# Parallel Numerical Solution of the 2-D Diffusion Equation using PETSc

Vishal Indivar Kandala

July 2, 2021

**Abstract**

In this work, the 2-D Diffusion equation is solved on a structured grid using the Finite Difference Method, This is done using a parallelized code, utilizing the PETSc library's C interface,run on the Terra cluster. Krylov Subspace Solvers were utilized and the solutions obtained (Time to convergence) through Gauss Siedel Iteration and the standard GMRES method are compared, along with a strong scaling analysis of both those solutions is illustrated.

## 1 Introduction

The objective of this work is to solve the 2-D Diffusion equation on a structured grid using the Parallel PETSc Library, this section outlines the problem domain, boundary conditions and the anatomy of the governing differential equation.

### 1.1 Domain and Boundary Conditions

A rectangular domain of unit thickness, with length 0.4 m and width 0.3m is bounded by dirichlet boundary conditions on all four sides, the domain can be seen in Fig.1. The material is assumed to have thermal diffusivity ($\alpha$) of 11.234 x $10^{-5} m^2/s$ and a thermal conductivity of 280 W/m.K, the plate is assumed to be homogenous and as having constant properties across the domain.

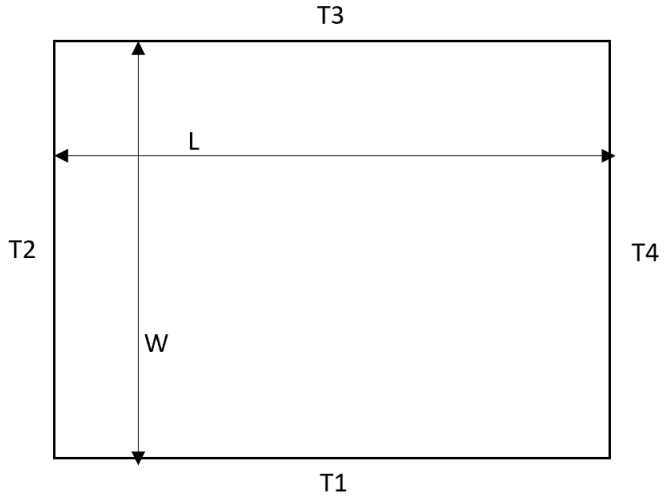


Figure 1: The Computational Domain

The Dirichlet boundary conditions imposed on the system mean that the boundaries are all held at temperatures shown in Table.1, where the edge which is held at the label can be seen in Fig.1.

| Label | Temperature |
|---|---|
| $T_1$ | 313 K |
| $T_2$ | 273 K |
| $T_3$ | 283 K |
| $T_4$ | 273 K |

Table 1: Boundary Temperatures

### 1.2 Governing Differential Equation

The 2-D Steady State Heat Equation or the diffusion equation is shown in Eq.1, where T is the dependent variable and x,y are the independent variables.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

This is a linear,homogenous,Second Order Elliptic Partial Differential Equation ($B^2 - 4AC < 0$) and hence exhibits two-way street behaviour or Boundary Value Problem (BVP) behaviour. This means that the information propagation happens at infinite speed and in all directions.

Diffusion phenomenon involves a gradual reduction of gradients across the domain and hence results in an averaging or smoothening effect and that is what is expected from the solution irrespective of the methodology implemented.

### 1.3 PETSc Linear Algebra Solvers

PETSc[1] is a Scientific Computing library that contains highly optimized, parallelized routines for Matrix and Vector Operations through custom objects (Vec,Mat,KSP) as well as the Distributed Array (DM) data structure which links the linear algebra solvers to the matrix and vector objects.

## 2 Methodology

Numerical solution of a Partial Differential Equation with non-homogenous boundary conditions can be obtained through the following three steps, each of which will be detailed in this section

1. Grid Generation.
2. Obtaining a System of Linear Equations.
3. Solving a system of Linear Equations.

### 2.1 Grid Generation

A structured, uniform, node based grid with a **star stencil** has been implemented to solve this problem, in the program, the variable i has been assigned to grid points along x-axis and the variable j has been assigned to grid points along y-axis.The stencil can be seen in Fig.3 and a 8x6 grid can be seen in Fig2. The grid spacing $\Delta x$ and $\Delta y$ for an N x M grid are given in Eq.2

$$\Delta x = \frac{L}{N-1}; \Delta y = \frac{W}{M-1} \tag{2}$$

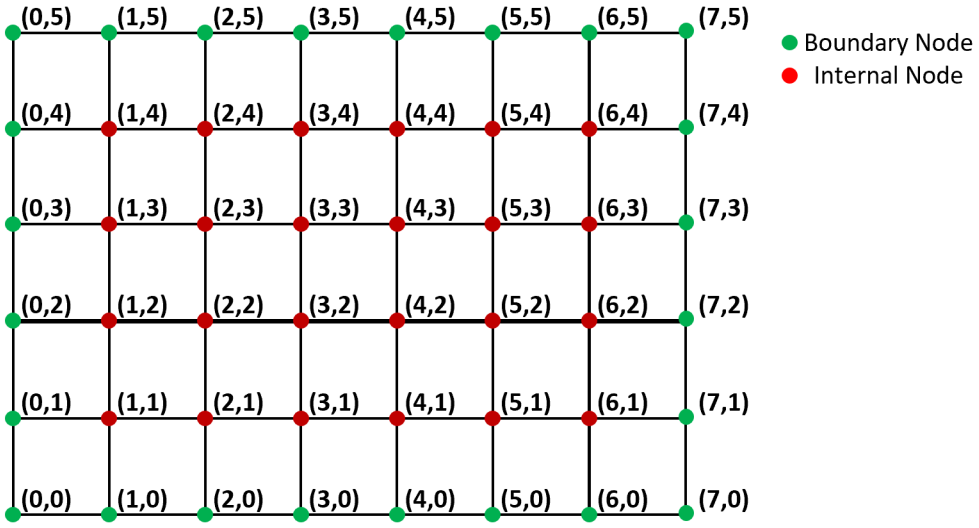


Figure 2: The Finite Difference grid utilized to solve the diffusion equation with the boundary grid points marked in green and the internal grid points marked in red, along with their indices.

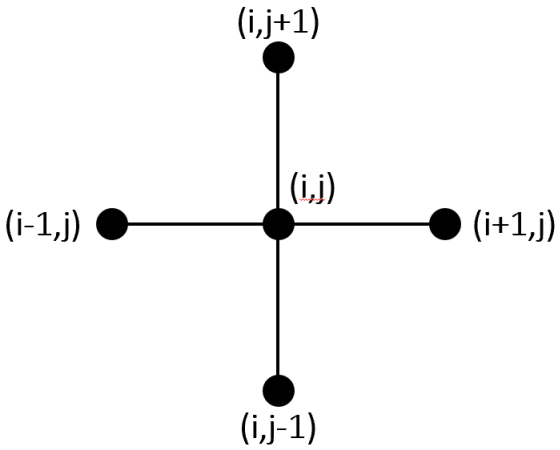


Figure 3: The Finite Difference star stencil implemented in this project

## 2.2 Finite Difference Scheme

To generate the linear system of equations, a 2-D second order central differencing scheme was implemented along both the x and y axes, on the star stencil, as shown in Eq.4

$$\text{Let} z = f(x, y) \tag{3}$$

$$\frac{\partial^2 z}{\partial x^2} = \frac{z(i+1, j) + z(i-1, j) - 2z(i, j)}{\Delta x^2} \tag{4}$$

To derive the accurate finite difference equations at various locations, the grid has been divided into 10 different regions which are illustrated in Fig.4, the finite Difference equations obtained in each region are described in the following sub sections.

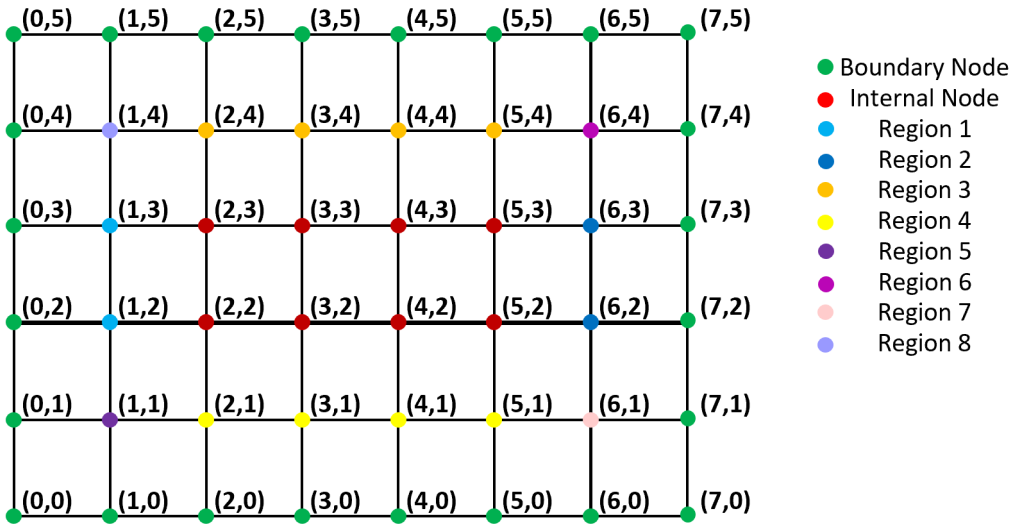


Figure 4: Boundary Condition Implementation through splitting the domain into multiple regions

### 2.2.1 Internal Nodes

For internal nodes i.e i>1,j>1 and i<N-2,j<N-2, discretizing the diffusion equation using the second order central differencing approach shown above leads to linear equations shown in Eq.7

$$\frac{1}{\Delta x^2} = D_x \tag{5}$$

$$\frac{1}{\Delta y^2} = D_y \tag{6}$$

$$2(D_x + D_y)T(i, j) - D_x T(i-1, j) - D_x T(i+1, j) - D_y T(i, j+1) - D_y T(i, j-1) = 0 \tag{7}$$

### 2.2.2 Boundary Nodes

Since Dirichlet Boundary conditions are to be applied, the finite difference equation obtained at the boundaries i.e i=0 or j = 0 or i=M-1 or j=N-1 simply places the boundary temperature in it's location as shown in Eq.8, Eq.9,Eq.10 and Eq.11.

$$T(i, 0) = T_1(i > 0; i < M-1) \tag{8}$$

$$T(0, j) = T_2(j > 0; j < N-1) \tag{9}$$

$$T(i, N-1) = T_3(i > 0; i < M-1) \tag{10}$$

$$T(M-1, j) = T_4(j > 0; j < N-1) \tag{11}$$

At the corners i.e the singularities, these points are not solved for anyway and hence, to maintain numerical consistency, decoration values obtained by averaging the boundary temperatures of the edges that coincide in that corner are placed.

#### 2.2.3 Regions 1,2,3,4

These regions correspond to the nodes that are close to the edges at the left,right,top and bottom boundaries respectively, here, one of the temperatures is a known value and is equal to the boundary value, hence, these equations have a constant term and three unknowns with co-efficients. The equations for Regions 1,2,3 and 4 are shown in Eq.12,Eq.13,Eq.14 and Eq.15 respectively.

$$2(D_x + D_y)T(1,j) - D_xT(2,j) - D_yT(1,j+1) - D_yT(1,j-1) = D_xT_2 \tag{12}$$

$$2(D_x + D_y)T(M-2,j) - D_xT(M-3,j) - D_yT(M-2,j+1) - D_yT(M-2,j-1) = D_xT_4 \tag{13}$$

$$2(D_x + D_y)T(i,N-2) - D_yT(i,N-3) - D_xT(i+1,N-2) - D_xT(i-1,N-2) = D_yT_3 \tag{14}$$

$$2(D_x + D_y)T(i,1) - D_yT(i,2) - D_xT(i+1,1) - D_xT(i-1,1) = D_yT_1 \tag{15}$$

#### 2.2.4 Regions 5,6,7,8

These nodes are near corners of the domain and hence, two of the neighbouring nodes would have known (boundary) temperature values, which leads to Eq.16,Eq.17,Eq.18 and Eq.19 respectively.

$$2(D_x + D_y)T(1,1) - D_xT(2,1) - D_yT(1,2) = D_yT_1 + D_xT_2 \tag{16}$$

$$2(D_x + D_y)T(M-2,N-2) - D_xT(M-3,N-2) - D_yT(M-2,N-3) = D_yT_3 + D_xT_4 \tag{17}$$

$$2(D_x + D_y)T(M-2,1) - D_xT(M-3,1) - D_yT(M-2,2) = D_yT_1 + D_xT_4 \tag{18}$$

$$2(D_x + D_y)T(1,N-2) - D_xT(2,N-2) - D_yT(1,N-3) = D_yT_3 + D_xT_2 \tag{19}$$

### 2.3 Solving System of Linear Equations

Formulating as shown above would lead to a system of linear equations of the form **Ax=b** which can be solved either directly (Gauss Elimination) or through iterative methods (Jacobi,Gauss-Seidel etc).
More often than not, unless the system is a very simple system with very low condition number i.e the ratio of the highest and lowest singular values of the co-efficient matrix, for example, a Tri-Diagonal system, these systems of linear equation are not amenable to direct solution and hence iterative solvers are preffered.
Some methods such as the Strongly Implicit Procedure combines both direct and indirect (iterative) methods to obtain solutions. In this section, the Gauss-Siedel iterative method is discussed and it's implementation using the PETSC linear solver class "KSP"[2] is detailed, there will also be a brief summary of Krylov Sub-Space methods and the GMRES[6] method that is implemented as the default solver in PETSc.

#### 2.3.1 Gauss Siedel Iteration

This involves iteratively minimizing the norm of the residual for each grid point, consider an internal point i,j, then the linear equation at this point is shown in Eq.20,here we observe that the solution "propogates" to the east and north and subsequently as the solution already was undertaken in the current iteration at the south and west neighbour of the current point, updated

values are available there and the values from previous iteration are available at the north and east neighbours. This is also illustrated in Fig.5

$$T^{k+1}(i,j) = \frac{D_x T^{k+1}(i-1,j) + D_x T^{k}(i+1,j) + D_y T^{k+1}(i,j-1) + D_y T^{k}(i,j+1)}{2(D_x + D_y)} \tag{20}$$

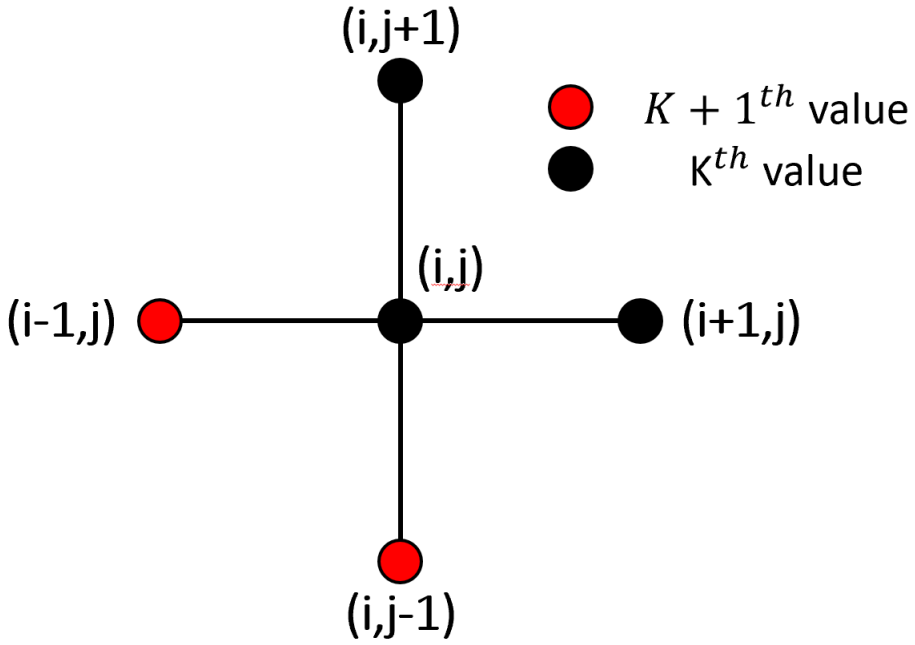


Figure 5: Information propogation in a Star-Stencil centered around the point i,j when Gauss-Siedel Iteration is used to solve the system

When considering the whole system, this can be represented as the splitting or "preconditioning" of the co-efficient matrix.

Consider the simplest solver i.e the richardson solver which is represented in Eq.22, which simply adds the residual to the current iteration to obtain the next iteration and minimizes the residual, this has been modified by adding a Pre-conditioner P as shown in Eq.23,accordingly, there are many ways in which a linear system can be factorized/split or transformed to better condition for solvability and an array of options are available in PETSC through the PC objects.

Parallelizing Gauss-Seidel iteration could be quite challenging and the most direct methods are the wavefront method and coloring method[4], however, it can be seen as the following split of the system shown in Eq.24, which means, by applying this as a preconditioner to the modified richardson iteration solution(Eq.23), we can obtain a gauss-siedel solution equivalent[3] to Eq.20 as demonstrated in Eq.25

$$\mathbf{A}x = b \tag{21}$$

$$x^{k+1} = x^k + (b - \mathbf{A}x^k) \tag{22}$$

$$x^{k+1} = x^k + \mathbf{P}(b - \mathbf{A}x^k) \tag{23}$$

$$\mathbf{A} = \mathbf{L}_* + \mathbf{U} \tag{24}$$

$$x^{k+1} = \mathbf{L}_*^{-\mathbf{1}}(b - \mathbf{U}x^k) \tag{25}$$

A slightly better iteration method which converges quicker is known as Successive Over/Under Relaxation and it simply multiplies a constant $\omega$ to the pre-conditioned residual, if this $\omega$ is chosen to be 1, then the solver is identically Gauss-Siedel and that is the way in which Gauss Siedel Iteration can be implemented in Petsc by using "richardson" solver, "sor" pre-conditioner with "omega" set to one and non-zero initialization that accelerates convergence.

Another way to implement Gauss-Seidel in PETSc is to use the "preonly" option available with the KSP solver object and using an "sor" preconditioner and setting "omega" to one as before.

### 2.3.2 Krylov Sub-Space Methods

Basic Iterative solution methods such as Jacobi, Gauss-Seidel and SOR are only valid for diagonally dominant systems and the rates of convergence can be quite low, hence more advanced methods are necessary to solve systems which are not as "nice".

Consider the initital solution $x_0$, the co-efficient matrix A, then the vectors A$X_0$,$A^2x_0$,$A^3x_0$ upto $A^kx_0$ span the "krylov Sub Space" of the system and various methods that involve these subspaces have been proven to be more efficient and adept at solving systems of linear equations, even if they are singular, assymmetric and/or not diagonally dominant.

The default solver that is implemented in PETSc and one of the most popular Krylov Subspace solver, one that does not assume any prior knowledge of the system matrix A is the Generalized Minimum Residual Method (GMRES), this involves minimizing the residual over the entire Krylov Subspace as shown in Eq.26 and the linear combination that is part of the Krylov Subspace that minimizes the residual is the solution vector $x_k$ as shown in Eq.27

$$min_{\mathbf{c}}\|A(c_0x_0 + c_1Ax_0 + c_2A^2x_0 + \dots c_kA^kx_0) - b\| \tag{26}$$

$$x_k = c_0x_0 + c_1Ax_0 + c_2A^2x_0 + \dots c_kA^kx_0 \tag{27}$$

The GMRES[6] Solver with incomplete LU Factorization is the default setting for the KSP Solver object available in the PETSc library,for the scaling analysis, this solver will be compared with the Gauss-Siedel Method.

# 3 Results and Discussion

Figure.6 shows the temperature contour of the steady state that the system described by the given domain would settle in, given that only diffusion is at play. This particular contour plot is produced with a grid of 400 nodes along the x-axis and 300 nodes along the y-axis that was solved on 4 Intel Xeon E5 Processor cores on the Terra Computing cluster. The solver used is the Generalized Minimal Residual Method with Incomplete LU Factorization pre-conditioning

From the figure, it can be observed that the boundaries are maintained at 313K,273K,283K and 273K respectively and the diffusion phenomenon spread the temperature distribution around effectively, the highest temperature in the domain is at the bottom edge and the least temperature is at the left and right edges, all the interior temperatures were between these highs and lows which was as expected. The temperature at the center of the domain is close to 290K and the temperature contours form a sort of progression from the bottom edge to the top which is what you would expect the diffusion operator to do.

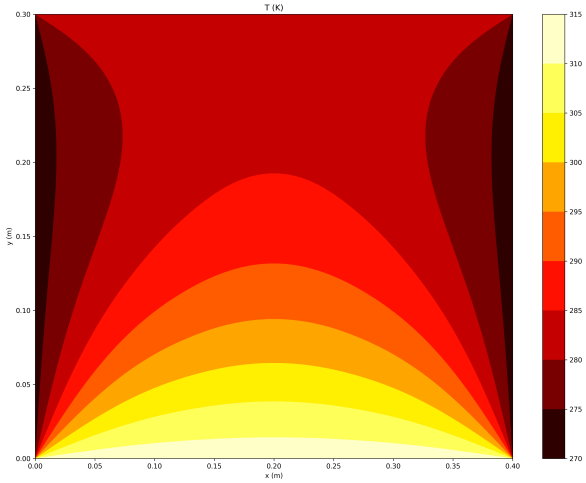


Figure 6: Steady State Temperature Contour obtained with a grid size of 400x300, solved using GMRES and ILU Preconditioning.

Figure.7 shows the temperature contour obtained with a coarser 100x100 grid implemented on 4 Intel Xeon E5 Processor Cores using the Gauss-Siedel Iteration Method, and even in this contour plot the same features can be observed as in Fig.6 i.e the averaging effect of the diffusion process is apparent.

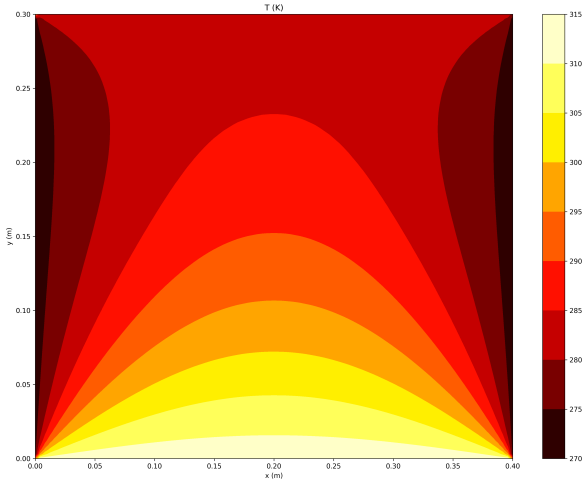


Figure 7: Steady State Temperature Contour obtained with a grid size of 100x100, solved using Gauss-Siedel Iteration.

| Case | N | M | Processors | Solver | Preconditioner | Tolerance | Iterations to Convergence |
|---|---|---|---|---|---|---|---|
| 1 | 400 | 300 | 4 | GMRES | Incomplete LU | 1e-5 | 456 |
| 2 | 100 | 100 | 4 | Richardson | SOR ($\omega = 1$) | 1e-5 | 3510 |

Table 2: Detailed Description of the cases depicted in Fig.6 and Fig.7

### 3.1 Scalability Analysis

The algorithm has been parallelized to take advantage of the capabilities of distributed computing power available through the Terra cluster which houses 28 Intel Xeon E5 cores in each node. To truly be able to gauge the benefit of parallelization, strong and weak scaling are defined, if an algorithm demonstrate strong scaling which is a stricter condition that would be ideal, to understand this metric, we look at a metric called speed up (Eq.28) which is regularly employed to analyze HPC performance. Here, $t_1$ is the time taken to execute the program by one particular core/processor where as $t_N$ is the time taken to execute the program by N cores/processors.

$$speed - up = \frac{t_1}{t_N} \tag{28}$$

Strong scaling is gauged by analysing the variation in speed-up as the number of processors increases, ideally this would be a linear relationship, however, as the number of processors increases the time taken to communicate between processors increases even as the time each processor takes to compute it's share of the code comes down and hence speed-up flattens out,this is known as Amdahl's law[5], practically though, performance tapers off even before and even fluctuates due to constraints placed by the architecture of the HPC system in use and the interconnect (communication) system in use.

In this particular project, the strong scaling performance of the gauss-seidel algorithm is tested with the problem size being set at 100x100 grid, and the GMRES Solver is analyzed with a grid size of 400x300. The computation times and the speed-up trends can be seen in Table.3, Table.4, Fig.8 and Fig.9 respectively.

| N | M | Processors | Execution Time (s) |
|---|---|---|---|
| 100 | 100 | 1 | 0.63 |
| 100 | 100 | 5 | 0.18 |
| 100 | 100 | 10 | 0.13 |
| 100 | 100 | 15 | 0.1 |
| 100 | 100 | 20 | 0.09 |
| 100 | 100 | 25 | 0.06 |

Table 3: The Problem size,Number of Processors used and the execution times in each case when Gauss-Siedel Algorithm was implemented to analyze scaling.

It has to be noted that beyond the 100x100 grid size, Gauss-Siedel solver is not converging even beyond 10,000 i.e the convergence rate is substantially low even though the trend towards convergence can be observed.

| N | M | Processors | Execution Time (s) |
|---|---|---|---|
| 400 | 300 | 1 | 1.61 |
| 400 | 300 | 5 | 0.37 |
| 400 | 300 | 10 | 0.24 |
| 400 | 300 | 15 | 0.16 |
| 400 | 300 | 20 | 0.14 |
| 400 | 300 | 25 | 0.26 |

Table 4: The Problem size,Number of Processors used and the execution times in each case when GMRES Algorithm was implemented to analyze scaling.

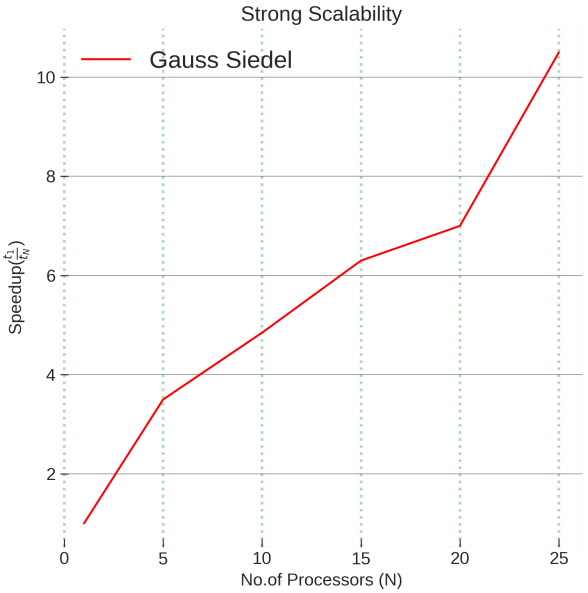


Figure 8: Speed-Up vs No.of Processors when solving the diffusion equation on a 100x100 grid using the Gauss-Siedel Method

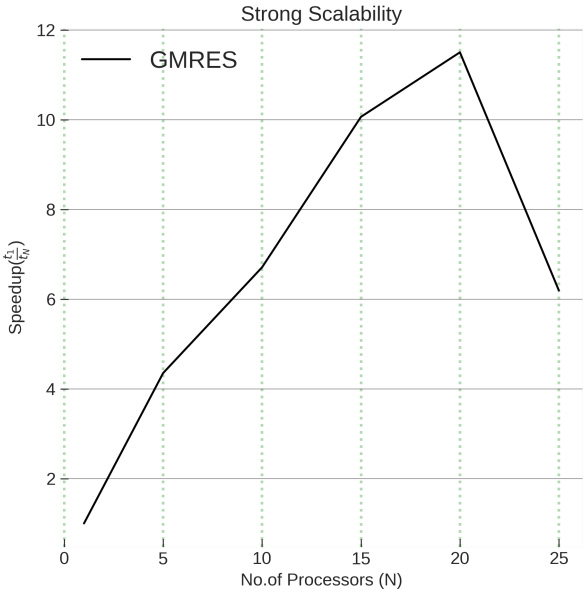


Figure 9: Speed-Up vs No.of Processors when solving the diffusion equation on a 400x300 grid using GMRES Method

It can be observed from both these plots that the highest speed-up acheived by GMRES solver is much higher than that achieved by the Gauss-Siedel solver and this speed-up is also achieved with far fewer CPUs, suggesting that the efficiency of GMRES is far better than Gauss-Siedel.

The drop in speed-up observed in Fig.9 can be explained, as mentioned before, by the architecture of the nodes in the Terra cluster, Each node has two sockets with 14 cores each that share their L3 Cache, the memory that is closest to the cores and can be accessed very quickly, hence, upto, around 14 processors, since the transmission overheads are less since all the cores share the same memory,we observe higher speed-up, however, beyond this, memory needs to be shared and communicated between the two sockets which together make up one node. In case of gauss-siedel iteration, the solution time that each core spends has drastically come down, balancing transmission overheads, resulting in increased speed-up.

Figure.10 and Fig.11 show the variation of the number of iterations it took for the solution to converge with the number of processors/cores involved in the computation, what is observed is that for Gauss-Siedel iteration, irrespective of the number of processors used, the number of iterations settles at a value of around 3550 and flattens out, where as in case of GMRES, there is significant variation that is also consistent with the speed-up trend observed in Fig.9, The no.of iterations required to converge declines when the maximum compute that shared memory was employed and this needs to be explored further by the author.

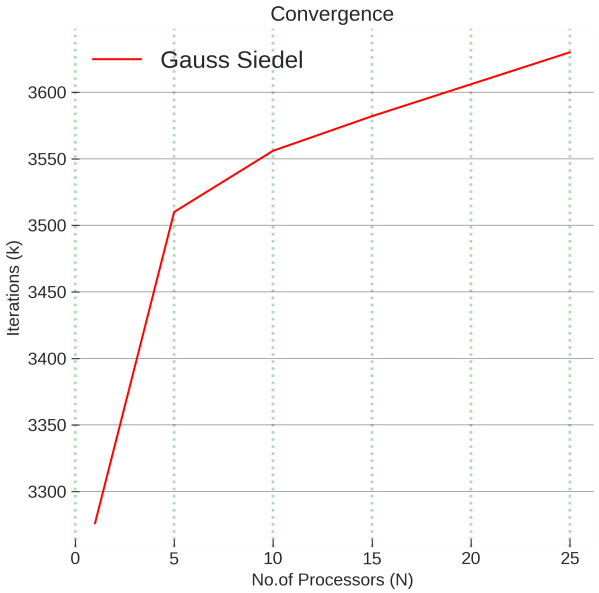


Figure 10: No.Iterations to Convergence Plotted for the Gauss-Siedel solver on a 100x100 grid

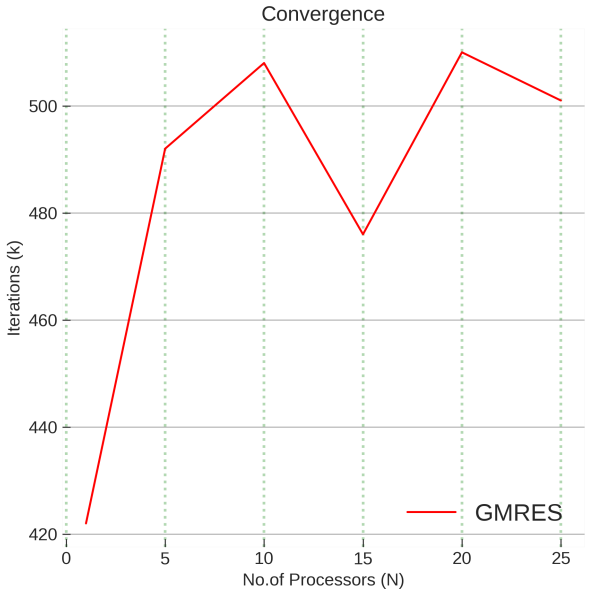


Figure 11: No.Iterations to Convergence Plotted for the GMRES solver on a 400x300 grid

# 4 Summary and Conclusions

The 2-D Diffusion equation was solved successfully over large uniform structured grids using the PETSc library interface with c language, The nature of the 2D Diffusion equation was explored, a 3 step process involving Grid Generation, Finite Difference Equation generation and solution of the thus produced linear system of equations was undertaken.

The grid generated was a structured, uniform grid that spanned the entire domain, it was implemented using the Distributed Array (DMDA) object available as part of the PETSc library, a star stencil ( Five-point stencil) was implemented.

To generate the difference equations, 2nd order Central Differencing scheme was employed, to implement the boundary conditions, the domain was split into 8 different regions of which regions 1,2,3,4 correspond to the west,east,north and south edges respectively, regions 5,6,7,8 correspond to the soth-west,north-east,south-east and north-west corners of the domain respectively.

The system of linear equations of the form Ax=b was obtained through the differencing scheme and the boundary conditions, to solve this, iterative methods were chosen and specifically two methods i.e Gauss-Siedel iteration which belonged to the "primitive" iterative schemes and the Generalized Minimal Residual (GMRES) Method which is a Krylov Subspace method and assumes no prior knowledge of the co-efficient matrix A i.e sparsity, diagonal dominance, postiive definiteness.

To implement both these solvers, PETSc's KSP solver object was utilized, to realize the gauss siedel solver, ksp type had to be set to the richardson solver with SOR preconditioning, which when the relaxation factor $\omega$ is set to unity, is identical to Gauss-Siedel iteration.

The results did show the averaging effect of the diffusion phenomenon and as were expected, a strong scaling analysis was done for both Gauss-Siedel ( on a 100x100 grid) and GMRES (on a 400x300 grid) and it was found that GMRES outperforms Gauss-siedel in terms of maximum speed up as well as efficiency and Gauss-Siedel requires far more ( at least an order of magnitude

higher) iterations to converge for the same problem size.

# References


[1] Satish Balay et al. "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries". In: *Modern Software Tools in Scientific Computing*. Ed. by E. Arge, A. M. Bruaset, and H. P. Langtangen. Birkhäuser Press, 1997, pp. 163–202.

[2] Satish Balay et al. *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.15. Argonne National Laboratory, 2021. URL: https://www.mcs.anl.gov/petsc.

[3] Satish Balay et al. *PETSc Web page*. https://www.mcs.anl.gov/petsc. 2021. URL: https://www.mcs.anl.gov/petsc.

[4] Hadrien Courtecuisse and Jérémie Allard. "Parallel dense gauss-seidel algorithm on many-core processors". In: *2009 11th IEEE International Conference on High Performance Computing and Communications*. IEEE. 2009, pp. 139–147.

[5] John L. Gustafson. "Amdahl's Law". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 53–60. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_77. URL: https://doi.org/10.1007/978-0-387-09766-4_77.

[6] Youcef Saad and Martin H Schultz. "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems". In: *SIAM Journal on scientific and statistical computing* 7.3 (1986), pp. 856–869.

# Appendices

# A Code and Software Development

The entire project was written in C,Python (Post Processing) as well as bash scripts and SLURM Job files on the Terra cluster.

PETSc library along with MPI were used on the C programs, numpy, system,os and matplotlib libraries were utilized to develop the post processing code in Python3.

## A.1 File System

The main directory is **/scratch/vishalkandala/PETSC/2DIFF/** which contains the following sub-directories and files:

- **src** which in-turn contains the **libs** directory inside which are the main source files **2diff.c**, the **makefile** and the python post-processing file **post**
- **build** which contains the executable **2diff.exe** which the make file is **src/libs/** would remove and replace whenever make is called.
- **data** which contains the output files i.e the solution output from the solvers as wel as the **timegmres.csv** and **timegs.csv** which are updated every time a new run is conducted and the execution time, iterations to convergence,grid sizes (along x and y) and the no.of processors used are all appended.
- **logs** is a directory that contains output files from solves that are of three different kinds namely, **dm.txt** files which contain DM information for that run, **profile.txt** that had all the profiling information related to the run and **res.txt** which had the residual output after every iteration.
- **batchproc** contains two directories **jobs** which houses the various SBATCH job files used while the **outputs** directory houses the output files generated when a job is done/cancelled/exits with an error.
- **plts** is a folder that contains all the plots including the contour plots as well as the scaling and iteration plots.
- **sweep** is an executable that deletes all data files, all plots and all the output files.
- **cleanmake** is an executable that changes directory to src/libs/ and calls make, while replacing the existing executable in the build directory
- **singlerun** is an executable that takes in 6 arguments, namely, number of processors, x-grid size, y-grid size,contour plot choice (0 means do not plot contour, 1 means plot and produce an eps file and 2 means plot and produce a png file.) and the speed-up plot choice.
- **multipost** is an executable that loads the matplotlib module and passes the x,y grid values as well as the choices for plot and speed-up plot to the python program post.
- **batchrun** is an executable that changes directory to /batchproc/outputs and submits a job request using the lsf file in /batch/jobs/
- **strongscale** is an executable that runs the cases discussed in this project on various number of processors and stores/creates data in the /data/ directory.

## A.2 Listings

```
//********************* 2D Diffusion Solver *****************
//Vishal Indivar Kandala | 6/25/21
//*********************************************************
//Finite Difference method: Central Differencing is implemented
//*********************************************************
// DMDA Data structure available in PETSc is utilized.
// *********************************************************
//Domain: L =0.4; W=0.3     |Stencil: Star
//     ____T3____   |
//   T1=313   |    |   |    T(i,j+1)
//   T2=273   |____|_W_|   |           |
//   T3=283   T2   L   T4  |           |
//   T4=273   |    |   |   | T(i-1,j)--- T(i,j)---T(i+1,j)
//     |___T1___|   |       |
//           |       |
//           |    T(i,j-1)
// ********************************************************************
//       T_xx+T_yy=0
// ********************************************************************
//    2*(F_x+F_y)*T(i,j)-(F_x*T(i+1,j))-(F_x*T(i-1,j))-(F_y*T(j,j+1))-(F_x*T(i,j-1)
    )
// ********************************************************************
// ********************************************************************
//
#include<petsc.h>
#include<stdlib.h>
#include<stdio.h>
//*************************************************************************
//Function to create the RHS vector b
PetscErrorCode FormRHS(DM da,Vec b)
{
  int i,j;

  double hx, hy, BC[4]={313.0,273.0,283.0,273.0}, one = 1.0;

  double **ab;

  DMDALocalInfo info;

  DMDAGetLocalInfo(da,&info);

  DMDAVecGetArray(da,b,&ab);

  hx=0.4/(info.mx-1);

  hy=0.3/(info.my-1);



  //BC[0]=313.0; BC[1]=273.0; BC[2]=283.0; BC[3]=273.0;

  for  (j=info.ys;j<(info.ys+info.ym);j++)
  {

    for (i=info.xs;i<(info.xs+info.xm);i++)

    {

      ab[j][i]=0;

      if(i == 0)
      {
        if(j == 0)
        {
          ab[j][i]=(BC[0]+BC[1])/2; // Numerical Decoration.
        }
        if(j == info.my-1)
        {
```

```
68            ab[j][i]=(BC[0]+BC[2])/2;
69          }
70          if (j>0 && j<info.my-1)
71          {
72            ab[j][i]=BC[1];
73          }
74        }
75        if (i == 1)
76        {
77          ab[j][i]=ab[j][i]+BC[1]/(hx*hx);
78          if(j == 0)
79          {
80            ab[j][i]=BC[0];
81          }
82          if ( j == 1)
83          {
84            ab[j][i]=ab[j][i]+(BC[0]/(hy*hy));
85          }
86          if (j == info.my-2)
87          {
88            ab[j][i]=ab[j][i]+(BC[2]/(hy*hy));
89          }
90          if (j == info.my-1)
91          {
92            ab[j][i]=BC[2];
93          }
94        }
95        if (i>1 && i<info.mx-2)
96        {
97          if(j == 0)
98          {
99            ab[j][i]=BC[0];
100         }
101         if (j == info.my-1)
102         {
103           ab[j][i]=BC[2];
104         }
105         if ( j == 1)
106         {
107           ab[j][i]=ab[j][i]+BC[0]/(hy*hy);
108         }
109         if (j == info.my-2)
110         {
111           ab[j][i]=ab[j][i]+BC[2]/(hy*hy);
112         }
113         if (j>1 && j<info.my-2)
114         {
115           ab[j][i]=0;
116         }
117       }
118       if (i == info.mx-2)
119       {
120         ab[j][i]=BC[3]/(hx*hx);
121
122         if(j == 0)
123         {
124           ab[j][i]=BC[0];
125         }
126         if ( j == info.my-1)
127         {
128           ab[j][i]=BC[2];
129         }
130         if ( j == 1)
131         {
132           ab[j][i]=ab[j][i]+(BC[0]/(hy*hy));
133         }
134         if (j == info.my-2)
135         {
136           ab[j][i]=ab[j][i]+(BC[2]/(hy*hy));
137         }
```

```
      }
      if (i == info.mx-1)
      {
        if(j == 0)
        {
          ab[j][i]=ab[j][i]+(BC[3]+BC[0])/2;
        }
        if(j == info.my-1)
        {

          ab[j][i]=ab[j][i]+(BC[3]+BC[2])/2;
        }
        if (j> 0 && j< info.my-1)
        {
          ab[j][i]=ab[j][i]+BC[3];
        }
      }  // End of If-Else clause

    } //End of y-loop
  }//End of x-loop

  DMDAVecRestoreArray(da,b,&ab);

  return 0;
}
//**********************************************************************
//Function to Create the matrix A
//**********************************************************************
PetscErrorCode FormMat(DM da, Mat A)
{
  DMDALocalInfo info;

  int i,j,ncols;

  double hx,hy,v[5];

  MatStencil row,col[5]; // links the da and matrix A

  DMDAGetLocalInfo(da,&info);

  hx=0.4/(info.mx-1);

  hy=0.3/(info.my-1);

  // The (m*n)*(m*n) A matrix is solved to find the rasterized (m*n)*(1) vector
    solution in PETSc as the solution cannot be stored as a matrix in PETSc.

  for (i=info.xs;i<(info.xs+info.xm);i++)
  {
    for (j=info.ys;j<(info.ys+info.ym);j++)
    {

      row.j = j; row.i = i; // The i*j th row in the (m*n)*(m*n) A matrix is
    selected.

// No.of entries to be made in each row of A matrix col[0].j = j; col[0].i =i; //
    Using the matrix Stencil Structure, we can place the values in v at the
    location (row.i*row.j),(col.i*col.j) so we are selecting the indices and
    placing each value.
      ncols = 1;

      col[0].i=i; col[0].j=j;

      v[0] = 2*((1/(hx*hx))+(1/(hy*hy)));

      // Addressing Boundaries

      if(i>1)
      {
        col[ncols].i =i-1; col[ncols].j = j;
```

```
        v[ncols] = -1/(hx*hx);
        ncols=ncols+1;
      }
      if(j>1)
      {
        col[ncols].i=i; col[ncols].j=j-1;
        v[ncols]=-1/(hy*hy);
        ncols=ncols+1;
      }
      if(i<info.mx-2)
      {
        col[ncols].i=i+1; col[ncols].j=j;
        v[ncols]=-1/(hx*hx);
        ncols=ncols+1;
      }
      if(j<info.my-2)
      {
        col[ncols].i=i;col[ncols].j=j+1;
        v[ncols]=-1/(hy*hy);
        ncols=ncols+1;
      }
      if (i == 0 || j == 0 || i == info.mx-1 || j == info.my-1)
      {
        ncols=1;
        v[0]=1;

      } // End of if-else clause


      MatSetValuesStencil(A,1,&row,ncols,col,v,INSERT_VALUES); // Enter the colums
    (the entire row) into the matrix A associated with da.

    } // End of y-loop

  } // End of x-loop

  MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);

  MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);

  return 0;

}
//************************************************************
//Function to write the solution to a CSV file
//************************************************************
PetscErrorCode WriteSol(DM da, Vec u)
{
  int i;

  int j;

  double **au,v;

  FILE *fid;

  const char * dir = "../data/";

  const char * fname = "2diff";

  const char * ftype = ".csv";

  char name_buffer[4096];

  DMDALocalInfo info;

  DMDAGetLocalInfo(da, &info);

  DMDAVecGetArray(da,u,&au);

```

```
  sprintf(name_buffer,"%s%s-%dx%d%s",dir,fname,(int)info.mx,(int)info.my,ftype);

  fid=fopen(name_buffer,"w+");

  for(j=info.ys;j<(info.ys+info.ym);j++)
  {
    for(i=info.xs;i<(info.xs+info.xm);i++)
    {
      v=au[j][i];
      if(i==info.mx-1)
      {
        fprintf(fid,"%.2lf",v);
      }
      else
      {
        fprintf(fid,"%.2lf,",v);
      }
    }
    fprintf(fid,"\n");
  }

  DMDAVecRestoreArray(da,u,&au);

  fclose(fid);


  return 0;
}

//**********************************************************
//Main
//**********************************************************

int main(int argc, char **argv)
{
  int flg1,flg2,flg3,its,n,m,p1,p2,p3;

  const char * ksp_type;

  PetscErrorCode ierr;

  PetscLogDouble duration,start, stop;

  DM da; //Distributed Member (or ) Data Management object. (Used for a structured
    grid).

  Mat A;

  FILE *fidtime;

  const char * dir = "../data/";

  const char * timefilegs = "timegs";

  const char * timefilegmres = "timegmres";

  const char * timeftype = ".csv";

  char name_buffer[1024];

  Vec b,u;

//  Vec useq;

//  VecScatter ctx;

  KSP ksp;

  DMDALocalInfo info; // Data Structure to hold information about the array

```

```
341   //n=atoi(argv[1]); m=atoi(argv[2]);
342
343   PetscInitialize(&argc,&argv,NULL,"Solve Poisson Equation in 2D");
344
345   ierr=PetscTime(&start); CHKERRQ(ierr);
346   //*****************************************************
347   // Creating the DMDA Data Structure and declaring it over the matrix A
348   //*****************************************************
349   ierr=DMDACreate2d(PETSC_COMM_WORLD,DM_BOUNDARY_NONE,DM_BOUNDARY_NONE,
        DMDA_STENCIL_STAR,2000,2000,PETSC_DECIDE,PETSC_DECIDE,1,1,NULL,NULL,&da);
        CHKERRQ(ierr);
350
351   ierr=DMSetFromOptions(da);CHKERRQ(ierr);
352
353   ierr=DMSetUp(da);CHKERRQ(ierr);
354
355   ierr=DMCreateMatrix(da,&A); CHKERRQ(ierr);// The size of A is determined by grid
        size specified in DMDACreate
356
357   ierr=MatSetFromOptions(A);CHKERRQ(ierr);
358
359   //*************************************************
360   // Creating the vectors b and u from DMDA object
361   // ************************************************
362
363   ierr=DMCreateGlobalVector(da,&b);CHKERRQ(ierr);
364
365   ierr=VecSetFromOptions(b);CHKERRQ(ierr);
366
367   ierr=VecDuplicate(b,&u); CHKERRQ(ierr);// Initializing u
368
369   //*******************************************
370   //Form Matrix A
371   //*******************************************
372   ierr=FormMat(da,A); CHKERRQ(ierr);
373
374   //*******************************************
375   //Form vector b
376   //*******************************************
377   ierr=FormRHS(da,b); CHKERRQ(ierr);
378
379 //  PetscPrintf(PETSC_COMM_WORLD, "***** b *********** \n");
380
381 //  VecView(b,PETSC_VIEWER_STDOUT_WORLD);
382   //*****************************************
383   //Solve the system Ax=b
384   //*****************************************
385   ierr=KSPCreate(PETSC_COMM_WORLD,&ksp); CHKERRQ(ierr);
386
387   ierr=KSPSetOperators(ksp,A,A); CHKERRQ(ierr);
388
389   ierr=KSPSetFromOptions(ksp); CHKERRQ(ierr);
390
391   ierr=DMDAGetLocalInfo(da,&info);
392
393   ierr=KSPSolve(ksp,b,u); CHKERRQ(ierr);
394
395   ierr=KSPGetTotalIterations(ksp,&its); CHKERRQ(ierr);
396
397   ierr=KSPGetType(ksp,&ksp_type);
398
399   ierr=PetscTime(&stop); CHKERRQ(ierr);
400
401   duration=stop-start;
402
403   flg1=strcmp(ksp_type,"richardson");
404
405   flg2=strcmp(ksp_type,"preonly");
406
407   flg3=strcmp(ksp_type,"gmres");
```

```
408
409   if(flg1==0 || flg2==0)
410   {
411     sprintf(name_buffer,"%s%s%s",dir,timefilegs,timeftype);
412   }
413   else if(flg3==0)
414   {
415     sprintf(name_buffer,"%s%s%s",dir,timefilegmres,timeftype);
416   }
417
418   fidtime=fopen(name_buffer,"a");
419
420   ierr=DMDAGetInfo(da,NULL,NULL,NULL,NULL,&p2,&p1,&p3,NULL,NULL,NULL,NULL,NULL,NULL
      );CHKERRQ(ierr);
421
422   ierr=PetscFPrintf(PETSC_COMM_WORLD,fidtime,"%d,%d,%d,%d,%.2lf\n",its,p2+p1+p3,
      info.mx,info.my,duration);
423
424 //  ********************************************
425 //  Print the solution out to a file
426 //  ********************************************
427
428 //  ierr=VecScatterCreateToZero(u,&ctx,&useq); CHKERRQ(ierr);
429
430 //  ierr=VecScatterBegin(ctx,u,useq,INSERT_VALUES,SCATTER_FORWARD);
431
432 //  ierr=VecScatterEnd(ctx,u,useq,INSERT_VALUES,SCATTER_FORWARD);
433
434   ierr=WriteSol(da,u); CHKERRQ(ierr);
435
436   //***************************************
437   // Memory De-allocation for vectors u,uexact,b, matrix A, solver object ksp and
      DM Data Structure da
438   // ***************************************
439   ierr=VecDestroy(&u);CHKERRQ(ierr);
440
441 //  ierr=VecScatterDestroy(&ctx);CHKERRQ(ierr);
442
443 //  ierr=VecDestroy(&useq);CHKERRQ(ierr);
444
445 //  ierr=VecDestroy(&useq);CHKERRQ(ierr);
446
447   ierr=VecDestroy(&b);CHKERRQ(ierr);
448
449   ierr=MatDestroy(&A);CHKERRQ(ierr);
450
451   ierr=KSPDestroy(&ksp);CHKERRQ(ierr);
452
453   ierr=DMDestroy(&da);CHKERRQ(ierr);
454
455   return PetscFinalize();
456 }
```

Listing 1: The core algorithm written using PETSc: **2diff.c**

```
include ${PETSC_DIR}/lib/petsc/conf/variables
include ${PETSC_DIR}/lib/petsc/conf/rules

CFLAGS = -Wall -Werror -g -O0

.PHONY: clean

all:: clean build

build: 2diff.o chkopts
  echo "******* 2D Diffusion Solver Make begins **********"
  -${CLINKER} ${CFLAGS} -o ../../build/2diff.exe 2diff.o ${PETSC_LIB}
  ${RM} 2diff.o
  echo "Make complete: Executable can be found at ../../build/"

clean::
  -${RM} edit  ../../build/2diff.exe
  echo "clean completed"
```

Listing 2: The **makefile**

```
#!/usr/bin/bash

ml purge

echo "Loading PETSc-3.8.3-intel-2017A-Python-2.7.12"

ml load PETSc/3.8.3-intel-2017A-Python-2.7.12

echo "Load Succesful"

cd $PWD/src/libs/

echo "make running"

make all

echo "make succesfull"
```

Listing 3: The **cleanmake** script

```
#!/usr/bin/bash

cd $PWD/batchproc/outputs/

sbatch ../jobs/2diff.slurm
```

Listing 4: The **batchrun** script

```
#!/usr/bin/bash

echo "Sweeping data"

cd $PWD/data/

rm *.csv

echo "Data Sweeped"

echo "Sweeping Plots"

cd ../plts/

rm *.png

rm *.eps

echo  "Plots Sweeped"

echo "Sweeping Batch outputs"
```

```

cd ../batchproc/outputs/

rm *

echo "Batch outputs Sweeped"

cd ../../
```

Listing 5: The **sweep** script

```
#!/usr/bin/bash

ml purge

ml load matplotlib

echo "Changing directory to src/libs/"

cd $PWD/src/libs/

echo "running post Processing"

./post $1 $2 $3 $4 $5

echo "Post Processing successfull"
```

Listing 6: The **multipost** script

```
#!/usr/bin/bash

echo "changing to build directory"

cd $PWD/build/

ml purge

echo "loading PETSc/3.8.3-intel-2017A-Python-2.7.12  Libary"

ml load PETSc/3.8.3-intel-2017A-Python-2.7.12

echo " Load Succesful"

echo "cleanng logs"

cd ../logs/

rm *

echo "logs cleaned"

cd ../build/

echo " Running Simulation"

mpirun -n $1 ./2diff.exe -da_grid_x $2 -da_grid_y $3 -dm_view :../logs/dm.txt -
    log_view :../logs/profile.txt  -ksp_monitor :../logs/res.txt -
    ksp_initial_guess_nonzero 1  -ksp_type richardson -pc_type sor -pc_sor_omega 1
    -ksp_rtol 1e-5

#mpirun -n $1 ./2diff.exe -da_grid_x $2 -da_grid_y $3 -dm_view :../logs/dm.txt -
    log_view :../logs/profile.txt  -ksp_monitor :../logs/res.txt -ksp_type
    richardson -pc_type sor -pc_sor_omega 1

echo " Simulation succesful, data can be found in /data/"

echo " Loading matplotlib"

ml purge

ml load matplotlib

echo "changing to src/libs/ directory"

cd ../src/libs/

echo "Runing Post Processing"

./post $2 $3 $4 $5

echo " Post Processing Succesful, plot can be found in /plts/"
```

Listing 7: The **singlerun** script

```
#!/sw/eb/sw/Python/3.8.6-GCCcore-10.2.0/bin/python
from numpy import *
from matplotlib.pyplot import *
import sys
from os import *
##########
##########
def speedup(tarray):
  sp=ones(len(tarray))
  for i in range(len(tarray)):
    sp[i]=tarray[-1,4]/tarray[i,4]
  return sp

#########
#########
vis=1
lw=0.3
fs=13
ts=18
style.use('seaborn-ticks')
rcParams['axes.linewidth']=0.01
rcParams['axes.prop_cycle']=cycler(color=["r", "#e94cdc", "0.1"])
rcParams['font.size']=fs
rcParams['legend.fontsize']=ts
c=int(sys.argv[3])
sc=int(sys.argv[4])
if (sc==1 or sc==2):
  fl1,flg2=0,0;
  tgresfile="../../data/timegmres.csv"
  tgsfile="../../data/timegs.csv"
  if (path.exists(tgresfile)):
    flg1=1
    tgres=loadtxt(tgresfile,delimiter=",")
    sgres=speedup(tgres)
  if (path.exists(tgsfile)):
    flg2=1
    tgs=loadtxt(tgsfile,delimiter=",")
    sgs=speedup(tgs)

  figdir="../../plts/"
  figname="speedup0"
  figtypes=[".eps",".png"]
  figfile=figdir+figname+figtypes[c-1]
  fig,ax=subplots(figsize=(7,7))
  ax.set(title="Strong Scalability",xlabel="No.of Processors (N)",ylabel="Speedup($
     \\frac{t_{1}}{t_{N}} $)")
  ax.plot(tgres[:,1],sgres,color='k',label="GMRES")
# ax.plot(tgs[:,1],sgs,color='r',label="Gauss Siedel")
  ax.legend()
  ax.tick_params(which='both',direction='in')
  ax.grid(axis="x",color="green",alpha=.3,linewidth=2,linestyle=":")
  ax.grid(axis="y",color="black",alpha=.5,linewidth=.5)
  fig.savefig(figfile,dpi=300)

  figdir="../../plts/"
  figname="speedup1"
  figtypes=[".eps",".png"]
  figfile=figdir+figname+figtypes[c-1]
  fig,ax=subplots(figsize=(7,7))
  ax.set(title="Strong Scalability",xlabel="No.of Processors (N)",ylabel="Speedup($
     \\frac{t_{1}}{t_{N}} $)")
# ax.plot(tgres[:,1],sgres,color='k',label="GMRES")
  ax.plot(tgs[:,1],sgs,color='r',label="Gauss Siedel")
  ax.legend()
  ax.tick_params(which='both',direction='in')
  ax.grid(axis="x",color="green",alpha=.3,linewidth=2,linestyle=":")
  ax.grid(axis="y",color="black",alpha=.5,linewidth=.5)
  ax.grid(True,which='both')
  fig.savefig(figfile,dpi=300)

```

```
  figdir="../../plts/"
  figname="iter1"
  figtypes=[".eps",".png"]
  figfile=figdir+figname+figtypes[c-1]
  fig,ax=subplots(figsize=(7,7))
  ax.set(title="Convergence",xlabel="No.of Processors (N)",ylabel="Iterations (k)")
# ax.plot(tgres[:,1],sgres,color='k',label="GMRES")
  ax.plot(tgs[:,1],tgs[:,0],color='r',label="Gauss Siedel")
  ax.legend()
  ax.tick_params(which='both',direction='in')
  ax.grid(axis="x",color="green",alpha=.3,linewidth=2,linestyle=":")
  ax.grid(axis="y",color="black",alpha=.5,linewidth=.5)
  fig.savefig(figfile,dpi=300)


  figdir="../../plts/"
  figname="iter0"
  figtypes=[".eps",".png"]
  figfile=figdir+figname+figtypes[c-1]
  fig,ax=subplots(figsize=(7,7))
  ax.set(title="Convergence",xlabel="No.of Processors (N)",ylabel="Iterations (k)")
# ax.plot(tgres[:,1],sgres,color='k',label="GMRES")
  ax.plot(tgres[:,1],tgres[:,0],color='r',label="GMRES")
  ax.legend()
  ax.tick_params(which='both',direction='in')
  ax.grid(axis="x",color="green",alpha=.3,linewidth=2,linestyle=":")
  ax.grid(axis="y",color="black",alpha=.5,linewidth=.5)
  fig.savefig(figfile,dpi=300)
#########
if (c==1 or c==2):
  fdir="../../data/"
  fname="2diff"
  n=int(sys.argv[1])
  m=int(sys.argv[2])
  ftype=".csv"
  ffile=fdir+fname+"-"+str(n)+"x"+str(m)+ftype
  data=genfromtxt(ffile,delimiter=",")
  L=0.4
  W=0.3
  x=linspace(0,L,num=n)
  y=linspace(0,W,num=m)
  xx,yy=meshgrid(x,y)

  figdir="../../plts/"
  figname="tcontour"
  figtypes=[".eps",".png"]

  fig,ax=subplots(figsize=(16,12))
  ax.set(title="T (K)", xlabel="x (m)", ylabel="y (m)")
  tcontour=contourf(xx,yy,data,cmap='hot')
  fig.colorbar(tcontour)
  figfile=figdir+figname+"-"+str(n)+"x"+str(m)+figtypes[c-1]
  fig.savefig(figfile,dpi=300)
```

Listing 8: The post processing script utilizing matplotlib written in Python:**post**

```
#!/bin/sh
#ENVIRONMENT SETTINGS; CHANGE WITH CAUTION
#SBATCH --export=NONE                #Do not propagate environment
#SBATCH --get-user-env=L             #Replicate login environment

##NECESSARY JOB SPECIFICATIONS
#SBATCH --job-name=2diff       #Set the job name to "Example1"
#SBATCH --time=0-03:00:00          #Set the wall clock limit to 0 Days and 5hrs
#SBATCH --ntasks=25                  #Request 50 tasks
#SBATCH --ntasks-per-node=25         #Request 245tasks/cores per node
#SBATCH --mem=4G                   #Request 4096MB (4GB) per node
#SBATCH --output=2diff.%j      #Send stdout/err to "Example3Out.[jobID]"

##OPTIONAL JOB SPECIFICATIONS
#SBATCH --account=122759699519             #Set billing account
#SBATCH --mail-type=ALL              #Send email on all job events
#SBATCH --mail-user=vishalkandala@tamu.edu    #Send all emails to email_address

cd /scratch/user/vishalkandala/PETSC/2DIFF/

./strongscale
```

Listing 9: The jobfile to run on the Terra cluster: **2diff.slurm**

```
#!/usr/bin/bash

echo "changing to build directory"

cd $PWD/build/

ml purge

echo "loading PETSc/3.8.3-intel-2017A-Python-2.7.12  Libary"

ml load PETSc/3.8.3-intel-2017A-Python-2.7.12

echo " Load Succesful"

echo "cleanng logs"

cd ../logs/

rm *

echo "logs cleaned"

cd ../build/

echo " Running Simulation"

mpirun -n 25 ./2diff.exe -da_grid_x 400  -da_grid_y 300  -dm_view :../logs/dm25.txt
     -log_view :../logs/profile25.txt  -ksp_monitor :../logs/res25.txt

mpirun -n 25 ./2diff.exe -da_grid_x 100  -da_grid_y 100  -dm_view :../logs/dm25gs.
    txt -log_view :../logs/profile1gs.txt  -ksp_monitor :../logs/res25gs.txt -
    ksp_type richardson -pc_type sor -pc_sor_omega 1 -ksp_initial_guess_nonzero 1

mpirun -n 20 ./2diff.exe -da_grid_x 400  -da_grid_y 300  -dm_view :../logs/dm20.txt
     -log_view :../logs/profile20.txt  -ksp_monitor :../logs/res20.txt

mpirun -n 20 ./2diff.exe -da_grid_x 100  -da_grid_y 100  -dm_view :../logs/dm20gs.
    txt -log_view :../logs/profile20gs.txt  -ksp_monitor :../logs/res20gs.txt -
    ksp_type richardson -pc_type sor -pc_sor_omega 1 -ksp_initial_guess_nonzero 1

mpirun -n 15 ./2diff.exe -da_grid_x 400  -da_grid_y 300  -dm_view :../logs/dm.txt -
    log_view :../logs/profile.txt  -ksp_monitor :../logs/res.txt

mpirun -n 15 ./2diff.exe -da_grid_x 100  -da_grid_y 100  -dm_view :../logs/dm15gs.
    txt -log_view :../logs/profile15gs.txt  -ksp_monitor :../logs/res15gs.txt -
    ksp_type richardson -pc_type sor -pc_sor_omega 1 -ksp_initial_guess_nonzero 1
```

```
38
39 mpirun -n 10 ./2diff.exe -da_grid_x 400  -da_grid_y 300  -dm_view :../logs/dm10.txt
       -log_view :../logs/profile10.txt  -ksp_monitor :../logs/res10.txt
40
41 mpirun -n 10 ./2diff.exe -da_grid_x 100  -da_grid_y 100  -dm_view :../logs/dm10gs.
      txt -log_view :../logs/profile10gs.txt  -ksp_monitor :../logs/res10gs.txt -
      ksp_type richardson -pc_type sor -pc_sor_omega 1 -ksp_initial_guess_nonzero 1
42
43 mpirun -n 5 ./2diff.exe -da_grid_x 400  -da_grid_y 300  -dm_view :../logs/dm5.txt -
      log_view :../logs/profile5.txt  -ksp_monitor :../logs/res5.txt
44
45 mpirun -n 5 ./2diff.exe -da_grid_x 100  -da_grid_y 100  -dm_view :../logs/dm5gs.txt
       -log_view :../logs/profile5gs.txt  -ksp_monitor :../logs/res5gs.txt -ksp_type
      richardson -pc_type sor -pc_sor_omega 1 -ksp_initial_guess_nonzero 1
46
47 mpirun -n 1 ./2diff.exe -da_grid_x 400  -da_grid_y 300  -dm_view :../logs/dm1.txt -
      log_view :../logs/profile1.txt  -ksp_monitor :../logs/res1.txt
48
49 mpirun -n 1 ./2diff.exe -da_grid_x 100  -da_grid_y 100  -dm_view :../logs/dm1gs.txt
       -log_view :../logs/profile1gs.txt  -ksp_initial_guess_nonzero -ksp_monitor
      :../logs/res1gs.txt -ksp_type richardson -pc_type sor -pc_sor_omega 1 -
      ksp_initial_guess_nonzero 1
50
51 echo " Simulation succesful, data can be found at 2DIFF/data/ "
```

Listing 10: The **strongscale** script

# Listings