and navin.singhaniya@rait.ac.in[5]

Abstract. This paper deals with GPU computing of special mathematical functions that are used in Fractional Calculus. The graphics processing unit (GPU) has grown to be an integral part of nowadays's mainstream computing structures. The special mathematical functions are an integral part of Fractional Calculus. This paper deals with a novel parallel approach for computing special mathematical functions used in Fractional Calculus. NVIDIA's GPU hardware is used to speed up the parallel algorithm. A comparison of the sequential code, vectorized code and GPU code is performed. We have successfully reduced the computation time of special mathematical functions using the parallel computing capabilities of GPU.

Keywords: GPU computing, Parallel computing, Fractional Calculus, Parabolic function, Bessel function, Wright function, Riemann Zeta function.

1. Introduction

Fractional calculus (FC) is widely used in the field of engineering because of its ability to model real-world systems with more precision [1]. In recent years the use of this technology has grown exponentially these days as a growing community of researchers has come together to revitalize the field with the help of advanced computing technology [3]. In most cases, this leads to certain differential equations whose solutions are given by Special Mathematical Function. These are non-elementary functions that are found to be useful in some applications and satisfy some special properties. Computation of such functions has proven to be a resource and time intensive task. To tackle this problem, coprocessors such as GPU and parallel algorithms have been used by researchers in the past [4]. The general purpose use of GPUs for computing is commonly known as GPGPU (General Purpose Computing on Graphics Processing Unit) [3]. This is achieved by Parallel Computing. In Parallel Computing, a large number of computing nodes or CPU cores are used to carry out logical and arithmetic operations from an algorithm in parallel fashion. This increases efficiency and reduces compute time.

It has been found that, for the special functions of mathematics are also very useful for finding solutions to the initial-and / or regional issues governed by partial differential equations, and fractional partial differential equations. These special functions are having wide application in different areas of mathematics. Some commonly used special functions are Wright function, Bessel function, Riemann Zeta function, Parabolic Cylinder function, etc.

In this paper we have done computation of different special functions using hardware Section 2 deals with the Introduction GPU. Section 3 will go through the Definition of Special Mathematical Function which we have done the analysis. In section 4 deals with the Implementation and the design part. In section 5 we have taken the results of the functions. And lastly in Section 6 we have concluded.

2. Graphics Processing Unit (GPU)

The Graphics Processing Unit (GPU) is a many-core device used for parallel computing applications. Initially intended only for graphics rendering purposes, now have found its way into scientific computing as well. The CPU has been designed for low latency to be able to switch between operations quickly. In the past, CPU designers opted towards making the multicore design for overcoming the ludicrously high power requirements that came with increasing clock speed [7]. Because graphics rendering is a simultaneous task, GPU designers chose the many-core technique early on. GPUs are designed for high throughput, allowing users to perform as many operations as possible. There is an ample amount of infrastructure on the chip to get low latency on the CPU, such as caches to ensure that a vast amount of data is readily accessible. There is a significant amount of control flow silicon. Counter to that, the balance has shifted in the GPU, where more arithmetic and logic units are needed. It is designed for compute-intensive, highly parallel computations and hence is specialized for them.
The GPUs use stream processing architecture that is well suited for compute-intensive parallel tasks [6]. Coupled with the current spike in interest in similar research endeavours, collectively known as GPGPU computing, GPUs have become a viable and efficient tool for implementing compute-intensive algorithms [8].

MATLAB is an abbreviation for Matrix Laboratory. It is one of the most widely used proprietary software and programming language by scientists and engineers for numerical computation, due to its excellent graphical capabilities, it has become a well-known tool for data analysis. Researchers also have been known to be using MATLAB scripts and toolboxes to construct immensely complicated systems [7]. The most crucial part of MATLAB is its Toolboxes. MATLAB Toolboxes are software packages/libraries tailored for a specific domain or application. One such toolbox was used in this study, The Parallel Computing Toolbox (PCT), which is the backbone of parallel computing in MATLAB [6]. We will be using the GPU enabled functions from PCT for a single GPU system, a detailed explanation of GPU focused part of PCT is in [3]. At this time, PCT is only available for NVIDIA CUDA enabled GPUs only.

This following table is GPU device specifications used during computing the function.

GPU Specification :

PARAMETER	SPECIFICATION
Speed	536@1.46-1.59 GHz 192 Bit @12000 MHz
Architecture	Turing
Core Speed	1455-1590(Boost) MHz
Memory Speed	12000 MHz
Memory Type	GDDRG6
Max.Amount of Memory	6144 MB
DirectX	DirectX 12.1
Transistor Count	6600 Millon
Cuda Cores	1536

Table 1: NVIDIA GTX 1660Ti GPU Specifications

CPU Specification :

PARAMETER	SPECIFICATION
Processor Name	Intel@CoreTMi7-9750H Processor
Lithography	14nm
No. of Cores	6
No.of Threads	12
Processor Based Frequency	2.6 GHz
Maz Turbo Frequency	4.50 GHz
Cache	12 MB

Table 2: CPU Intel@CoreTMi7-9750H Processor Specifications.

3. Special Mathematical Functions Used in Fractional Calculus

Special functions implemented in this paper are Wright function, Bessel function, Riemann Zeta function, Parabolic Cylinder function. The Wright function plays an important part in the theory of the partial differential equations of the fractional order that are used for modeling of most phenomena including. The anomalous diffusion processes or in the theory of the complex systems. Bessel functions are used to solve in 3D wave equation at a given frequency. The solution is generally a sum of spherical bessels functions that gives the acoustic pressure at a given location of the 3D space. The Riemann zeta function used in analytic number theory and has applications in applied statistics, physics, probability theory. We have used numerical method of this functions and coded that using MATLAB.

3.1. Parabolic Function:

The parabolic cylinder functions (PCFs), also known as Weber parabolic cylinder functions. It is a class of functions sometimes called Weber functions.

$$y_1(x) = e^{-x^2/4} F_1\left(\frac{1}{2}a + \frac{1}{4}; \frac{1}{2}; \frac{1}{2}x^2\right) \quad (1)$$

$$y_2(x) = xe^{-x^2/4} F_1\left(\frac{1}{2}a\frac{3}{2}; \frac{3}{2}; \frac{1}{2}x^2\right) \quad (2)$$

The above two equations 1 and 2 are the even and odd equation respectively. Where, 1F1(a:b:z) is a confluent hypergeometric function of the first kind. We have checked the results for the values of N = 200, and parameter A = 1, 4, 6, and having step size *-10 : 1e-3 : 10, -10 : 1e-4 : 10, -10 : 1e-5 : 10.*

3.2. Bessel Function:

Bessel's function was first formulated by mathematician Daniel Bernoulli. Bessel's function is a solution of differential equation which is known as Bessel equation

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - n^2)y = 0 \quad (3)$$

For an arbitrary complicated range α, the order of the bessel function. Although α and −α produce the equal differential equation, it is conventional to define different bessel capabilities for those two values in this sort of manner that the bessel features are by and large easy capabilities of α. Bessel function are mainly

important for most of the problems of wave propogation and in static potentials and in cylindrical co-ordinate systems. It is can be used to define the function by its series expansion around x = 0

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\,\Gamma(m+\alpha+1)}(x/2)^{2m+\alpha} \quad (4)$$

Here, Γ(z) is the gamma function. Using the above equation 4 we have analysed the function and obtained the results. We have checked the results for the values of N=200, parameter Alpha = 0.9, 1.5, 1.9, and have plot against step size *0 : 1e-3 : 3, 0 : 1e-4 : 3, 0 : 1e-5 : 3.*

3.3. Wright Function:

The Wright function is defined by

$$\emptyset = (\alpha, \beta; z) = \sum_{k=0}^{\infty} \frac{z^k}{k!\,\Gamma(\alpha k + \beta)} \quad (5)$$

Where, Γ(z) is the gamma function. It became added for the primary time via E.M. Wright in [8] in connection together with his investigations inside the asymptotic principle of partitions. The Wright function characteristic belongs to the magnificence of the hypergeometric capabilities; it may be reduced to the recognized special function for some values of the parameters ρ and β because of the relation. The Wright function along with the Mittag-Leffler function plays a prominent role in the theory of the partial differential equations of the fractional order. Using the above equation 5 we have analysed the function and and computed results for the values N=200, and have plot against step size *0 : 1e-2 : 3, 0 : 1e-3 : 3, 0 : 1e-4 : 3.*

3.4. Riemann Zeta Function :

Riemann expanded the definition of Euler ξ (s) function in all complex numbers (except for a simple pole in s = 1 and one remainder). Euler's product description for this function is still valid if the actual s is more than one. To help understand the values of some of the more complex numbers, Riemann found the performance efficiency of the Riemann zeta function:

$$\xi\,(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \quad (6)$$

Using the above equation 6 we have analysed the function and results for the values of N=200, and have plot against step size: *1e6, 1e4, 1e5.*

4. Design and Implementation

While changing over MATLAB code to run on the GPU, it is best to begin with a correct and optimized MATLAB code. The code should provide the required results accurately. For CPU programming, vectorization is considered to be a great practice, whereas for GPU, vectorization is essential for achieving high execution. Subsequently it is essential to vectorize the code by replacing looped scalar operations with MATLAB matrix and vector operations. Profiling gives accurate information about the time taken by each line or section of the code. The Run & Time include of MATLAB is an viable tool which profiles the code while executing it. The section of code that the profiler appears taking the maximum execution time on the CPU are the ones that we need to focus on. Writing the GPU code utilizing the functions in MATLAB which are overloaded to run on GPU is very useful. The variables or data required for execution on GPU device is to begin with transferred to device memory using gpuArray() function. The capacities with these variables as arguments are specifically executed on GPU. After the handling on GPU, the results are replicated back utilizing gather() function.

Following algorithm is used for converting the MATLAB code for GPU execution:

1. Start with the code that gives correct results.
2. Make the vectorization code of the function.
3. Run the code using Run & Time Function.
4. Analyze and Convert the section that takes more time to execute on GPU.
5. Transfer the variables used in the section described in step #4 to GPU memory using GPU Array function.
6. After GPU execution, transfer the output variables back to CPU memory using gather function.
7. After receiving the gathered data from GPU, rest of the execution is performed on CPU.

To calculate the speed up use the following equation

$$\text{SPEEDUP TIME} = \frac{\text{CPU EXECUTION TIME}}{\text{GPU EXECUTION TIME}}$$

5. Result and Analysis

First we have written sequential code using the definition of functions. After checking the correctness of the code we have profiled the sequential code using MATLAB profiler. Then we have observed the section of code which consume more time for executions, that section we have vectorized using Parallel Computing toolbox of MATLAB. Output of vectorized code is also benchmark with sequential code for verifying correctness of code. Finally we have converted the vectorized code to GPU code by transferring data to GPU memory using inbuilt

function of MATLAB like gpuArray(). Fig 1, Fig 2, Fig 3 and Fig 4 shows the output benchmarking plots of different functions. From the benchmarking plots it is clear that output obtained from vectorized code is same as the output obtained from GPU code.

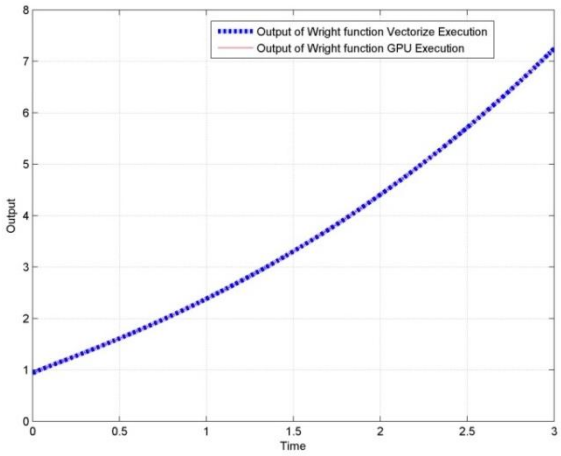

Fig. 1. Output Benchmarking plots of Wright function.

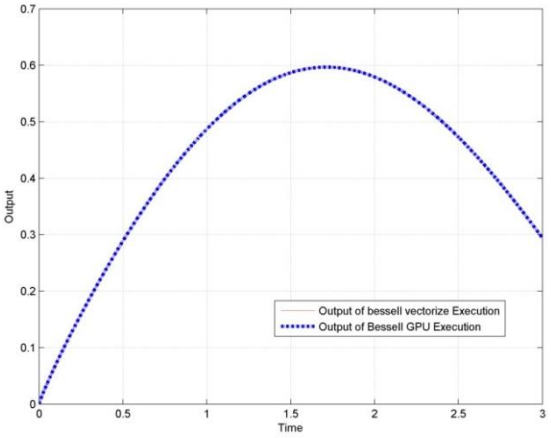

Fig. 2. Output Benchmarking plots of Bessell function.

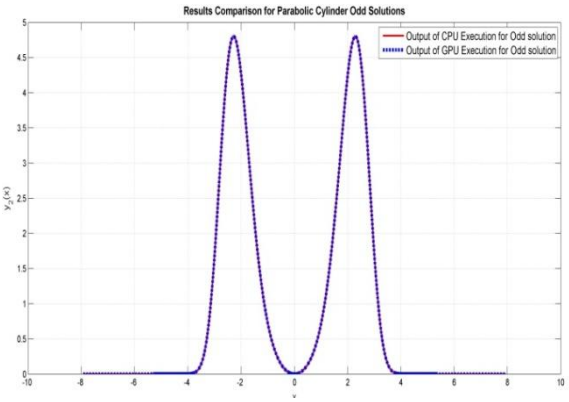

Fig. 3. Output Benchmarking plots of Odd Parabolic Cylinder function.

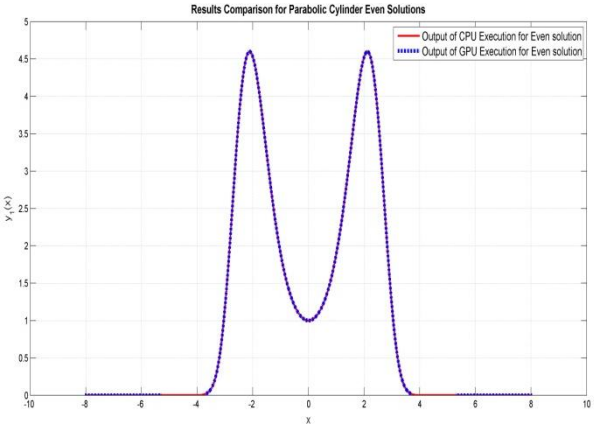

Fig. 4. Output Benchmarking plots of Even Parabolic Cylinder function.

The speed up obtained are in the range of 2x to 180x for systems with NVIDIA GTX 1660Ti GPU. From the results it is observed that speedup depends on the amount of calculation operation like multiplication, division, power, number of iterations of an operation and amount of data. We can further extend this work by writing more optimized MATLAB and using more powerful platforms like CUDA.

In the Results tables T_S=Sequential Time for execution of code, T_V=Vectorize Time for execution of code, T_G= GPU Time for execution of code. We have observed the speed up with respect to T_S=Sequential Time for execution of code verses T_V=Vectorize Time for execution of code (T_S/T_V), T_V=Vectorize Time for execution of code verses T_G= GPU Time for execution of code (TV/TG) and T_S=Sequential Time for execution of code verses T_G= GPU Time for execution of code (T_S/T_G). For the Wright function T_S/T_V speedup is approximately 152x, T_V/T_G speedup is approximately 1.2x, and T_S/T_G speedup is approximately 181x. In the table of Bessel function T_S/T_V speedup is approximately 76x, T_V/T_G speedup is approximately 1.2x, and T_S/T_G speedup is approximately 79x. . In the result table of Riemann Zeta function T_S/T_V speedup is approximately 3x, T_V/T_G speedup is approximately 1.4x, and T_S/T_G speedup is approximately 5x. In the table of Even and Odd Parabolic cylindrical function, T_V/T_G speedup is approximately 1.2x. From the results we have observed that speedup increases as the data increases like for example speedup increases as we change step size from stepsize of 1e-2 to 1e-4. Time taken for Sequential code is more as compare to vectorize code and GPU code, from all the three codes GPU code is fastest and very optimized.

WRIGHT FUNCTION							
Parameters		**Sequential**	**Vectorize**	**GPU Time**	**Speedup**		
ALPHA ,BETA	**Step Size**	**T_S**	**T_V**	**T_G**	**T_S/T_V**	**T_V/T_G**	**T_S/T_G**
0.6	1.00E-02	0.3483	0.004	0.002	79.15	1.56	124.069
0.8	1.00E-02	0.3998	0.003	0.002	120.4	1.16	140.6643
0.9	1.00E-02	0.2734	0.005	0.004	52.57	1.13	59.43
0.6	1.00E-03	3.96	0.019	0.018	208.7	1.00	210.30
0.8	1.00E-03	4.01	0.022	0.020	178.84	1.08	194.73
0.9	1.00E-03	3.594	0.017	0.014	206.06	1.17	242.83
0.6	1.00E-04	27.53	0.164	0.135	167.77	1.21	203.81
0.8	1.00E-04	36.2	0.175	0.140	206.92	1.25	258.72
0.9	1.00E-04	26.8	0.18	0.135	149.2	1.33	199.03
		Total Average Speed			**1.52E+02**	**1.214**	**181.51**

Table 3. : SpeedupTime for Wright Function

EVEN PARABOLIC FUNCTION				
Parameters		**Vectorize**	**GPU Time**	**Speedup**
A	**Step Size**	**T_V**	**T_G**	**T_V/T_G**
A=1	1e-3	0.129	0.0777	1.6641
A=4	1e-3	0.0938	0.0781	1.2002
A=6	1e-3	0.1074	0.0762	1.4094
A=1	1e-4	0.7124	0.682	1.0446
A=4	1e-4	0.7651	0.6838	1.119
A=6	1e-4	0.7292	0.6829	1.0678
A=1	1E-5	**OUT OF MEMORY**		
	TOTAL AVERAGE SPEED			**1.25085**

Table 4. : Speedup Time for Even Parabolic Function

RIEMANN ZETA FUNCTION							
Parameters		**Sequential**	**Vectorize**	**GPU Time**	**Speedup**		
S	**Step Size**	**T_S**	**T_V**	**T_G**	**T_S/T_V**	**T_V/T_G**	**T_S/T_G**
2	1.00E+06	0.03	0.01	0.0052	3.25	2.27	7.384
3	1.00E+06	0.13	2.93E-02	0.03	1.06	1.06	4.64
4	1.00E+06	0.12	0.03	0.0239	3.88	1.39	5.389
2	1.00E+05	0.002	0.00	0.0011	1.93	1.36	2.636
3	1.00E+05	0.01	0.01	0.0093	1.98	1.08	2.1290
4	1.00E+05	0.0177	0.0036	0.0031	4.916	1.161	5.7096
2	9.00E+05	0.04	0.01	0.0043	3.31	2.63	8.70
3	9.00E+05	0.11	0.03	0.0247	4.05	1.13	4.5748
4	9.00E+05	0.10	2.97E-02	0.0222	3.65	1.34	4.887
		Total Average Speed			**3.11**	**1.49**	**5.116**

Table 5. : SpeedupTime for Remann Zeta Function

BESSEL FUNCTION							
Parameters		**Sequential**	**Vectorize**	**GPu Time**	**Speedup**		
Alpha	**Step Size**	**T_S**	**T_V**	**T_G**	**T_S/T_V**	**T_V/T_G**	**T_S/T_G**
0.9	1e-3	2.701	0.03	0.02	90.05	1.111	100.0
1.5	1e-3	3.291	0.029	0.02	112.72	1.057	119.2
1.9	1e-3	2.268	0.029	0.02	78.23	1.043	81.61
0.9	1e-2	0.428	0.006	0.00	62.97	1.283	80.79
1.5	1e-2	0.367	0.006	0.00	58.31	1.166	68.03
1.9	1e-2	0.260	0.004	0.00	54.35	1.066	57.97
0.9	1e-5	350.99	OUT OF MEMORY				
		Total Average Speed			**76.11**	**1.121**	**79.14**

Table 6. : Speedup Time for Bessel Function

ODD PARABOLIC FUNCTION				
Parameters		**Vectorize**	**GPU Time**	**Speedup**
A	**Step Size**	**T_V**	**T_G**	**T_V/T_G**
A=1	1e-3	0.0848	0.072	1.1786
A=4	1e-3	0.0886	0.072	1.2314
A=6	1e-3	0.0862	0.0736	1.1705
A=1	1e-4	0.7974	0.6784	1.1754
A=4	1e-4	0.7456	0.6768	1.1017
A=6	1e-4	0.7354	0.6767	1.0869
A=1	1E-5	**OUT OF MEMORY**		
	TOTAL AVERAGE SPEED			**1.157416667**

Table 7. : Speedup Time for Odd Parabolic Function

6. Conclusion

The graphical processing unit is visually and visibly changing the path of standard cause computing. In this work, different special mathematical functions are computed using Graphical processing unit (GPU). While computing special function we have observed acceleration because of multiple processing cores of graphics processor. Special codes for these functions were developed. Then using Parallel Computing Toolbox of MATLAB vectorized version of these codes are developed and computed using GPU hardware. Thus we can observe that the special mathematical functions can be successfully implemented on GPU.

References

1. Kiryakova, Virginia. ”The special functions of fractional calculus as generalized fractional calculus operators of some basic functions.” Computers mathematics with applications 59, no. 3 (2010): 1128-1141.
2. Gil, Amparo, Javier Segura, and Nico M. Temme. Numerical methods for special functions. Society for Industrial and Applied Mathematics, 2007.
3. Patil, Parag, Navin Singhaniya, Chaitanya Jage, Vishwesh A. Vyawahare, Mukesh D. Patil, and P. S. V. Nataraj. ”GPU Computing of Special Mathematical Functions used in Fractional Calculus.” Frontiers in Fractional Calculus 199 (2018).
4. Keluskar, Yugesh C., Megha M. Navada, Chaitanya S. Jage, and Navin G. Singhaniya. ”Implementation of Airy function using Graphics Processing Unit (GPU).” In ITM Web of Conferences, vol. 32, p. 03052. EDP Sciences, 2020.
5. Choy, Ron, and Alan Edelman. ”Parallel MATLAB: Doing it right.” Proceedings of the IEEE 93, no. 2 (2005): 331-341.
6. Sharma, Gaurav, and Jos Martin. ”MATLAB®: a language for parallel computing.” International Journal of Parallel Programming 37, no. 1 (2009): 3-36.
7. Diener, Joseph E., and Atef Z. Elsherbeni. ”FDTD Acceleration using MATLAB Parallel Computing Toolbox and GPU.” Applied Computational Electromagnetics Society Journal 32, no. 4 (2017).
8. Owens, John D., Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. ”GPU computing.” Proceedings of the IEEE 96, no. 5 (2008): 879-899.
9. Banks, Nicola E. ”Insights from the parallel implementation of efficient algorithms for the fractional calculus.” (2015).
10. Singhaniya, Navin G., Mukesh D. Patil, and Vishwesh A. Vyawahare. "Implementation of special mathematical functions for fractional calculus using DSP processor." In 2015 International Conference on Information Processing (ICIP), pp. 811-816. IEEE, 2015.
11. “Parallel Computing”, Wolfram Mathworld. [Online].Available:http://mathworld.wolfram.com/ParallelComputing.html
12. “Best Practices for MATLAB GPU Coding”, Cornell University Center for Advanced Computing [Online].Available:https://www.cac.cornell.edu/matlab
13. N.E.Banks,“Insights from the parallel implementation of efficient algorithms for the fractional calculus”, University of Chester, United Kingdom, 2015.