Parallelizing Numerical PDE Solvers

Isaac Shirk, Laurel Koenig

Introduction

Partial differential equations are ubiquitous throughout many scientific fields.

Some common examples include:

The Heat / Diffusion Equation

"u" can either be temperature or a concentration

$$\frac{\partial u}{\partial t} = \nabla^2 u$$

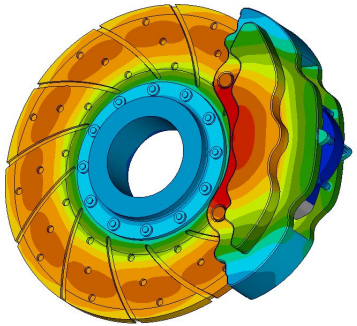


Image Credit: https://www.theseus-fe.com/ths_content/images/thematic/thermal/heat-transfer_conduction_brake-disc.png

The Wave Equation

Useful in acoustics and vibration modeling.

$$\frac{\partial^2 u}{\partial t^2} = \nabla^2 u$$

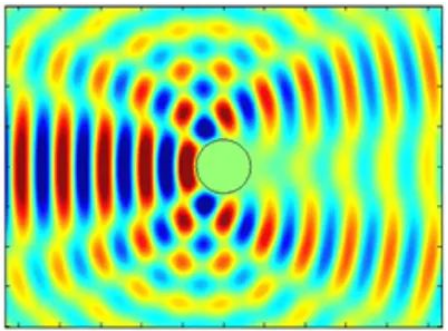


Image Credit:
https://spectrum.ieee.org/media-library/controlling-acoustic-wave-scattering-from-an-object.png?id=25583667&width=400&height=296

Solving PDEs

PDEs can be solved analytically in simple cases.

Often times the problems that need to be solved have complicated domains and boundary conditions, such that it is not feasible to compute the analytical solution.

This necessitates using numerical methods to get an approximate solution within some error bound.

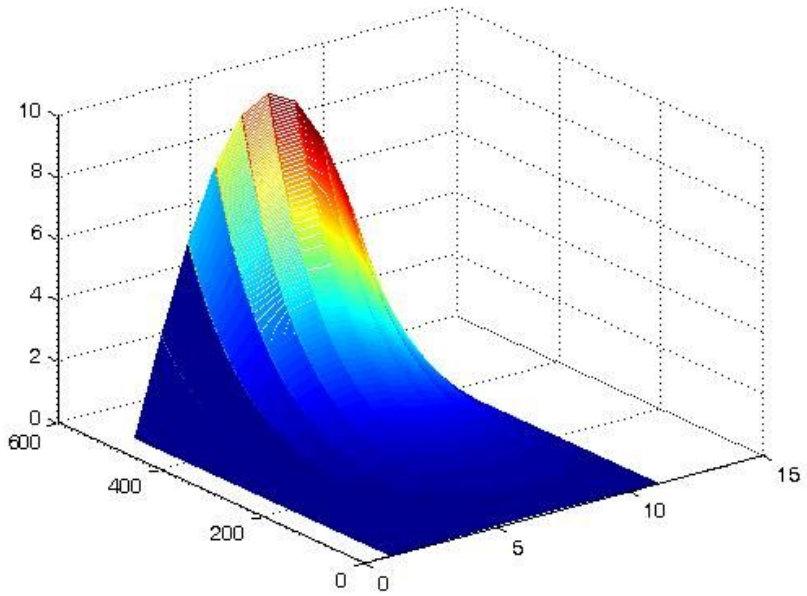


Image Credit: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/40096/versions/1/screenshot.jpg

Different Numerical Methods

- Finite Difference Method
- Finite Element Method
- Multigrid Method

Finite Difference Method

One of the simplest methods.

It aims to approximate derivatives over a short step, or difference, as a method of approximating a solution.

Divide the domain into a grid of discrete spatial points, and use the fixed distances between points to estimate derivatives.

The three most common variations are:

- Explicit Method
- Implicit Method
- Crank-Nicholson Method

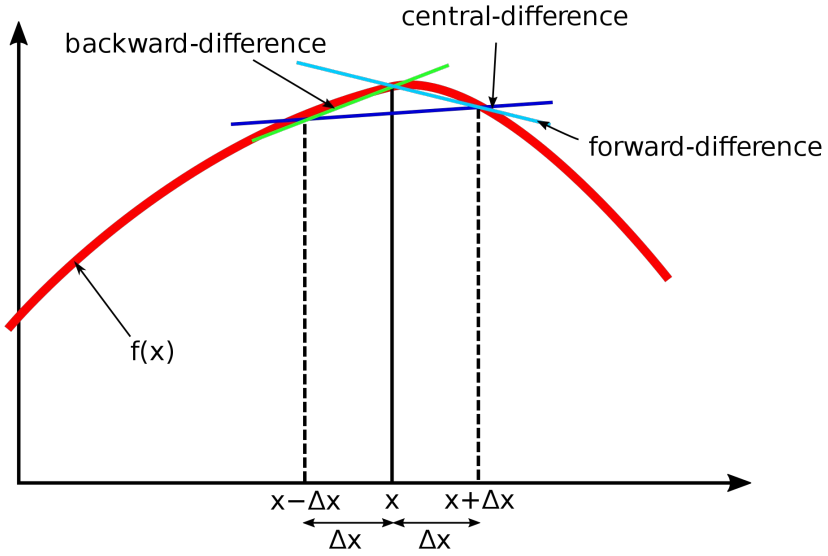


Image Credit: https://en.wikipedia.org/wiki/Finite_difference#/media/File:Finite_difference_method.svg

Finite Element Method

Similar to the Finite Difference Method, this method divides a space up into a collection of small elements that are easier to solve.

Unlike the previous method, Finite Element Method can discretize irregular spaces, and can use a triangular mesh

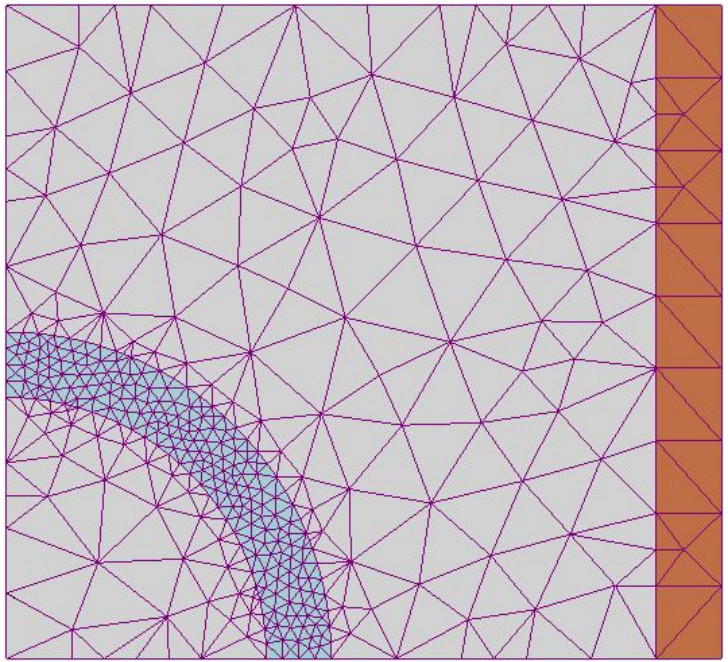


Image Credit: https://en.wikipedia.org/wiki/Finite_element_method#/media/File:Example_of_2D_mesh.png

Multigrid Method

The multi-grid method increases the convergence of a solution on a fine grid by augmenting it with calculated solutions from successively coarser grids.

It can be paired with the Finite Element Method as well.

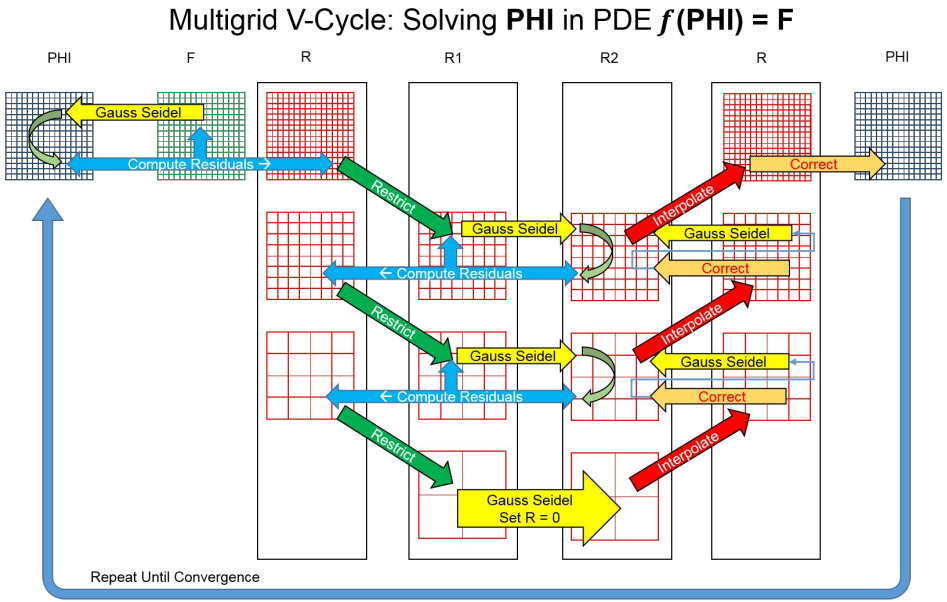


Image Credit:
https://upload.wikimedia.org/wikipedia/commons/thumb/8/8c/Multigrid_Visualization.png/456px-Multigrid_Visualization.png

We chose to Implement Finite Difference in Parallel

Why?

- It is the most straightforward to implement
  - There is only one grid to keep track of, unlike Multigrid
  - The grid is rectangular and regular, unlike the variable triangular meshes in Finite Element
- A plain rectangular grid is easier to divide up among processes for parallel computation

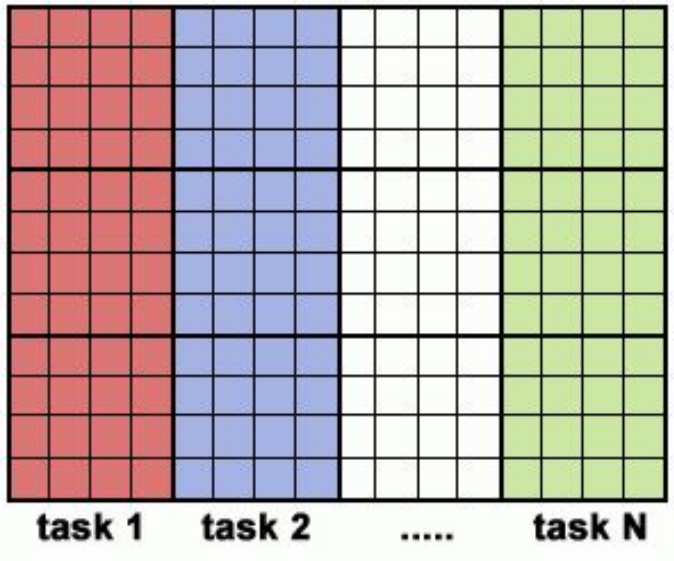
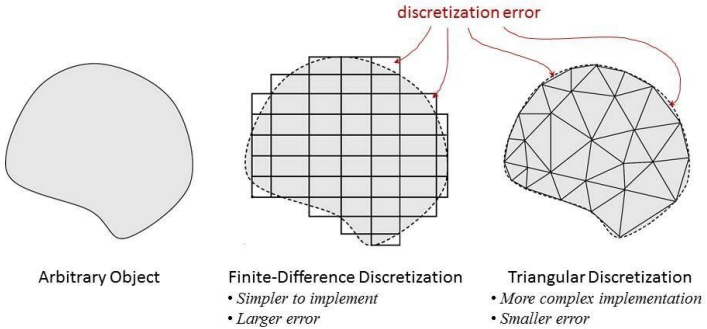


Image Credit: https://hpc.llnl.gov/sites/default/files/styles/no_sidebar_3_up/public/array_proc2.gif?itok=Cict07K0

Challenges When Using the Finite Difference Method

- Curved, or other irregular boundaries can be difficult to discretize
- If a boundary moves over time, that movement can also lead to discretization errors.
- To ensure good accuracy a very fine mesh must be used.
    - In higher dimensions, this is a n^2 or n^3 increase in computation time
    - If most of the error occurs in one part of the grid, then much of the computation will be wasted on areas where nothing interesting is happening.



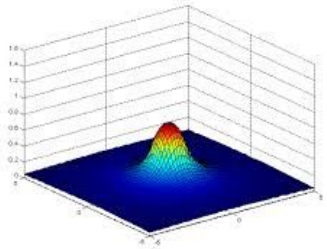


Image Credit: https://i.ytimg.com/vi/BddajYoBHiI/maxresdefault.jpg

Image Credit: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSnpEU8OymNLhJPCj-j4hvE_upLSt655jmXVw&usqp=CAU

A Solution: Adaptive Grid

An adaptive mesh grid is one that does not always have the same resolution.

In areas that need more precision, you can increase the grid resolution.

This also works to better represent curves!

The changes in resolution are typically in powers of two.

This way you don't waste computation time on less interesting areas of the domain.

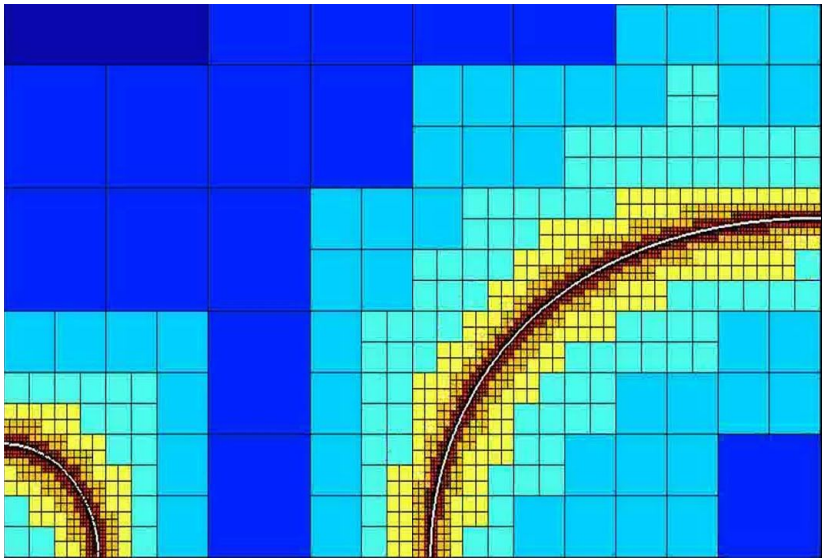


Image Credit: https://i.ytimg.com/vi/5NvYsI4szwY/maxresdefault.jpg

How To Use the Adaptive Grid

Starting with some regular grid, estimate the error at each point and increase the resolution where it is too high.

You also need to define a special rule for calculating derivative at asymmetric grid points

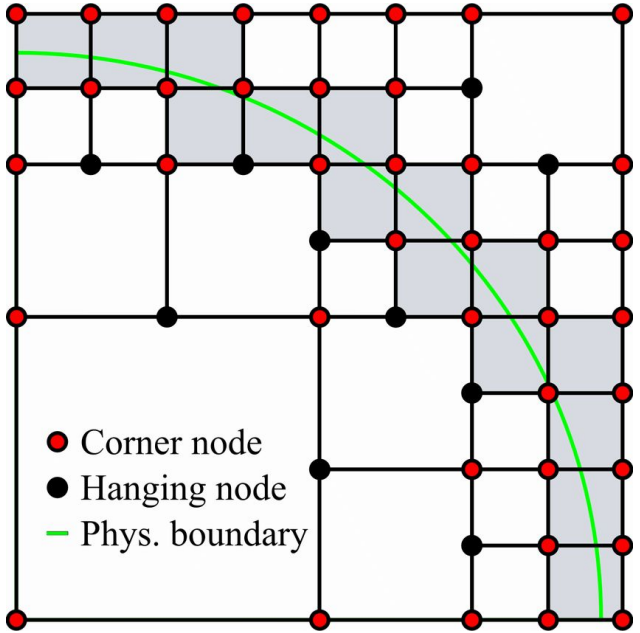


Image Credit:https://www.researchgate.net/publication/317584421/figure/fig6/AS:668258034282509@1536336539748/Generic-finite-element-quadtree-mesh-with-hanging-nodes.ppm

Code Structure

We will be writing this project in C++ and Julia to compare performance and ease of implementation

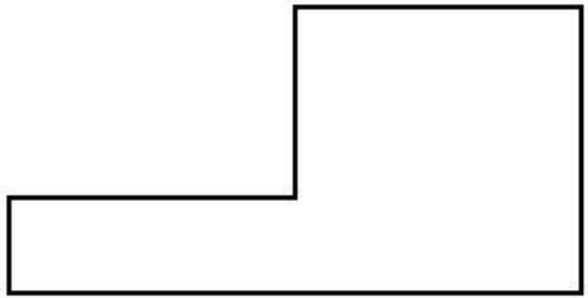
We will now review the high level view of the code as applied to a 2D Heat equation

Define the Domain

This is the entire computational domain, and must be constructed of rectangular units.

Curved edges can input as special boundary conditions later

Can get the specifications from a file or from the user.

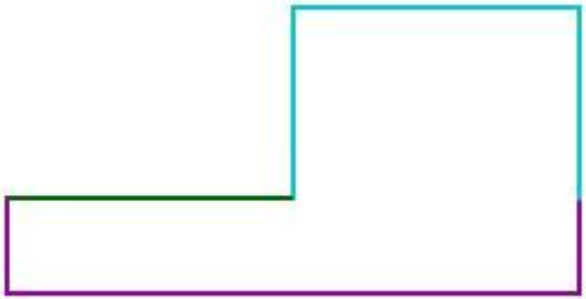


Define The boundary conditions

These are the conditions applied at the edge of the grid

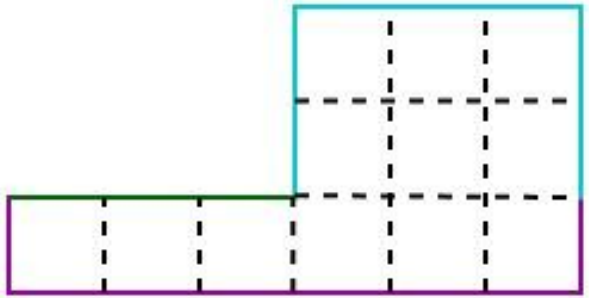
You could also define some block in the middle of the grid at some constant temperature

Can also specify some curved interior boundary.



Generate the Basic Mesh

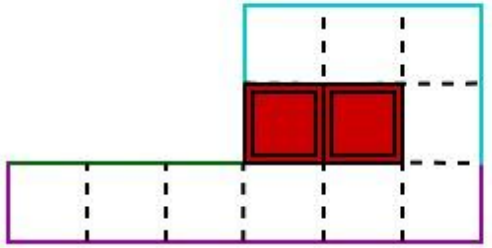
This would be represented as 2D arrays holding temperature values at each point in the grid.

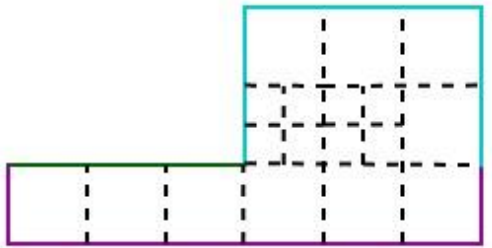


Adapt the Grid Based on Error Estimations

Calculate the error at each point on the grid, and mark the nodes for expansion.

Increase the resolution of the grid at those nodes, and repeat until you get to the finest resolution you allow.



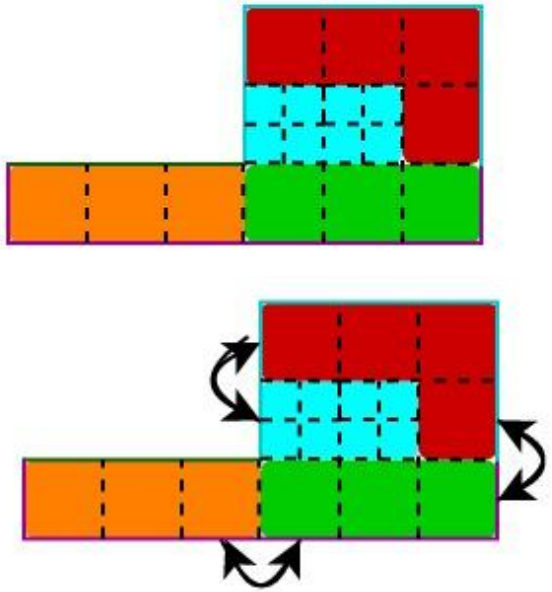


Divide the Grid to be Processed in Parallel

Divide the grid into chunks to send to separate processes.

Should divide into roughly equal number of computations, and should have the same resolution throughout.

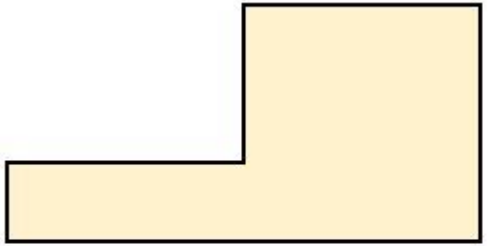
Boundaries between chunks should not be too complicated to eases communication between processes.

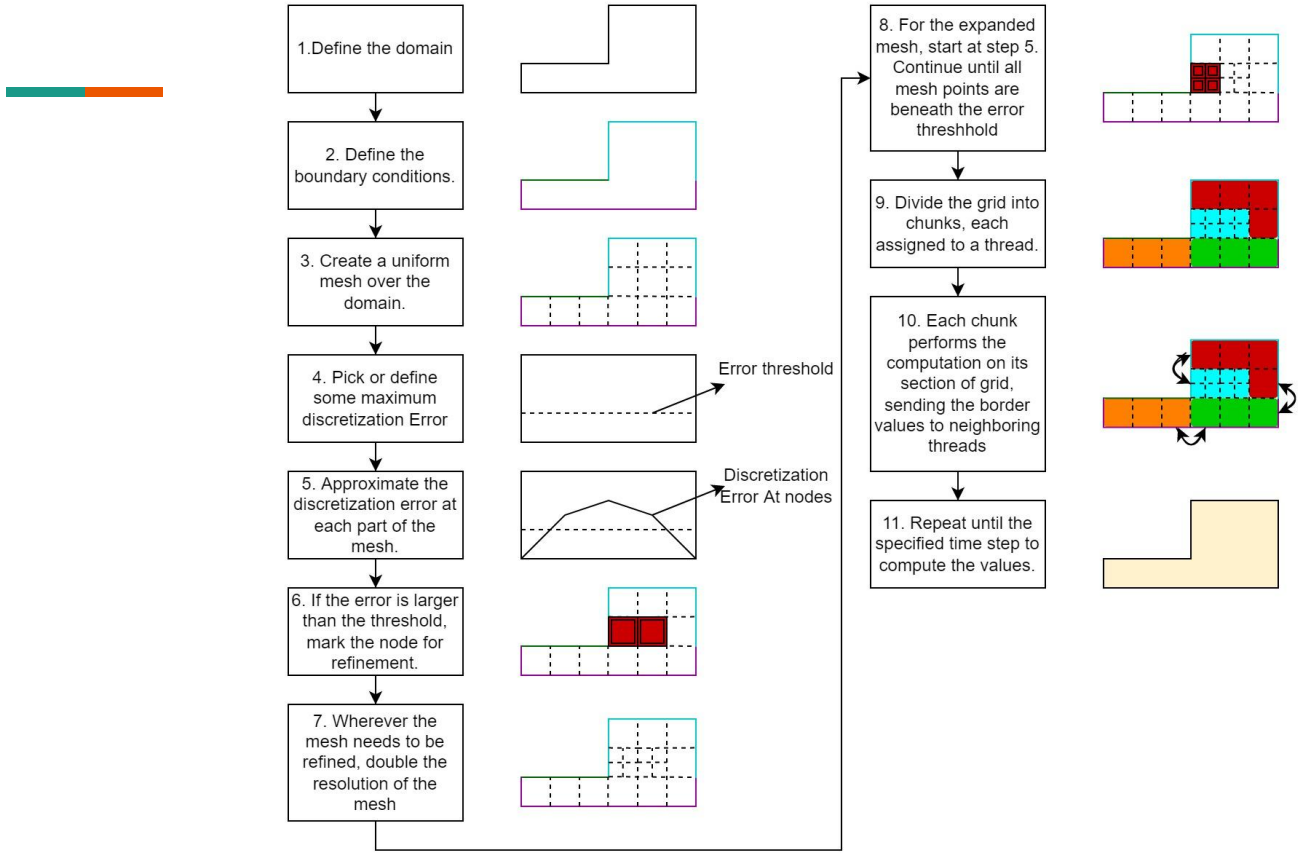


Iterate the PDE Solver

Run the solver on each chunk until the desired time.

Compare the results to a uniform, high resolution grid to get an estimate of the error.




1.Define the domain
2. Define the boundary conditions.
3. Create a uniform mesh over the domain.
4. Pick or define some maximum discretization Error
Error threshold
5. Approximate the discretization error at each part of the mesh.
Discretization Error At nodes
6. If the error is larger than the threshold, mark the node for refinement.
7. Wherever the mesh needs to be refined, double the resolution of the mesh
8. For the expanded mesh, start at step 5. Continue until all mesh points are beneath the error threshhold
9. Divide the grid into chunks, each assigned to a thread.
10. Each chunk performs the computation on its section of grid, sending the border values to neighboring threads
11. Repeat until the specified time step to compute the values.

Code Examples

```
using LinearAlgebra

function CBEM( x, t, b1, b2, f, k)
    nx = length(x)
    nt = length(t)
    r = k * Float64(t.step) / Float64(x.step)^2

    M = SymTridiagonal(fill(1-2r, nx-2),
                       fill(r, nx-3))

    U = [reshape(b1.(t),1,nt);
         [f.(x[2:end-1]) zeros(nx-2, nt-1)];
         reshape(b2.(t),1,nt)]

    for step ∈ 2:length(t)
        b = [r*U[1,step-1]; zeros(nx-4); r*U[end, step-1]]
        U[2:end-1, step] = M * U[2:end-1, step-1] + b
    end

    return U
end
```

```
Threads.@threads for x in 1:100 y in 1:100
    arr2[x,y] = (arr1[x+1,y]+arr1[x-1,y]+arr1[x,y+1]+arr1[x,y-1])/4
end
```

Anticipated Results

- It should be fast
  - Less unnecessary computation
  - Parallelization of computation
- It should be accurate
  - Increases spatial resolution when needed
- It would struggle with moving boundary conditions
  - The mesh adaptation would not be able to update to the new position of the boundary condition.

Resources

1. Sakane, S., Takaki, T., and Aoki, T. (2022). Parallel-GPU-accelerated adaptive mesh refinement for three-dimensional phase-field simulation of dendritic growth during solidification of binary alloy. Materials Theory, 6(1). https://doi.org/10.1186/s41313-021-00033-5
2. Solve a 2D Heat Equation Using Data Parallel C++. Intel. (2022). Retrieved 2 September 2022, from https://www.intel.com/content/www/us/en/developer/articles/technical/solve-a-2d-heat-equation-using-data-parallel-c.html\#gs.b31uc2.
3. Horak, Verena \& Gruber, Peter. (2005). Parallel Numerical Solution of 2-D Heat Equation. 3. 47-56.
4. Create adaptive 2-D mesh and solve PDE. MathWorks. (2022). Retrieved 2 September 2022, from https://www.mathworks.com/help/pde/ug/adaptmesh.html.
5. Preney, T., Wheeler, J.D., \& Namy, P. (2018). Adaptive mesh refinement: Quantitative computation of a rising bubble using COMSOL Multiphysics®.
6. Oberman, A. M., & Zwiers, I. (2016). Adaptive finite difference methods for nonlinear elliptic and parabolic partial differential equations with free boundaries. *Journal of Scientific Computing*, *68*(1), 231-251.
7. Jovanovic, R., Tuba, M., Simian, D., & ROMANIA, S. S. (2008, July). An algorithm for multi-resolution grid creation applied to explicit finite difference scheme. In *Proceedings of the 12th WSEAS international conference on Computers* (pp. 1123-1128).
8. https://math.mit.edu/~stevenj/18.303/Lecture1.pdf