Project Report

on

Study of Parallel Finite Difference solver using AMReX framework

Presented by

Sanket Thakare	PRN : 220940141030
Raju Ranjan	PRN : 220940141022
Shankar Lal Jatav	PRN : 220940141027
Pankaj Gharde	PRN : 220940141015

Guided by

Dr. V. V. Shenoi

Joint Director HPC - Infrastructure & Ecosystem

C-DAC, Pune

Centre for Development of Advanced Computing (C-DAC), Pune

Chapter 1

Introduction

1.1 AMReX Framework

AMReX is a software framework containing all the functionality to write massively parallel, block-structured adaptive mesh refinement (AMR) applications.

AMReX is developed at LBNL, NREL, and ANL as part of the Block-Structured AMR Co-Design Center in DOE's Exascale Computing Project.

AMReX supports hierarchical parallelism through MPI+X (+X)

- Multi-core: "MPI over grids, OpenMP over tiles"
- Hybrid w/ GPUs: "MPI over grids, CUDA (or HIP or DPC++) on GPUs"
- Written in C++ (minimum requirement = C++14; can use features of C++17 if available)
- Many GPU-specific optimizations - memory Arenas, fused kernel launches, etc...

Ref. AMReX Documentation

1.2 Mathematical modeling of physical systems

The mathematical model of problems in science and engineering is a system of differential equations either ordinary or partial differential equations involving the variables in the problem. It involves identification of the relevant variables, known as model parameters. The model in turn is used to simulate the actual processes leading to the physical phenomena in a computer experiment. Through simulations, one can obtain the observable of the experimental system described by the model. This is followed by validation and refinement of the model on comparison with the experimental data. The time required for simulation of realistic model increases drastically as one includes several parameters to refine the model to obtain better results closer to the experimental observation. With the advent of better computer hardware and software it is now possible to simulate the model within a reasonable time.

1.3 HPC for Scientific Applications

High Performance Computing (HPC) has become an integral part of simulation science due to the following:

- Want to get the work done as fast as possible (greater clock speed)
- More resources, compute as well as data storage (for an application) multi core, SMP, distributed system (clusters), super computers
- Cost effective (emergence of cluster computing)
- Problems mandate it [Too large, complex, expensive and/or dangerous to do other way], Climate / Weather Modeling, Data intensive problems (data mining, oil reservoir simulation), Problems with large length and time scales (cosmology)

This has led to scientific research becoming a community effort, paving way for centrally managed computing infrastructure, development of community software and training of the manpower.

1.4 Partial Differential Equation

Partial differential equation (PDE) solvers are employed by a large fraction of scientific applications in diverse areas such as heat diffusion, electromagnetics, and fluid dynamics. The PDE solvers are a class of iterative kernels which update array elements according to some fixed pattern, called stencil. The form of the differential operator is reflected in the stencil used for the computations. PDE solvers based on finite- difference methods are known as stencil computations in scientific computing literature. Application of PDE is mainly s follow:

- Elliptic - Elliptic equations generally arise from physical problem (system) moving towards steady state that involves a diffusion process and has reached equilibrium, steady state temperature distribution. Examples -Laplace equation and Poisson equation.
- Parabolic - Parabolic PDEs tend to arise in systems evolving in space and time such as time dependent diffusion problems such as transient flow of heat. Example - Diffusion equation.
- Hyperbolic - Hyperbolic PDEs deal with systems such as wave propagation in a medium characterized by second order derivatives w.r.t time. Example - Wave equation.

1.4.1 Laplace Equation

Laplace equation is a second-order partial differential equation (PDE) that appears in many areas of science and engineering. Laplace Equation arises as a steady state problem for the heat equation that do not vary with time where Δ or ∇2 is the Laplacian operator which is generate stencils and steady state solution satisfies two things first is U is such that it obeys the equation in the given domain and second one is boundary condition u is prescribed on domain D.

Following is the Laplace's equation in 2D

$$U(x, y) = 0$$

$$\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y} = 0 \qquad \text{or} \quad \nabla^2 u = 0$$

1.4.2 Boundary Conditions

The Dirichlet Boundary value problem for the Laplacian on a rectangular domain specifies the value of the function on the surface. Values at the boundaries (Top, Bottom, Left, Right) are known. Boundary conditions are U(x, 0) = f1(x), U(x, H) = f2(x), U(0, y) = g1(y) and U(L, y) = g2(y) For Example - In Figure 1.1 domain D is given (0, L) × (0, H) as (0, 1) × (0, 1) and boundary values are top boundary is U(x, 1) = sin(2πx), Bottom is U(x, 0) = 0, Left is U(0, y) = 0 and right is U(1, y) = −sin(πy).

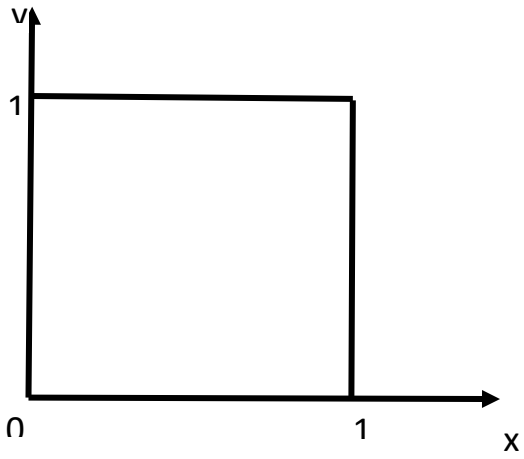


Fig 1.1: Domain

1.4.3 Finite Difference Methods

The finite-difference methods (FDM) are numerical methods for solving differential equations by approximating them with difference equations, in which finite differences approximate the derivatives. This discretization method is among the widely used approach to numerical solutions of partial differential equations.

For a continuous function f(x) in the interval [x0, xN+1], where xi = iΔx, with Δx = 1/(N+1), the derivative of f(x) with respective x is $f'(x)$ defined as:

$$f'(\alpha) = \lim_{x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

The first derivative of $f(x)$ at x = x_i, $f'(x_i)$:

$$f'(x) = \frac{f(x_i) - f(x_i - 1)}{x_i - x_i - 1} = \frac{f(x_i) - f(x_i - 1)}{\Delta x}$$

The second derivative of $f(x)$ at $x = x_i$, $f''(x_i)$

$$f''(x_i) = \frac{f'(x_{i+1}) - f'(x_i)}{\Delta x} = \frac{\frac{f(x_{i+1}) - f(x_i)}{\Delta x} - \frac{f(x_i) - f(x_{i-1})}{\Delta x}}{\Delta x} = \frac{f(x_{i+1}) - 2f(x_i) - f(x_{i-1})}{\Delta x^2}$$

This can be generalized to partial derivatives of function of multi variables.

1.4.4 Discretization

Divide the given domain into discrete counter parts is called discretization that gives grid of size (N + 1)×(N + 1) and U(x, y) = U(iΔx, jΔy) ≡ Ui, j where x ≡ iΔx where Δx = 1/N, y ≡ jΔy where Δy = 1/N and Δx = Δy where 0 < x < 1 and 0 < y < 1. The solution Ui, j is to be obtained over this domain. Figure 1.2 shows domain after discretization where n = 3.

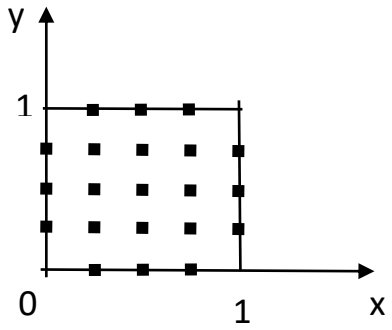


Fig 1.2: Discretize Domain

Approximation of the second partial derivatives in the PDE by a 2nd order centered difference scheme is,

$$\frac{\partial^2 U}{\partial x^2} = \frac{\left(U_{i+1,j} - 2U_{i,j} + U_{i-1,j}\right)}{(\Delta x)^2}$$

$$\frac{\partial^2 U}{\partial y^2} = \frac{\left(U_{i,j+1} - 2U_{i,j} + U_{i,j-1}\right)}{(\Delta y)^2}$$

Rearrange both the equations

$$\frac{\left(U_{i+1,j}-2U_{i,j}+U_{i-1,j}\right)}{(\Delta x)^2}+\frac{\left(U_{i,j+1}-2U_{i,j}+U_{i,j-1}\right)}{(\Delta y)^2}=0$$

Solution of the Laplace equation is obtained as,

$$U_{i,j}=\frac{1}{4}\left(U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1}\right)$$

Thus a 5 -point stencil for computation is formed along with its neighbors including itself for every point in the computation grid.

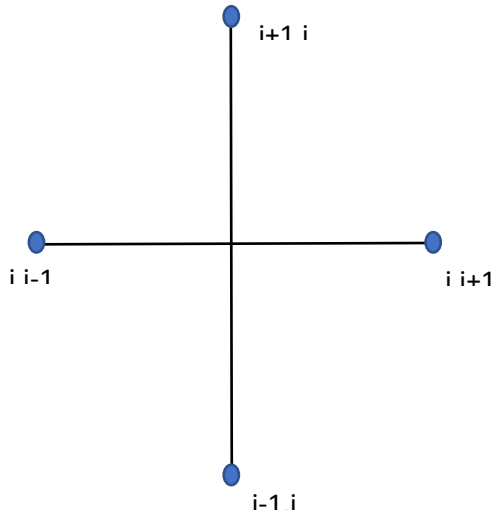


Fig 1.3: 5-point stencil

Algorithm for Laplace equation

1. Choose the structure of grid
2. Apply boundary condition
3. Initialize the computation grid
4. for Iter= Iters to Itere in Δt steps do
5. for all discretized points in computation domain do
6. Stencil computation (update U^t)
7. end for
8. end for

Chapter 2

Methods to solve problem

2.1 Study of Project

This study involves implementation of parallel Gauss Jacobi solver (stencil computation) and performance optimization.

Here we demonstrate **Parallel Implementation of 2D Heat Equation using AMReX Framework.**

2.2 Parallel Jacobi solver

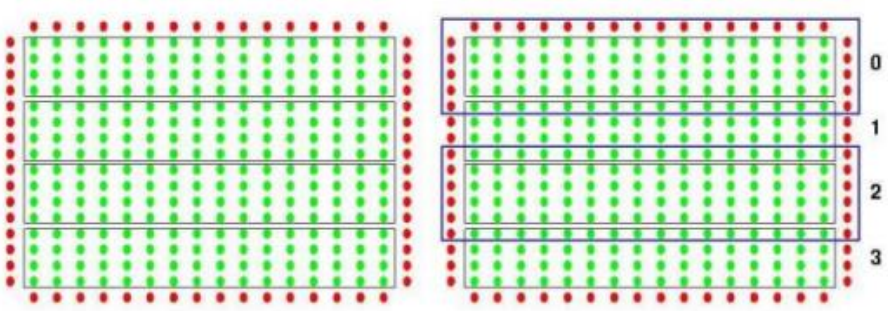


Fig. 2.1: Jacobi Solver (MPI: 1D decomposition)

- Geometric decomposition of the domain (across p processes) 'p' strips.
- Working data for each process includes: **its domain+ neighbor rows/columns** around the domain (ghost regions).
- For computing $(M-1)^2$ / points, require 2 (M− 1) points (along i direction) + 2(M− 1) points (along j direction) known as **halo** region.
- Could be boundary points (fixed)/on the neighbor process (Changes with every iteration).

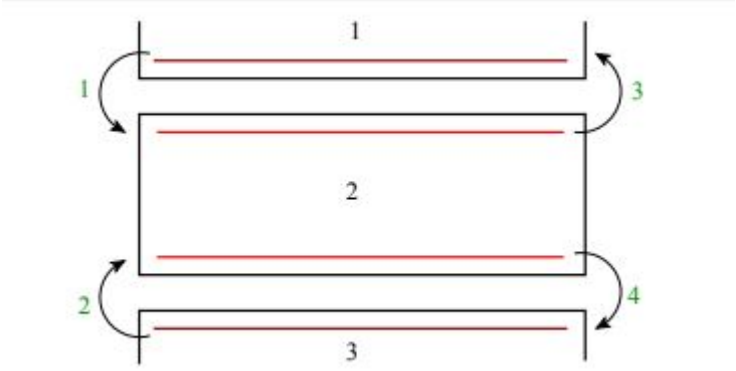


Fig . 2.2: Exchange of neighbors

- To compute u (i, j) on process ‘2’ at points along **red** lines, one needs u (i, j)’s from previous iteration for points along **brown** lines on processes ‘1’ and ‘3’. [1 & 2]
- Similarly, to compute u (i, j) on processes ‘1’ and ‘3’ at points along **brown** lines, one needs u (i, j)’s from previous iteration for points along **red** lines on process ‘2’. [3 & 4]
- Data exchange of the halo regions across **neighbor** (up/down) processes.
- Mapped the processes to different sub domains on Cartesian grid, given a process (rank),
- find its coordinates, neighbors as well as enable transfer of data (**FOUR** exchanges).

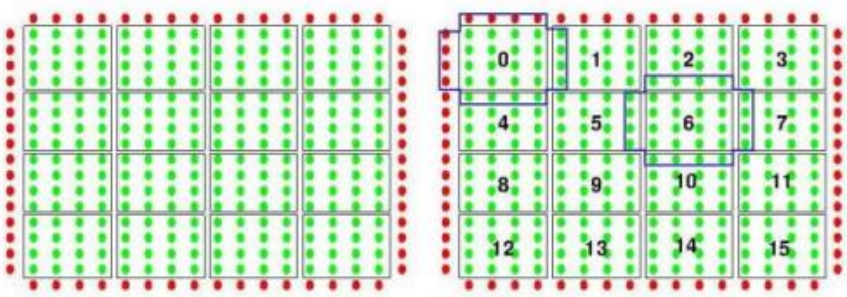


Fig. 2.3: Jacobi Solver (MPI: 2D decomposition)

- Geometric decomposition of the domain (across p processes) ‘p’ tiles.
- Working data for each process: **its domain + neighbor rows /columns** (halo regions).
- For computing $(M-1)^2/p$ points requires $4(M-1)/\sqrt{p}$ points around the sub domain (Known as halo), with $(M-1)/\sqrt{p}$ pt.s (different neighbor processes).
- Could be boundary points (fixed)/on the neighbor process (changes every iteration).
- Mapped the processes to different sub domains on Cartesian grid, given a process (rank), find its coordinates, neighbors as well as enable transfer of data (EIGHT exchanges).

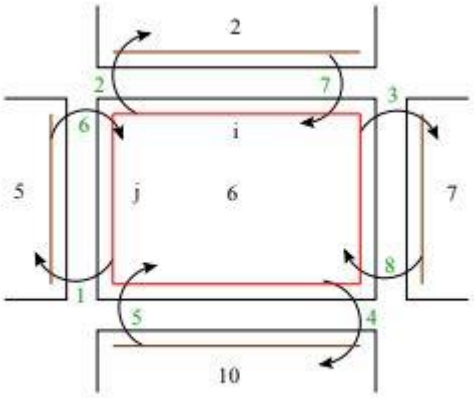


Fig. 2.4: Exchange of neighbors

- Data exchange of the halo regions among **neighbor** (up/down & left/right) processes.
- To compute u (i, j)'s on process '6' for points along **red** lines ('i' direction), one needs u(i,j)'s for points along **brown** lines ('i' direction) on processes '2' & '10'. [5 & 7]
- Similarly, to compute u (i, j) s on process '6' for points along **red** lines ('j' direction), one needs u (i, j)'s for points along **brown** lines ('j' direction) on processes '5' & '7'. [6 & 8]
- To compute u (i, j)'s on processes '2' & '10' for points along **brown** lines ('i' direction) and on processes '5' & '7' for points along **brown** lines ('j' direction), one needs u (i, j) s on the process 6 along **red** lines (two each along 'i' and 'j' directions). [2, 4 & 1, 3] - to compute the rank of the source and destination processes for communication

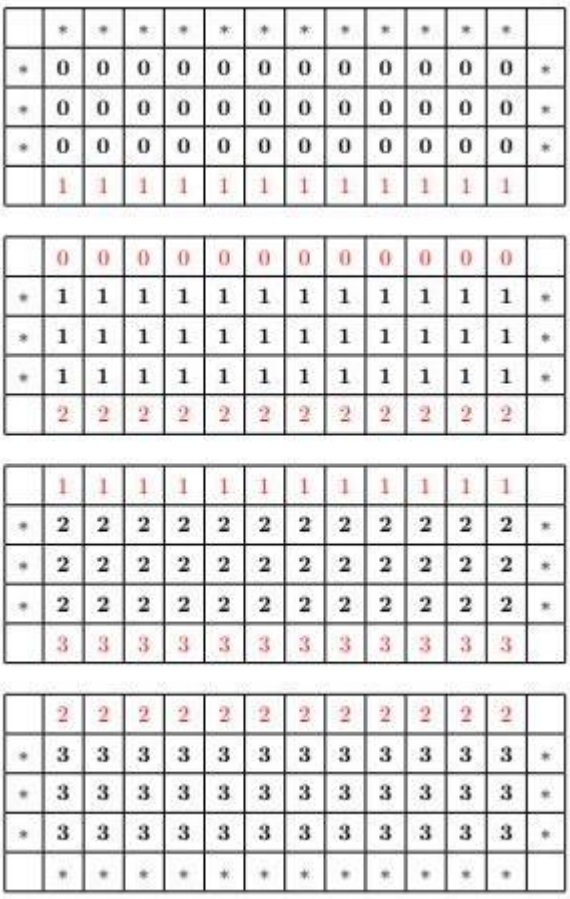
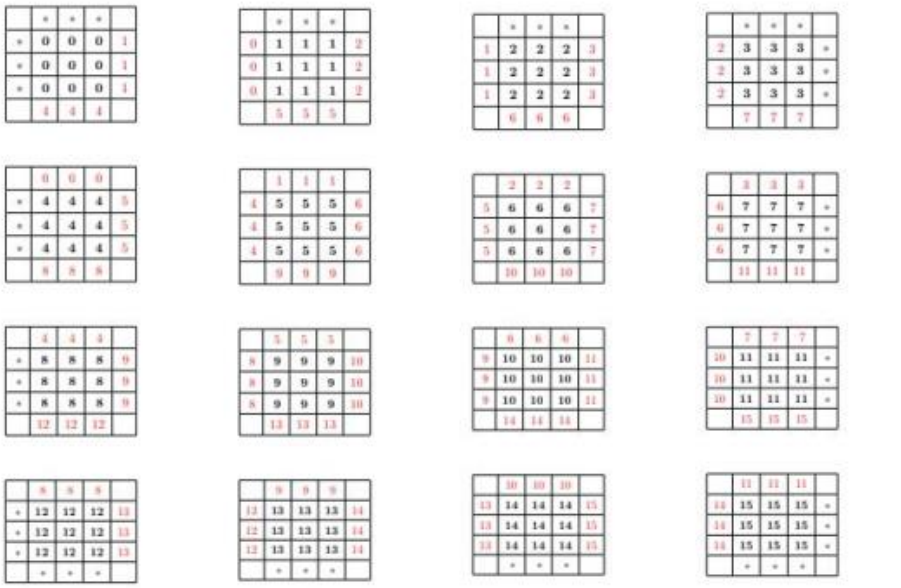


Fig. 2.5: Data Exchange

- 12 × 12 domain ⇒ 4 [3 × 12 sub-domains, 5 × 14 domain with boundaries (Ghost regions)]



12×12 domain ⇒ **16** [3×3 sub-domains, 5×5 domain with boundaries (ghost regions)]

Fig. 2.6: Data Exchange

- This is an iterative method where computation of a point requires four Neighbours of previous iteration. Here is the iterative solution for the Jacobi method where (m + 1) refers to current iteration and m refers to previous iteration.

$$U_{i,j}^{(m+1)} = \frac{1}{4}\left(U_{i+1,j}^{m} + U_{i-1,j}^{m} + U_{i,j+1}^{m} + U_{i,j-1}^{m}\right)$$

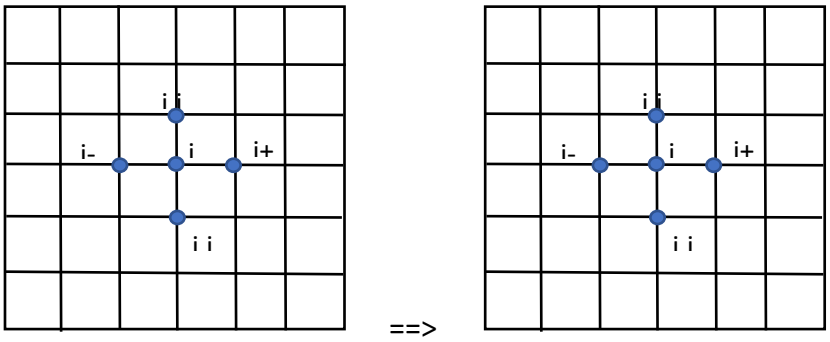


Fig. 2.7: Two data grids: (1) Read (previous iteration) and (2) Write (current iteration)

To compute a current iteration point four neighbors of the previous iteration will be required for that two grids are required one for read and one for write. Two grids are needed for the solver as shown in Fig.(1.8), one for write (current iteration dots) and another one for read(from previous iteration). To compute a point $(U_{i,j})$ write grid uses four Neighbour's from read grid. Similarly it computes all (n2) points then the read becomes write and the write becomes read since at every iteration the grids swapped.

Convergence Test: In convergence test take maximum absolute difference of current iteration and previous iteration among U i, j then compare the difference with some threshold (tolerance) if it is less then stop the execution else continue. This convergence test will be executed at each iteration.

Chapter 3

Implementation

Here we implemented 2d-Heat Equation

3.1 2D-Heat Equation

The heat equation and finite-difference method. Heat equation is basically a partial differential equation, it is

$$\frac{\partial u}{\partial t} - \alpha \nabla u = 0$$

If we want to solve it in 2D (Cartesian), we can write the heat equation above like this

$$\frac{\partial u}{\partial t} - \alpha \left(\frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y} \right) u = 0$$

where *u* is the quantity that we want to know, *it* is for temporal variable, *x* and *y* are for spatial variables, and α is diffusivity constant. So basically, we want to find the solution *u* everywhere in *x* and *y*, and over time *t*.

Now let's see the finite-difference method (FDM) in a nutshell. Finite-difference method is a numerical method for solving differential equations by approximating derivatives with finite differences. Remember that the definition of derivative is

$$f'(\alpha) = \lim_{h \to 0} \frac{\big(f(\alpha + h) - f(\alpha)\big)}{h}$$

In the finite-difference method, we approximate it and remove the limit. So, instead of using differential and limit symbol, we use delta symbol which is the finite difference. Note that this is oversimplified, because we have to use Taylor series expansion and derive it from there by assuming some terms to be sufficiently small, but we get the rough idea behind this method.

$$f'(\alpha) \approx \frac{\big(f(\alpha + h) - f(\alpha)\big)}{h}$$

In finite-difference method, we are going to "discretize" the spatial domain and the time interval *x*, *y*, and *t*. We can write it like this

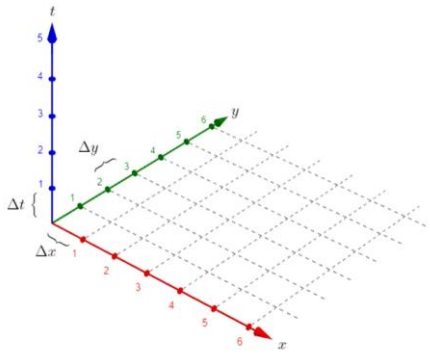


Fig 3.1: Cartesian coordinate (where x and y axis are for spatial variables, and *t* for temporal variable)

$$x_i = i\Delta x$$

$$y_j = j\Delta y$$

$$t_k = k\Delta t$$

As we can see, *i*, *j*, and *k* are the steps for each difference for *x*, *y*, and *t* respectively. What we want is the solution *u*, which is

$$u(x, y, t) = u_{i,j}^k$$

Note that *k* is superscript to denote time step for *u*. We can write the heat equation above using finite-difference method like this

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} - \alpha\left(\frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{\Delta x^2} + \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{\Delta y^2}\right) = 0$$

If we arrange the equation above by taking $\Delta x = \Delta y$, we get this final equation

$$u_{i,j}^{k+1} = \gamma\left(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - 4u_{i,j}^k\right) + u_{i,j}^k$$

Where

$$\gamma = \alpha\frac{\Delta t}{\Delta x^2}$$

We can use this stencil to remember the equation above (look at subscripts *i*, *j* for spatial steps and superscript *k* for the time step)

Fig 3.2: Explicit method stencil

We use explicit method to get the solution for the heat equation, so it will be numerically stable whenever

$$\Delta t \leq \frac{\Delta x^2}{4\alpha}$$

2.2 Implementation using AMReX Code

The original heat equation is approximated by algebraic equation, which is computer friendly. For an exercise problem, let's suppose a thin square plate with the side of 50-unit length. The temperature everywhere inside the plate is originally 40 degrees (at $t = 0$), let's see the diagram below (this is not realistic, but it's good for exercise)

AMReX_Utility.H

- `It` provides a number of convenient functions and macros that simplify common tasks in AMReX code and help to ensure correct behavior and efficient execution.
- `It` provides a convenient and flexible interface for writing and reading plot files in AMR simulations, which is an important aspect of analyzing and visualizing simulation data.

AMReX_ParmParse.H

- It provides a convenient and flexible interface for parsing input parameters in scientific simulations, which is an important aspect of making simulations more flexible and user-friendly.

AMReX_Print.H

- Provides a convenient and flexible interface for printing messages to the console or a file in scientific simulations, which is an important aspect of analyzing and visualizing simulation data.

amrex::Initialize () and amrex::Finalize ()

- Initialize amrex region where finalize end amrex region

BoxArray

- Union of Boxes at a given level

IntVect

- Dimension length array for indexing.

MultiFab

- Collection of FArrayBoxes at a single level
- Contains a Distribution Map defining the relationship across MPI Ranks.
- Primary Data structure for AMReX mesh-based work.

Ghost cell operations (int Nghost)

- FillBoundary - fills ghost cells from corresponding valid cells
- SumBoundary - adds from ghosts to corresponding valid

MFIter

- MFIter class is a useful tool for iterating over the grid boxes in a MultiFab object in a parallelized, cache-friendly manner, making it a valuable component of many AMR applications
- designed to work with the concept of tiling, which is a technique for
- **No table of contents entries found.**
- breaking up a large computation into smaller, cache-sized chunks that can be processed independently. By default, the MFIter class tiles the grid boxes into cache-sized chunks, allowing for efficient use of the memory hierarchy.

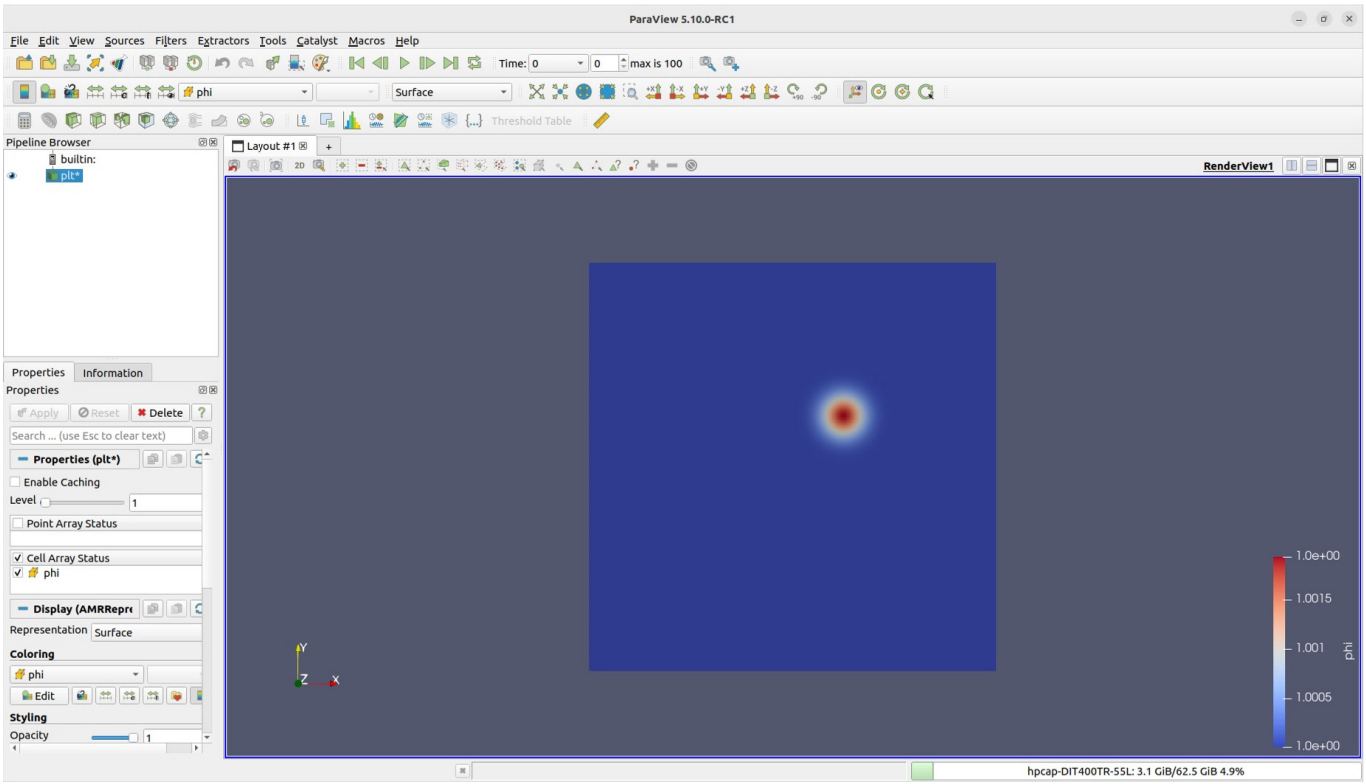
Chapter 4

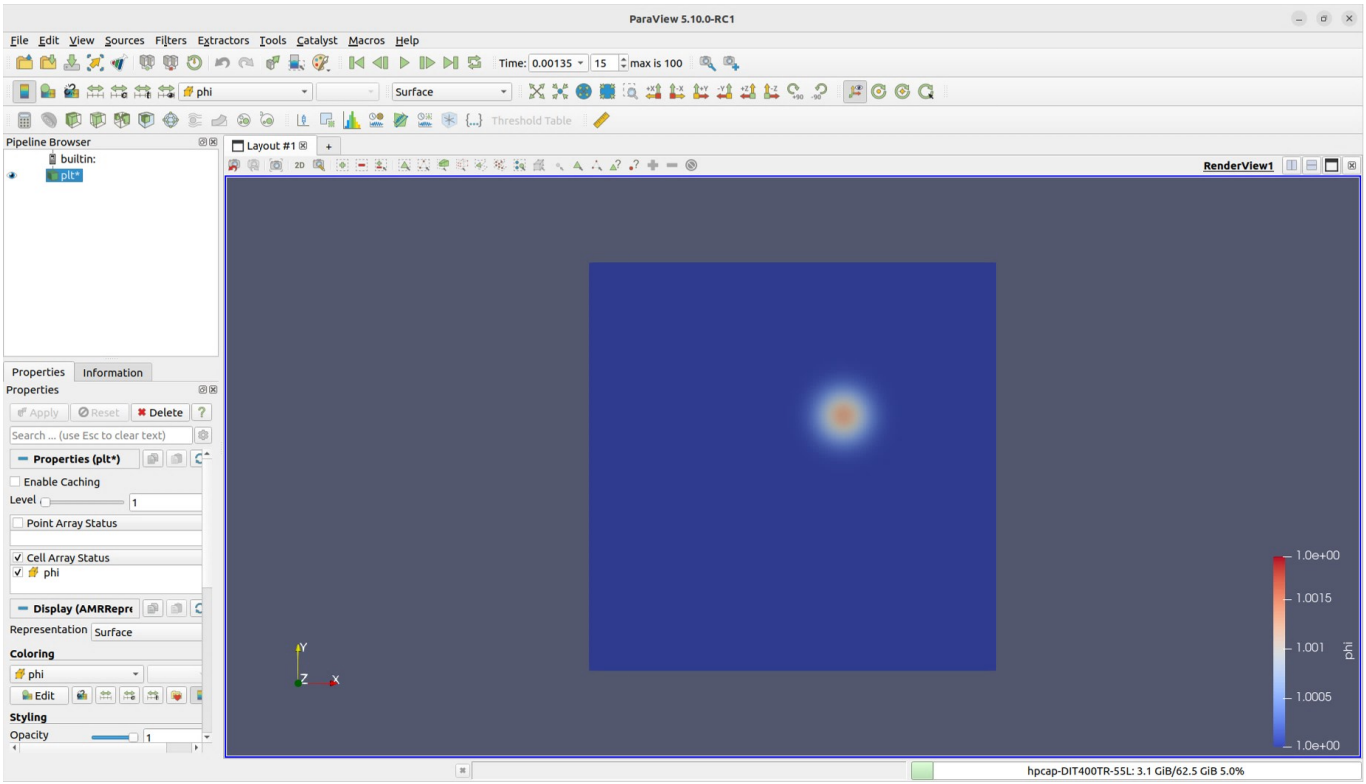
Conclusion

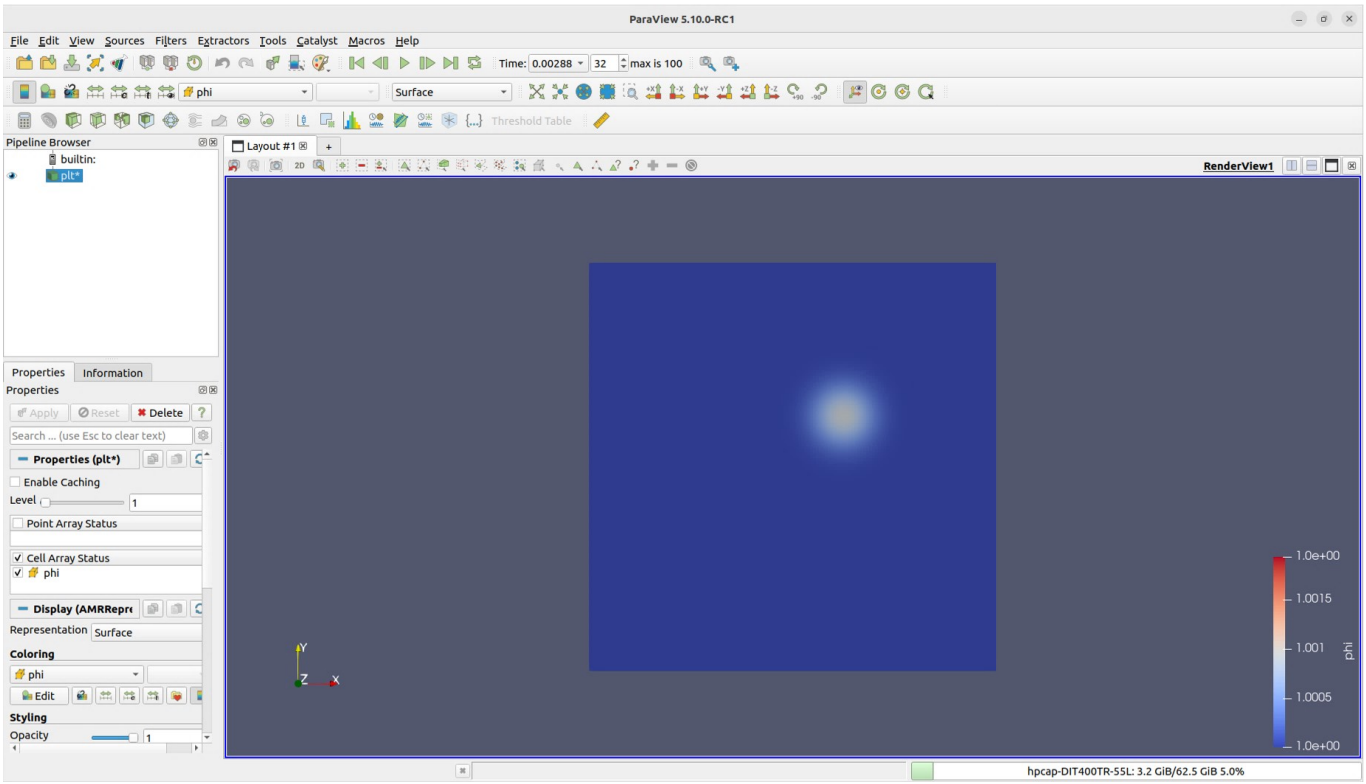
Here we study what time taken to heat diffuse from a high temperature to room temperature.

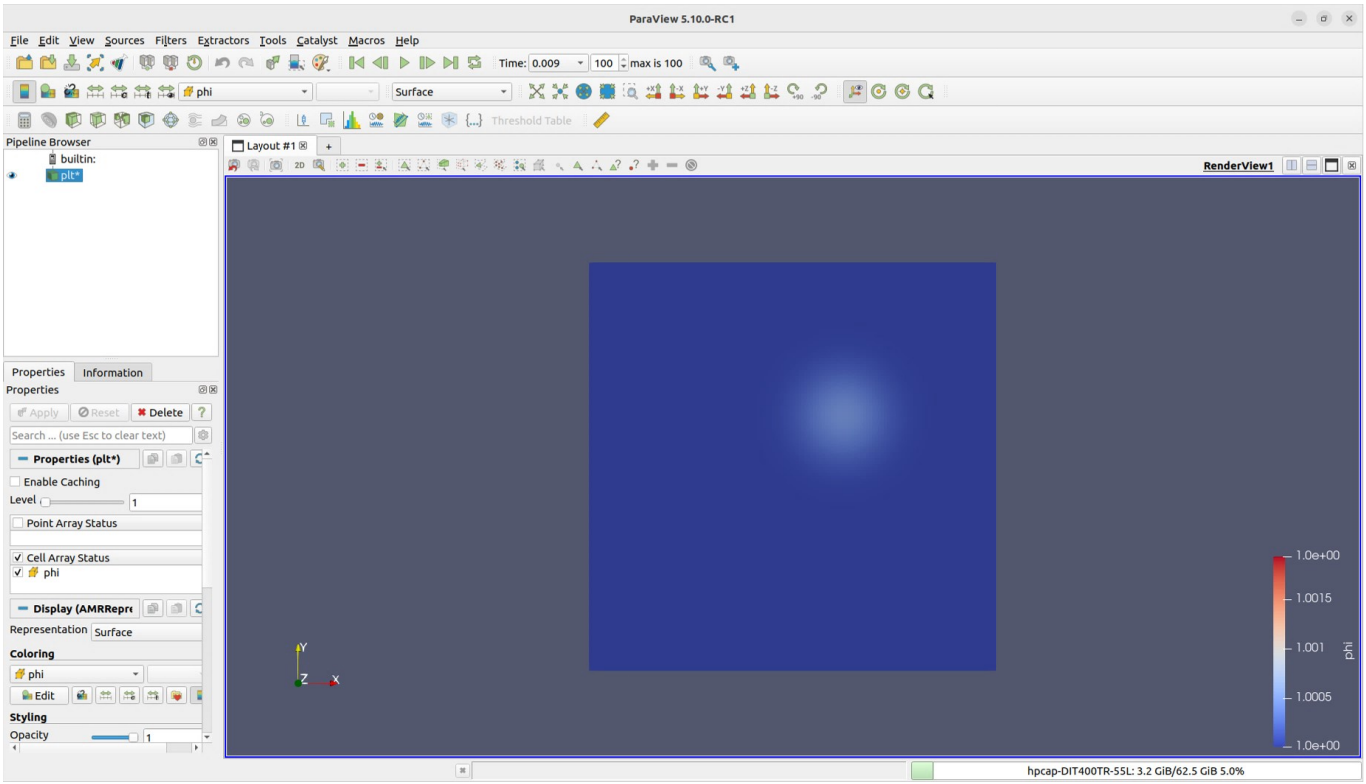
We take square plate which initially at 0^oC initial. At t=0 at partial part of plate heated to 47^oC. After that we observe the temperature of the plate and time.

For visualization we use ParaView Application and result visualization as follow:


ParaView 5.10.0-RC1
File Edit View Sources Filters Extractors Tools Catalyst Macros Help
Time: 0
max is 100
phi
Surface
Threshold Table
Pipeline Browser
builtin:
plt*
Layout #1
RenderView1
Properties
Information
Apply
Reset
Delete
Search ... (use Esc to clear text)
Properties (plt*)
Enable Caching
Level
Point Array Status
Cell Array Status
phi
Display (AMRRepr
Representation
Surface
Coloring
Edit
Styling
Opacity
1.0e+00
1.0015
1.001
1.0005
1.0e+00
phi
hpcap-DIT400TR-55L: 3.1 GiB/62.5 GiB 4.9%


ParaView 5.10.0-RC1
File Edit View Sources Filters Extractors Tools Catalyst Macros Help
Time: 0.00135
15
max is 100
phi
Surface
Threshold Table
Pipeline Browser
builtin:
plt*
Layout #1
RenderView1
Properties
Information
Apply
Reset
Delete
Search ... (use Esc to clear text)
Properties (plt*)
Enable Caching
Level
Point Array Status
Cell Array Status
phi
Display (AMRRepr
Representation
Surface
Coloring
Edit
Styling
Opacity
1.0e+00
1.0015
1.001
1.0005
1.0e+00
phi
hpcap-DIT400TR-55L: 3.1 GiB/62.5 GiB 5.0%


ParaView 5.10.0-RC1
File Edit View Sources Filters Extractors Tools Catalyst Macros Help
Time: 0.00288 32 max is 100
phi
Surface
Threshold Table
Pipeline Browser
builtin:
plt*
Layout #1
RenderView1
Properties Information
Properties
Apply Reset Delete
Search ... (use Esc to clear text)
Properties (plt*)
Enable Caching
Level 1
Point Array Status
Cell Array Status
phi
Display (AMRRepr
Representation Surface
Coloring
phi
Edit
Styling
Opacity 1
1.0e+00
1.0015
1.001
1.0005
1.0e+00
phi
hpcap-DIT400TR-55L: 3.2 GiB/62.5 GiB 5.0%


ParaView 5.10.0-RC1
File Edit View Sources Filters Extractors Tools Catalyst Macros Help
Time: 0.009 100 max is 100
phi
Surface
Threshold Table
Pipeline Browser
builtin:
plt*
Layout #1
RenderView1
Properties Information
Properties
Apply Reset Delete
Search ... (use Esc to clear text)
Properties (plt*)
Enable Caching
Level 1
Point Array Status
Cell Array Status
phi
Display (AMRRepr
Representation Surface
Coloring
phi
Edit
Styling
Opacity 1
1.0e+00
1.0015
1.001
1.0005
1.0e+00
phi
hpcap-DIT400TR-55L: 3.2 GiB/62.5 GiB 5.0%