ISSN (Online) 2394-6849

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

Implementation of Numerical Methods for Partial Differential Equation Using Parallel Computing

[1] Mr. Akshay A. Jadhav, [2] Mrs. Trupti P. Agarkar, [3] Mr. Vishwesh A. Vyawahare, [4] Mr. Mukesh D. Patil
[1][2][3] Department of Electronics Engineering, [4]Department of Electronics and Telecommunication Engineering
[1][2][3][4] Ramrao Adik Institute of Technology

***Abstract* - The performance and use of parallel computing in the field of differential calculus is increased tremendously opening up new avenues for applying these in the field of numerical computation for high speed performance. The computation time required to find analytical as well as numerical solution is tested and compared. In this work we have harnessed this property of GPU to accelerate the grid point calculations for numerical calculations and the performance of numerical method using CPU and GPU is compared. The numerical Methods for integer order PDE are studied, analyzed and implemented on GPU using parallel computing toolbox of MATLAB. The finite difference methods of PDE like explicit, implicit method are tested for the results, for parabolic, hyperbolic and elliptical type of PDE's. The positive speed up is achieved for elliptical type of PDE. The verification of results with the analytical solution is made by the mean square error.**

Index Terms: Parallel Computing Toolbox, GPU Computing, Partial Differential Equation, Numerical Method.

I.INTRODUCTION

Partial differential equations (PDEs) are used to describe most of the physical phenomena's in the domains like fluid dynamics, electricity, magnetism, mechanics, optics or heat flow. To solve these PDE's it's difficult to do the pen and paper calculations or to use the analytical methods. Finding the numerical solution or the approximate solution over the analytical solution is preferably considered. The numerical solutions are extremely abundant, because sometimes we don't have an analytical approach or the analytical solution is too slow. Instead of computing for 15 hours and getting an exact solution, we rather compute for 15 seconds and get a good approximation.

While solving the partial differential equations for example regularly used diffusion equation $U_t = U_{xx}(1D)$ *or* $U_t = U_{xx} + U_{yy}(2D)$ we require a rigorous calculations, to overcome this we use a GPGPU pipeline. It is a process of analyzing a data in terms of graphical arrays and does the parallel processing in one or more GPU and CPU. GPUs are having large number of cores but at the same time they operate at lower frequencies. Thus, the speed of operation of GPU in comparison with CPU is greater for pictures and graphical data. Transferring data into graphical form and then with the help of GPU, scan and analyze the data will result in large speedup. General-purpose computing on graphics processing units (GPGPU) means the use of a graphics processing unit (GPU), which typically handles computation only for graphics. It used to perform computation in the applications traditionally handled by the central processing unit (CPU). The use of multiple video cards in one computer, or large numbers of graphics chips, further parallelizes the already parallel nature of graphics processing. Addition to this even a single GPUCPU framework provides advantages that multiple CPUs on their own can't offer due to specialization in each chip [3]. To solve the hyperbolic (Wave Equation) and elliptical (Poisson's Equation) type of PDE we used the explicit method as the finite difference numerical method.

II. INTRODUCTION TO PDE

PDE (Partial Differential Equation) is a partial derivative of he equation and contains an unknown function depending on more than one variable. For a general 2nd order linear PDE $AU_{xx} + BU_{xy} + CU_{yy} + DU_x + EU_y + FU = G$ [1], [2] Where A, B, C, D, E, F, G are all functions of x,y. This equation is classified w.r.t sign of the discriminate ΔS i.e. ΔS = B2 Δ 4AC. The few regularly tested PDE's are classified based on the values of Δ [1].

The PDE is solved to find the unknown function U. This solution for PDE can be of two types first is analytical (i.e. Exact) solution and second is the numerical (i.e. Approximate) solution, Which will satisfies any initial and boundary conditions mentioned with PDE [7]. Analytical solutions are the

ISSN (Online) 2394-6849

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

exact solutions and can be achieved by pen and paper calculations where as Numerical solutions are approximate solutions and we are unable to find it in finite time and it is difficult to solve it by using pen and paper. This distinction between analytical and numerical can vary with the applications. In Most of the applications PDEs do not have analytical solutions so it is a necessary to use a numerical procedure to find an approximate solution [3].

To find an approximate solution we have to approximate the independent variables at discrete values and implement the numerical method via a computer program. To find the unique solution we need to concentrate on three things

1) Domain Dependency: The domain for PDE means the region in which PDE is applied in space and time variables. In practical applications, PDE will be defined on a bounded domain, or in a closed boundary [4], [6].

2) Initial Condition: we have to define a initial condition such that it specifies the value of unknown function u at the initial time on the domain [1], [2], [11].

3) Boundary Condition:

Dirichlet Boundary Condition: The values of unknown function U on the domain boundary are specified by the dirichlet boundary condition. Numerically, this condition guarantees the existence and uniqueness of the PDE [1], [11].

Von-Neumann Boundary condition: The von-Neumann condition is defined for the domain boundary as the normal derivatives with respect to a variable. The von-Neumann condition can be defined by the condition: (y) [2], [6], [11].

Unlike the Dirichlet condition, the Neumann condition ensures only that the equation has a unique solution up to an additional constant. When the Neumann condition is constant, it implies that the rate of variation of u across the boundary does not change [1], [2].

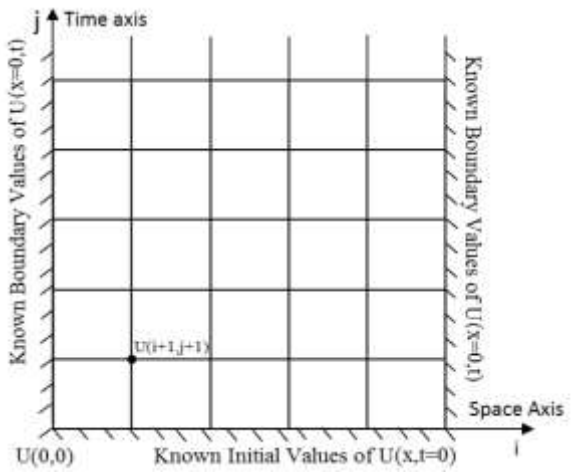

Fig. 1. Initial and Boundary Condition considered for grid Solution [1]

III. PARALLEL COMPUTING TO SOLVE NUMERICAL METHODS FOR PDE

While dealing with big technical problem we came across the issues like long computational running time for the process or the part of the code is computationally intensive or the system of application for which we are coding has large input data set. All these issues hammer the speed up of process and slow down the performance. The best solution to deal with this problem is parallel processing.

General purpose computation on GPUs (GPGPU) uses a graphics processing unit (GPU) for general-purpose parallel processing applications. No dependencies or communications between tasks is the most important criteria for the parallel computing [14].Vectorization is one of the concern thing for GPU coding, Most operations on the GPU operate in a vectorized fashion, one operation can be performed on up to four values at once. This functionality of producing output from two different independent input data sources is useful in graphics because almost every basic data type is a vector (either 2-D, 3-D, or 4-Dimensional). [12], [15]

When MATLAB's GPU enabled functions are executed on the GPU, data will be transferred from MATLAB's workspace to GPU device memory. The command 'gpuArray' provides a specific array type for such data transfer, then GPU enabled functions can run on these data. The command 'gather' returns the calculated results, which are stored in GPU, back to MATLAB's workspace [14].The procedure of GPU computing in MATLAB is as shown in figure(2)

ISSN (Online) 2394-6849

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

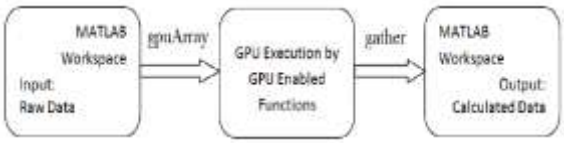

Fig. 2. Steps followed in GPU computing

The advantage of MATLAB GPU programming is that users can easily utilize GPU computing by adding just few more commands to their original MATLAB codes. Disadvantages include that only limited MATLAB functions are GPU enabled and the computing efficiency of MATLAB GPU is less than those codes written in CUDA [14].

IV. GPU COMPUTING WITH MATLAB

GPU accelerated computing off loads compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. A simple way to understand the difference between a GPU and a CPU is to compare how they process tasks [8]. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously [8], [14].

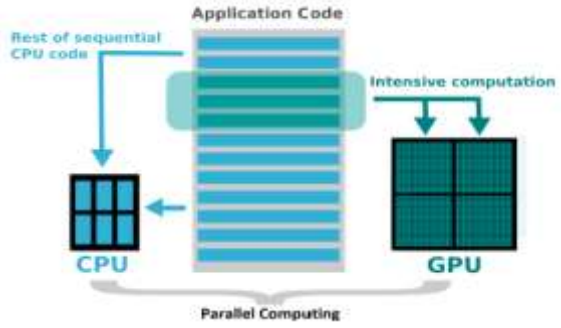

Fig. 3. Working of GPU Acceleration [14]

we are using the hybrid GPU-CPU system in which CPU will be treated as a host or a primary processor and the GPU is called the device or a secondary processor. When we run an application code on the machine the CPU as main processor decides which is the "computationally-intensive" part of the program [9]. This part actually takes large amount of computational time and due to which its data set can be executed in parallel on GPU with the ongoing process [14]. We are using MATLAB as our software platform which supports the NVIDIA GPU, with the help of that we came to understand which part of the application code is consuming more time so, that part can be parallelized. 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 executed output data set is gathered by the GPU memory and transferred back to the CPU memory, which will be available in MATLAB workspace and for further calculations. As a result to all this the computationally complicated analysis of mathematical equations and methods programs runs faster and this results in efficient processing for this computationally intensive codes. [12], [15]

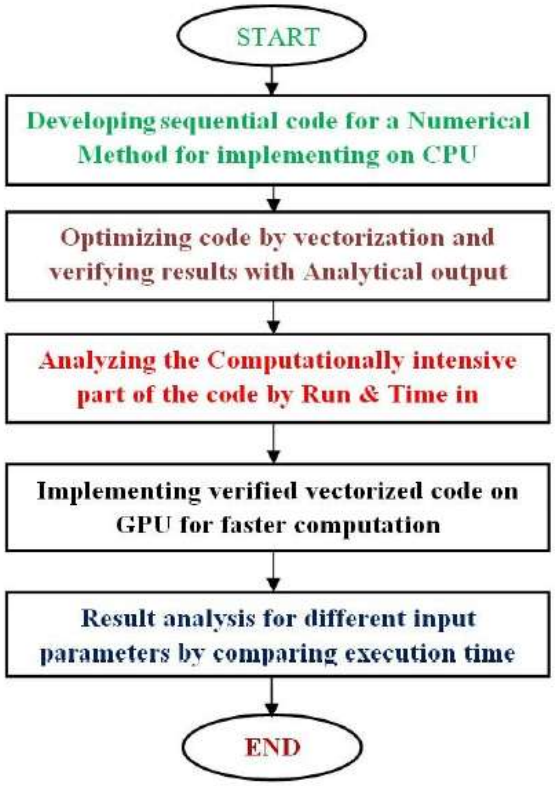

Fig. 4. Workflow of GPU Computing

V. PARALLEL COMPUTING TOOLBOX (PCT)

The Parallel Computing Toolbox (PCT) is a part of MATLAB toolbox. It helps us to solve computationally intensive and data-intensive problems using MATLAB speedily on our local multi core computer or on Shared Computing Cluster. Parallel processing operations such as gpuArray and gather,

ISSN (Online) 2394-6849

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

implements the tasks and perform data-parallel algorithms in MATLAB. Converting serial MATLAB applications to parallel MATLAB applications it generally requires few code modifications and no programming in a low-level language is necessary. We can run our parallel applications interactively or in batch [14].

The main advantages of parallel computing are:

1) Significant amount of time and money are saved.
2) Larger problems can be solved.
3) Achievement of parallelism.
4) Usage of non-local resources.

Parallel Computing Toolbox helps us to solve computationally and data-intensive problems using multi core processors, GPUs, and computer clusters. High-level constructs parallel for-loops, special array types, and parallelized numerical algorithms which helps to parallelize MATLAB applications without CUDA or MPI programming. The toolbox uses the full processing power of multi core desktop Without changing the code, we can run the same applications on a computer cluster or a grid computing service.

VI. HARDWARE ASPECTS:

leopard - GPU compute node has

- Tesla K40 GPU with 2880 cores 12 GB RAM.
- Intel(R) Core(TM) i7-7500U CPU @ 2.90GHz with 6 cores 16 GB RAM.

VII. NUMERICAL METHOD TO SOLVE HYPERBOLIC PDE

The time dependent (transient) phenomena are described by Hyperbolic Equations for e.g. sea tides,String vibrations,wave motion,liquid flow. From the general 2nd order linear PDE for the value of discriminant Δ = B2-4AC > 0 particular PDE is classified as hyperbolic PDE. The 1D wave equation, $u_{tt} = c^2 u_{xx}$ (c = constant > 0), is the simplest model of a physical system involving a hyperbolic PDE. [8] Let us consider the second order hyperbolic equation in one space dimension

$$u_{tt} = c^2 u_{xx}, \qquad \forall (x,t) \epsilon (0,1) \times (0,T). \qquad (1)$$
$$u(x,0) = f(x), U_t(x,0) = g(x) \qquad \forall 0 \le x \le 1,$$
$$u(0,t) = U_L(t), u(1,t) = u_R(t) \qquad \forall t \epsilon (0,T),$$

Considering the Dirichlet boundary conditions and the initial conditions, and considering the mesh which is uniformly spaced [3], [8].

- T= $Nt\Delta$ t = $j\Delta$ t
- xj = $xn\Delta$ x=iΔx

where $\Delta x = \frac{xr - xl}{xn}$ & $\Delta t = \frac{T}{Nt}$

To find the solution by explicit method for wave equation we have to discritize it over a domain of boundaries,for that will use finite difference approximations to replace the second order derivatives,namely

$$u_{tt}(x_j, t_n) \approx \frac{(u(x_j,t_{n+1}) - 2u(x_j,t_n) + u(x_j,t_{n-1}))}{(\Delta x)^2 + O((\Delta t)^2)},$$
$$u_{xx}(x_j, t_n) \approx \frac{(u(x_{j+1},t_n) - 2u(x_j,t_n) + u(x_{j-1},t_n))}{(\Delta x)^2 + O((\Delta x)^2)},$$

Since the errors are of orders of (t)2 and (x)2, we expect to be able to choose the space and time step sizes of comparable magnitude: t x

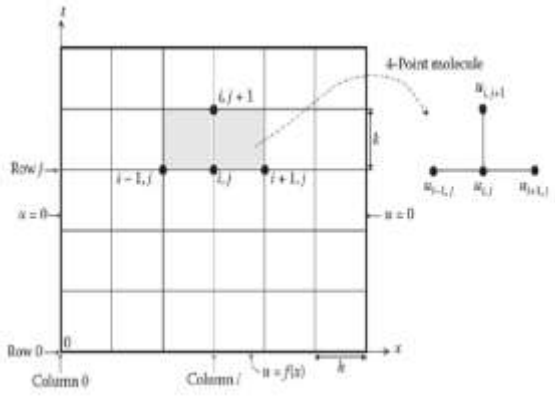

Fig. 5. schematic diagram of a five point formula for Explicit scheme [3]

Substituting the finite difference formulae into the partial differential equation(1), and rearranging terms, we are led to the iterative system

$$u_i^{j+1} = \mu^2 u_{i+1}^j + 2(1-\mu^2)u_i^j + \mu^2 u_{i-1}^j - u_i^{j-1} \qquad (2)$$

for i = 1,2,3,..... and j = 1,....N_t - *1*.

for the numerical approximations $u_j^i \approx u(x_j, t_n)$ to the solution values at the mesh points.The positive parameter

ISSN (Online) 2394-6849

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

$$\mu = \frac{c\Delta t}{\Delta x} > 0,$$

Stability Analysis:

For the stability of 2, denoting $\mu = rho * \frac{c\Delta t}{\Delta x}$ and using the von Neumann technique, we get the condition as $|\mu| \leq 1,$ under which this method is stable [6].

Example:

$$u_{tt}(x,t) = u_{xx}(x,t), \qquad xl < x < xr, 0 < t < T,$$
$$u(x,t)|_{t=0} = f(x) = \sin(\pi * xj), \qquad xl \leq x \leq xr,$$
$$u(x,t)|_{x=0} = gl(t) = 0, u(x,t)|_{x=1} = gr(t) = 0 \qquad 0 \leq t \leq T.$$

The code solves the above problem with string of length l, T = 1 by the Explicit scheme. The boundary and initial conditions are chosen properly such that our problem has the analytic solution

$$u(x,t) = \sin(\pi * x) * \cos(\pi * t)$$

Input		Output			
Xn Space Samples	Nt Time Samples	Mean Square Error (MSE)	CPU Computation Time (Tv)	GPU Computation Time (Tg)	Speed Up (Tv/Tg)
500	500	2.95E-27	0.0218	0.0307	0.7109
1500	1500	2.94E-28	0.0516	0.0606	0.8514
5000	5000	1.95E-28	1.3677	2.1345	0.6407
15000	15000	7.91E-29	12.7998	13.0192	0.9831

TABLE I
RESULTS OBTAINED FOR WAVE EQUATION BY EXPLICIT SCHEME

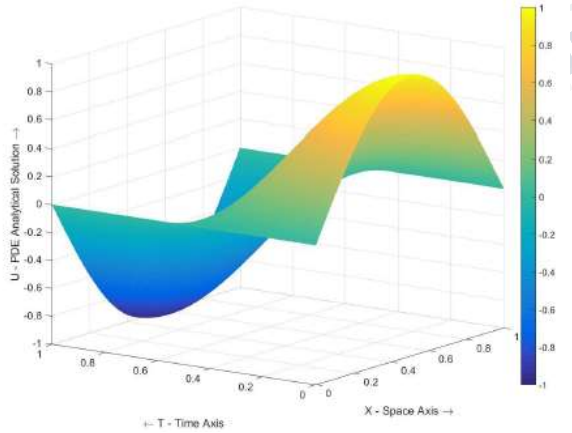

Fig. 6. Analytical solution for hyperbolic equation by Explicit Scheme

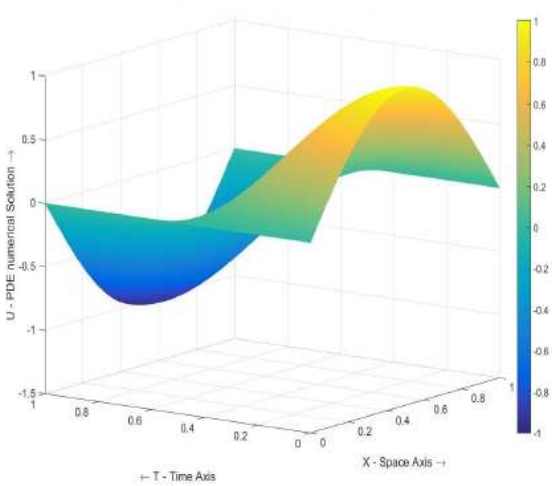

Fig. 7. Numerical solution for hyperbolic equation by Explicit Scheme

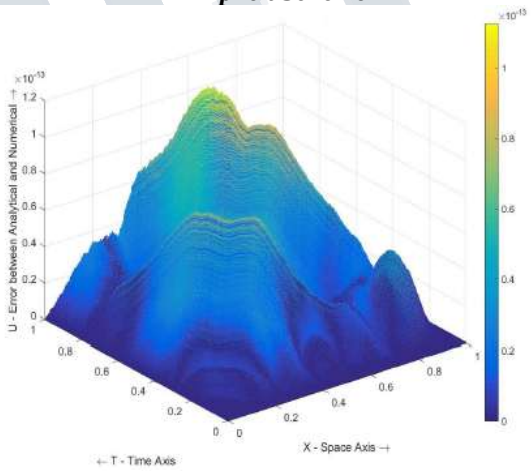

Fig. 8. Error Analysis for hyperbolic equation by Explicit Scheme

1) The Analytical Solution (figure 6) and Numerical Solution (figure 7) are almost matching with the error specified in (figure 8) result table for each space and time step combination.

2) As step size for space or time decreases that is as we are going to increase space and time samples the truncation error tends to zero and solution becomes more stable also it approaches towards the exact solution.The mean square error(MSE) is of the order of 10^{-29}

3) Speed Up for GPU is negative the reason for this is the for loop which we are unable to vectorize contains the dependency issues. As the method itself has the dependencies on neighbouring values unless they are

ISSN (Online) 2394-6849

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

calculated it is difficult to find further values, but we are trying to overcome this.

4) one important thing is both the time and space samples are same to achieve the stability criteria.

VIII. NUMERICAL METHOD TO SOLVE ELLIPTICAL PDE

Eliptical partial differential equatins arise usually from equilibrium or steady state problems and their solutions. From the general *2nd* order linear PDE for the value of discriminant $\Delta = B2 - 4AC < 0$ particular PDE is classified as elliptical PDE. The very well known elliptical equations are poisson's equation, $\frac{\delta^2\phi}{\delta x^2} = f(x,y)$ which is often written as $\nabla^2\phi = f(x,y)$ and Laplace's equation $\frac{\delta^2\phi}{\delta x^2} = \nabla^2\phi = 0$ [9].

Suppose that Ω is a bounded domain of R2 with boundary $\partial\Omega$

The equation

$$a(x,y)\frac{\delta^2 u}{\delta x^2} + 2b(x,y)\frac{\delta^2 u}{\delta x \delta y} + c(x,y)\frac{\delta^2 u}{\delta y^2} = f(x,y,u,\frac{\delta u}{\delta x},\frac{\delta u}{\delta y}) \quad (3)$$

is said to be elliptic if b2 □ ac < 0 for all points (x; y) Considering the Poisson's equation [9].

$$-(uxx + uyy) = f(x,y) \quad \forall (x,y)\epsilon\Omega = (0,1)^2 \quad (4)$$

$$u|_{\delta\Omega} = g(x,y) \quad \forall (x,y)\epsilon\delta\Omega. \quad (5)$$

$$u(0,y) = u^L(y), u(1,y) = u^R(y) \quad \forall y\epsilon(0,1). \quad (6)$$

$$u_y(x,0) = g^B(x), U_y(x,1) = g^T(x) \quad \forall x\epsilon(0,1). \quad (7)$$

As before, we assume that is covered by a uniform grid $x_i = ih, y_j = jh, \quad 0 \le i,j \le N_i, h = \frac{1}{N_i}$

The approximate solution at point (xi, yj) is denoted by u^j_i . Using the secondorder central difference [1],

$$\delta_x^2 = \frac{u^j_{i+1} - 2u^j_i + u^j_{i-1}}{h^2}$$

$$\delta_y^2 = \frac{u^{j+1}_i - 2u^j_i + u^{j-1}_i}{h^2}$$

to approximate the derivatives uxx and uyy in 4 respectively,

we obtain the five-point difference scheme:

$$\frac{u^j_{i+1} - 2u^j_i + u^j_{i-1}}{h^2} + \frac{u^{j+1}_i - 2u^j_i + u^{j-1}_i}{h^2} + f^j_i, = 0, \quad 1 \le i,j \le J-1 \quad (8)$$

or

$$u^j_{i+1} + u^j_{i-1} + u^{j+1}_i + u^{j-1}_i - 4u^j_i = -h^2 f^j_i. \quad (9)$$

Example:

with a homogeneous Dirichlet boundary condition over the domain $\Omega = (0; 1)2$. The right-hand side function f is chosen as $f(x, y) = 2\pi 2\sin(\pi x)\sin(\pi y)$ such that the exact solution is given by $u(x; y) = \sin(\pi x)\sin(\pi y)$.

Following results were obtained on specified hardware:

Input J Time & Space Samples	Output Mean Square Error (MSE)	CPU Computation Time (Tv)	GPU Computation Time (Tg)	Speed UP (Tv/Tg)
50	2.82E-08	0.2237	0.1844	1.2131
70	7.25E-09	0.8776	0.6316	1.3894
90	7.91E-11	2.6558	2.4386	1.0891
120	3.97E-12	2.8541	9.2136	0.3097
140	NAN	Out of Memory		

TABLE II
RESULTS OBTAINED FOR POISSON'S EQUATION BY EXPLICIT SCHEME

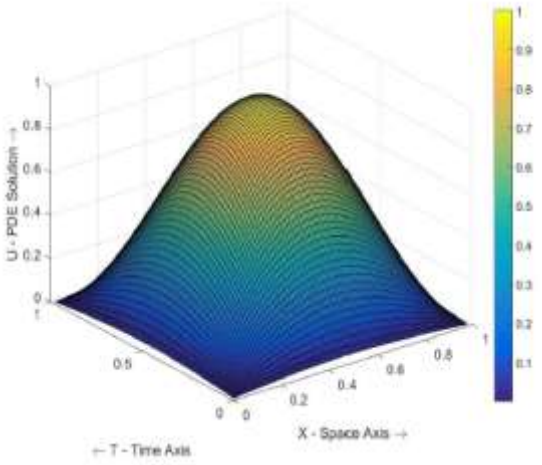

Fig. 9. Analytical solution for Eliptical equation

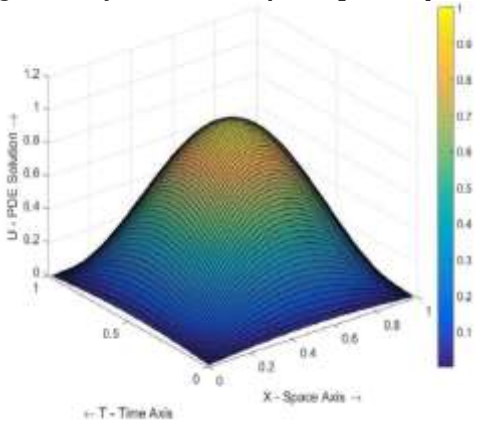

Fig. 10. Numerical solution for Eliptical equation

ISSN (Online) 2394-6849

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

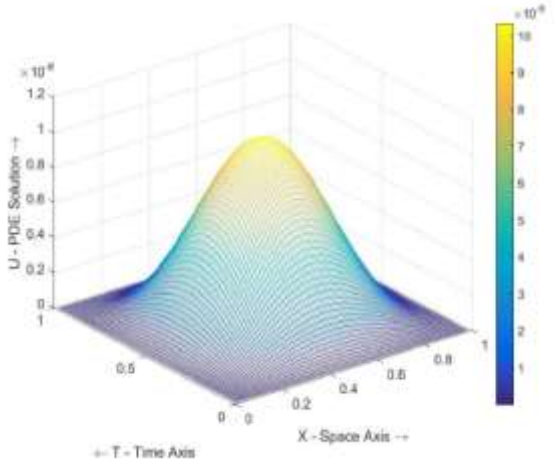

Fig. 11. Error Analysis for Eliptical equation

Result Analysis:

1) The Analytical Solution (figure 9) and Numerical Solution (figure 10) are almost matching with the error (figure 11) specified in result table II for each space and time step combination.

2) As we are going to increase space and time samples the truncation error tends to zero and solution becomes more stable also it approaches towards the exact solution.

3) In case of speed ups for large number of input samples better speed up (almost 3 times) is achieved between vectorized code and sequential code.

4) for the last sample we can see that at J=140 GPU reaches its memory limit and through a error message as 'out of memory'.The error is Not a Number (NAN).

IX. CONCLUSION

To evaluate the benefits of use of the GPU to solve Partial Differential Equations we ran a benchmark study between analytical solution and the numerical solution, in which we measured the amount of time the algorithm took to execute various time steps for increasing number of grid size. It is observed that increase in complexity of the functions increases the speed-up. The improved speed up was achieved for large number of samples. [13] [14]

On the hardware side the execution speed offered by the GPU depends on its number of cores and compute capability. The disadvantage for this parallel computing with MATLAB is, we don't have any control over the number of cores of GPU to be used for the execution. The usage of cores of GPU will be decided by the MATLAB. We will have that advantage in CUDA.

To achieve speed up with the GPU our application code must satisfy the important criteria as the fact that sending the data between CPU and GPU must take less time than the performance gained by running on GPU, then only it will be good candidate of GPU functionality.The promising power of GPU with MATLAB is the main advantage. GPU enabled functions and gpuArray are useful for speedup without the use of CUDA programming.

REFERENCES

[1] Farlow, Stanley J. Partial differential equations for scientists and engineers. Courier Corporation, 1993.

[2] Smith, Gordon D. Numerical solution of partial differential equations: finite difference methods. Oxford university press, 1985.

[3] Lindfield, George, and John Penny. Numerical methods: using MATLAB. Academic Press, 2012.

[4] Ames, William F. Numerical methods for partial differential equations. Academic press, 2014.

[5] Peaceman, Donald W., and Henry H. Rachford, Jr. "The numerical solution of parabolic and elliptic differential equations." Journal of the Society for Industrial and Applied Mathematics 3.1 (1955): 28-41.

[6] Arnold, Douglas N. "Stability, consistency, and convergence of numerical discretizations." Encyclopedia of Applied and Computational Mathematics. Springer Berlin Heidelberg, 2015. 1358-1364.

[7] John, Volker. "Numerical methods for partial differential equations." Lecture notes (2013).

[8] Trangenstein, John. "Numerical solution of hyperbolic conservation laws."Cambridge University Press, (2009).

[9] Urroz, G. E. "Numerical Solution of Laplace Equation." Utah State University paper (2004).

[10] Peter Knabner, Lutz Angermann. "Numerical Methods for Elliptic and Parabolic Partial Differential Equations" Springer-Verlag,New York Berlin Heidelberg,2000.

[11] Arnold, Douglas N. "Lecture notes on Numerical Analysis of Partial Differential Equations." (2012).

[12] Chun-Hasiang Han, "Introduction to Matlab GPU Acceleration for COmputational Finance." Natational Tsing-Hua University,Taiwan, December 26, 2013.

[13] "Partial Differential Equation". En.Wikipedia.Org, 2017, https:// en.wikipedia .org/ wiki/ Partial-differential-equation.

[14] "NVIDIA On GPU Computing And The Difference Between Gpus And Cpus". Nvidia.Com, 2017, http://www.nvidia.com/object/what-isgpu-computing.html.

[15] "Graphics Processing Unit". En.Wikipedia.Org, 2017, https:// en .wikipedia .org /wiki / Graphics-processing -unit.