Parallel Adaptive Resolution Numerical PDE Solutions

Isaac Shirk, Laurel Koenig

Introduction: A Recap

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

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

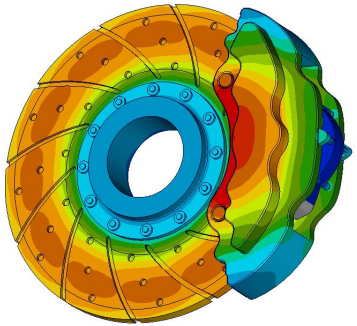


Image Credit: https://www.theseus-fe.com/ths_content/images/thematic/thermal/heat-transfer_conduction_brake-disc.png

The Wave Equation

Useful in acoustics and vibration modeling.

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

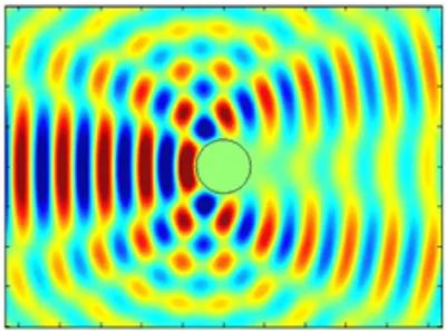


Image Credit:
https://spectrum.ieee.org/media-library/controlling-acoustic-wave-scattering-from-an-object.png?id=25583667&width=400&height=296

Solving PDEs

PDEs can be solved analytically in simple cases.

It is often not feasible to compute the analytical solution.

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

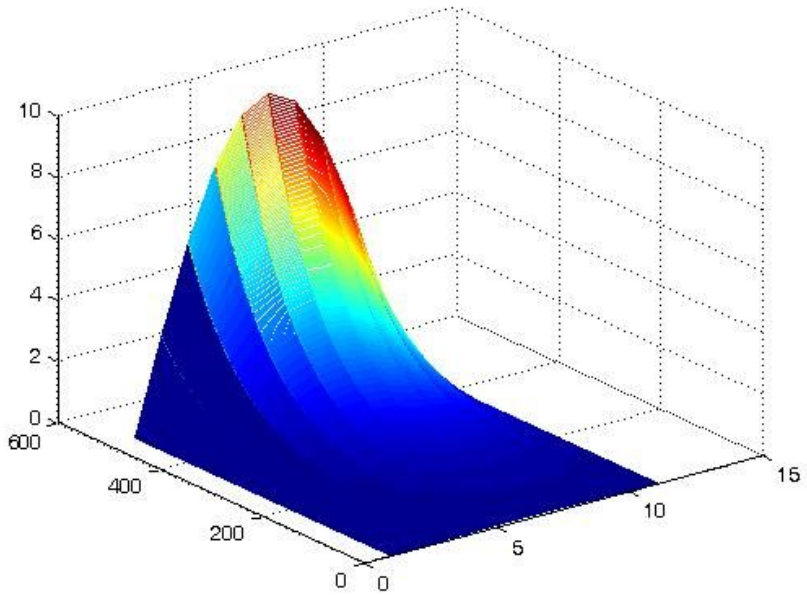


Image Credit: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/40096/versions/1/screenshot.jpg

We chose to Implement Finite Difference in Parallel

- It would allow for easier parallelization compared to other numerical methods
- We had previous experience with Finite difference methods

$$u_{i,j}^{t+1} = u_{i,j}^{t} + \Delta t \cdot \left(\frac{u_{i+1,j}^{t} + u_{i-1,j}^{t} + u_{i,j+1}^{t} + u_{i,j-1}^{t} - 4u_{i,j}^{t}}{h^2} \right)$$

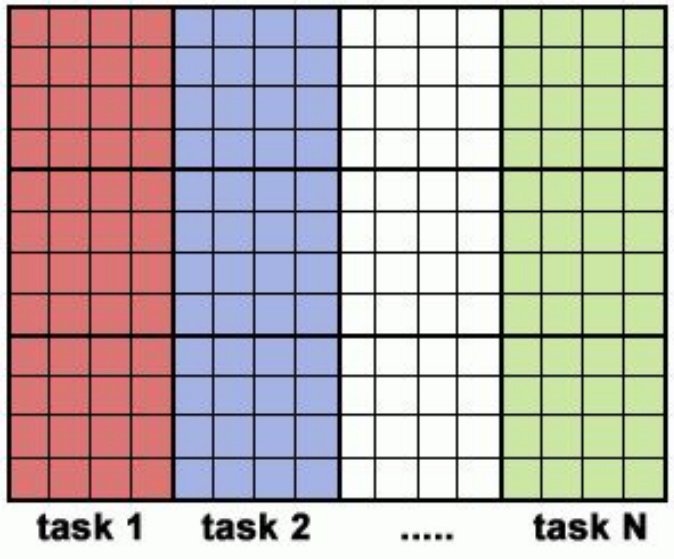


Image Credit: https://hpc.llnl.gov/sites/default/files/styles/no_sidebar_3_up/public/array_proc2.gif?itok=Cict07K0

Variation: Adaptive Resolution Grid

A way to make the project more interesting

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

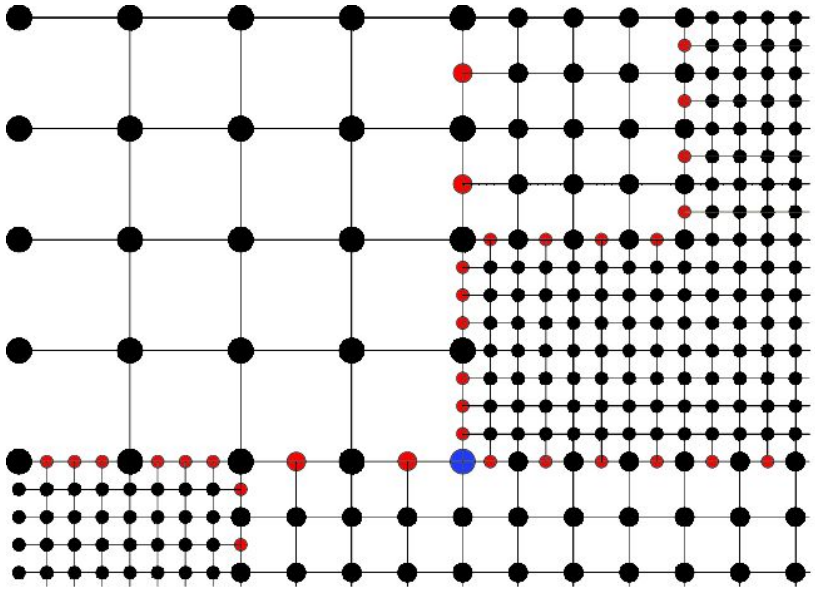
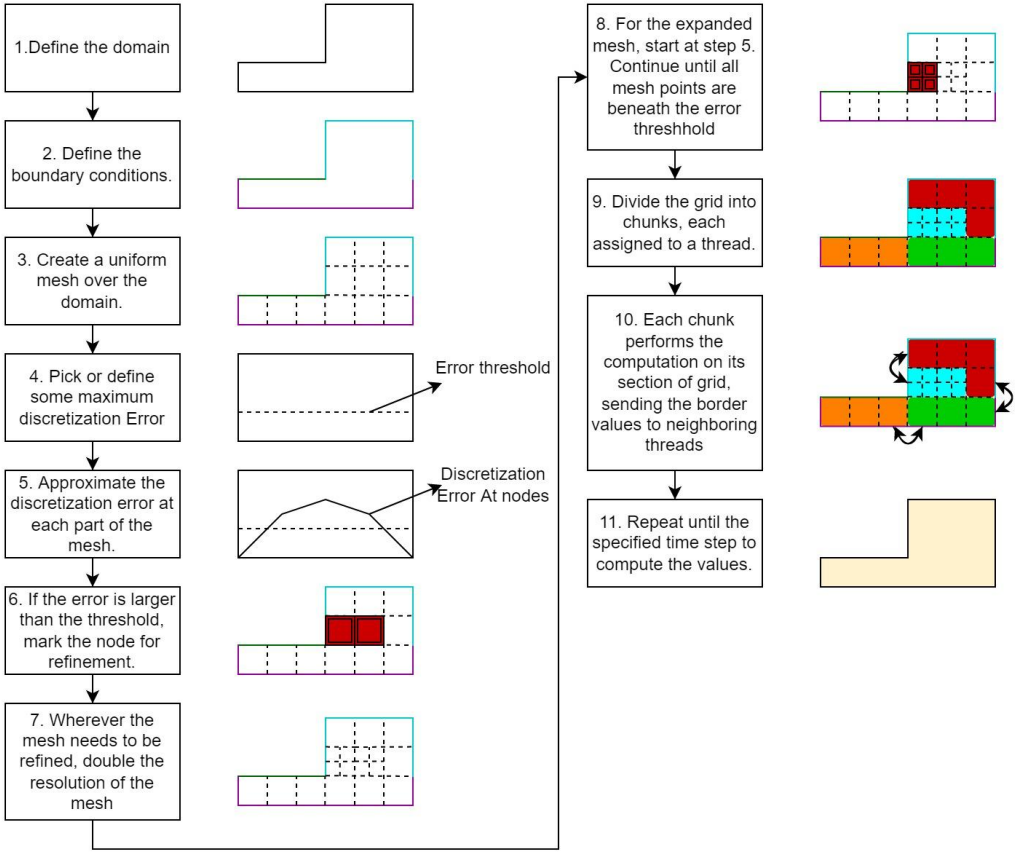


Image Credit:
https://www.researchgate.net/figure/Depiction-of-our-node-centered-adaptive-grid-representation-Red-nodes-indicate_fig5_225292193

Process Review



Adaptive Grid Construction: The Quadtree

A common method of adaptive resolution construction is to use a quadtree.

Similar to binary trees, can be used to search 2D points in logarithmic time!

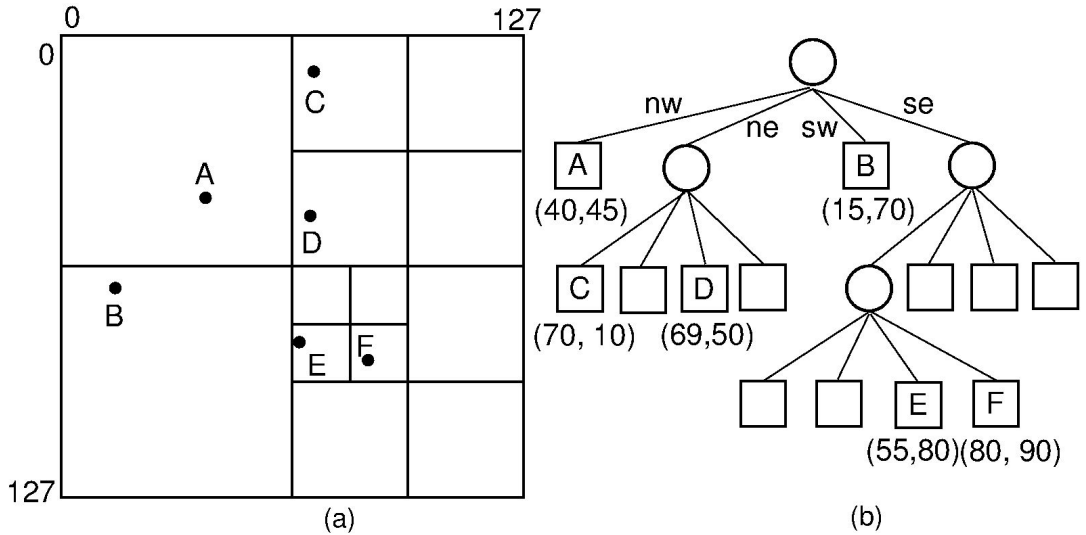


Image Credit: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/PRquadtree.html

Julia and Quadtrees

Used the Julia RegionTrees library[9]

We ended up using only Julia for this project as there were no libraries for quadtrees or sparse arrays that we were able to use.

RegionTrees was missing some critical functions

- Nearest neighbor
- Balance quadtree

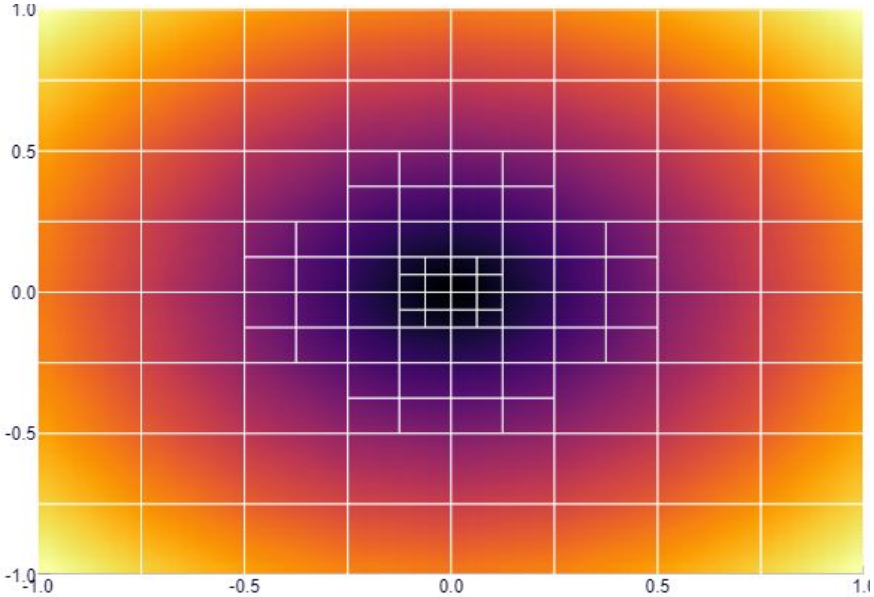


Image Credit: https://github.com/rdeits/RegionTrees.jl

Algorithm: Nearest Neighbor

Unlike an array, finding a nearest neighbor is not easy

Written using the pseudo code from [10]

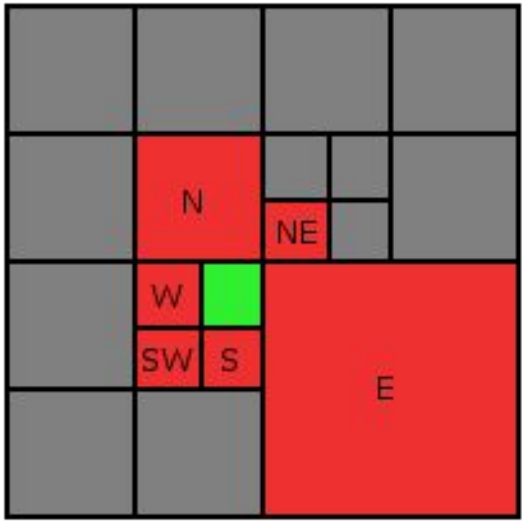


Image Credit: https://geidav.wordpress.com/2017/12/02/advanced-octrees-4-finding-neighbor-nodes/

Algorithm: Balance Quadtree

This is a way to make sure that all leaves are within one level of each other

This also has the nice property of guaranteeing a maximum 1 level of resolution change between sections of the adaptive resolution grid

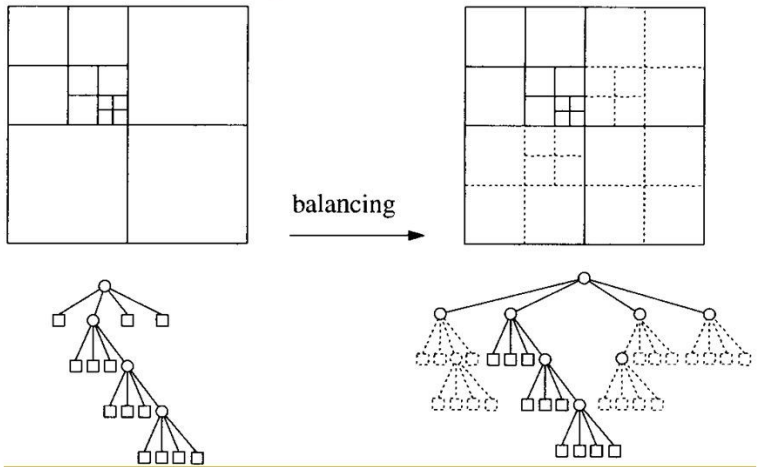
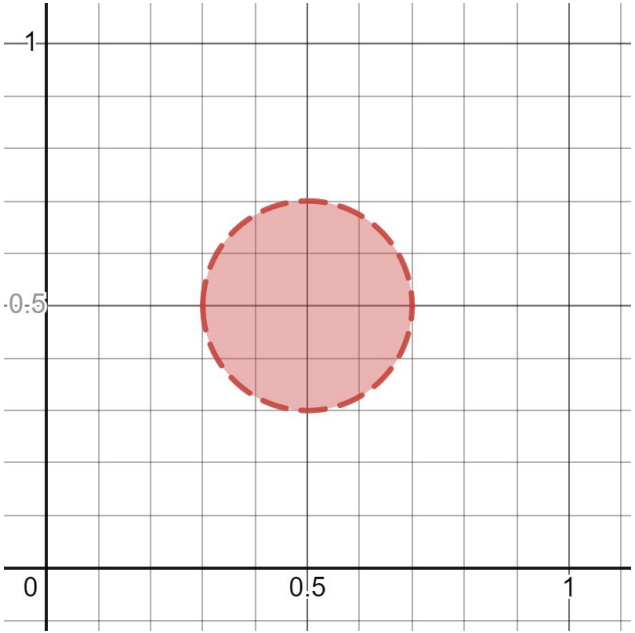


Image Credit: https://slideplayer.com/slide/8829374/

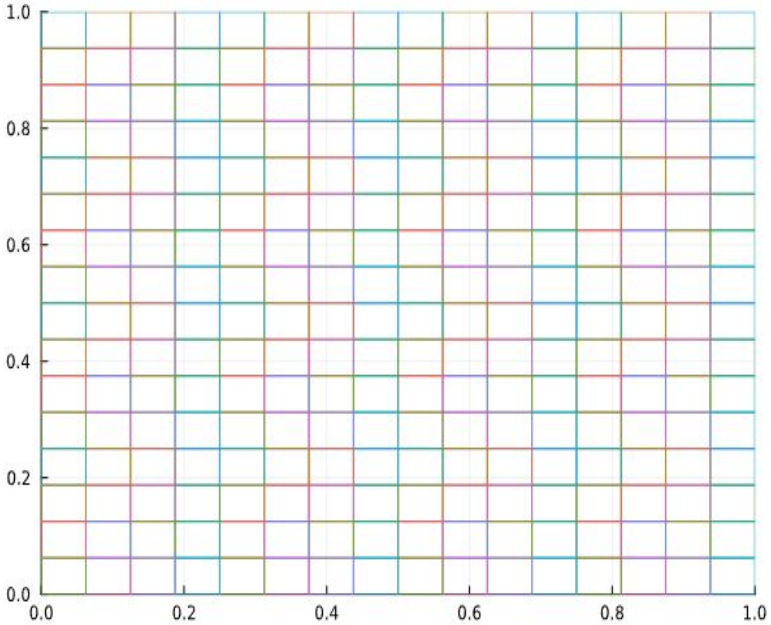
Problem Setup:

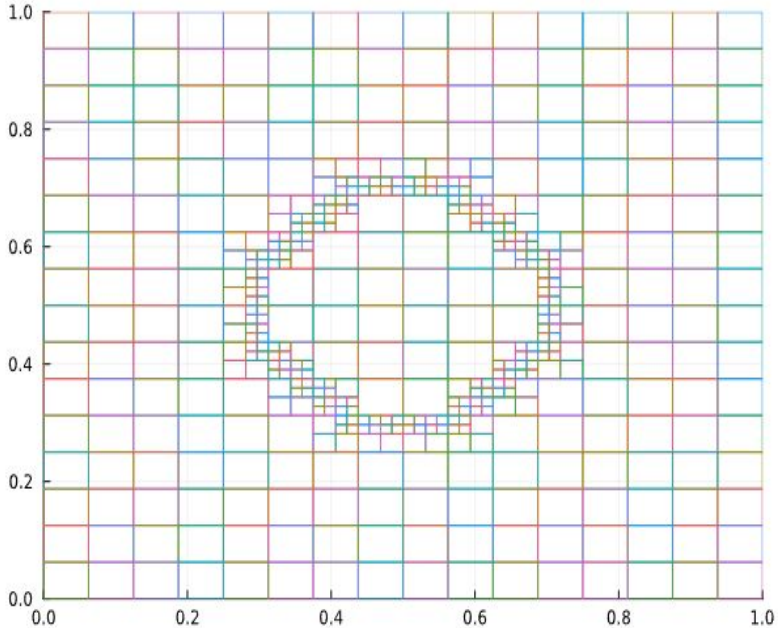
Now that we have our quadtree tools ready:

- Constant boundary conditions
- Circular center region set at a high value
- Rest of the square region set to a low value
- Maximum resolution: 1/64,
- Minimum resolution: 1/16
- Delta t: 0.0001s
- Final t: 4s

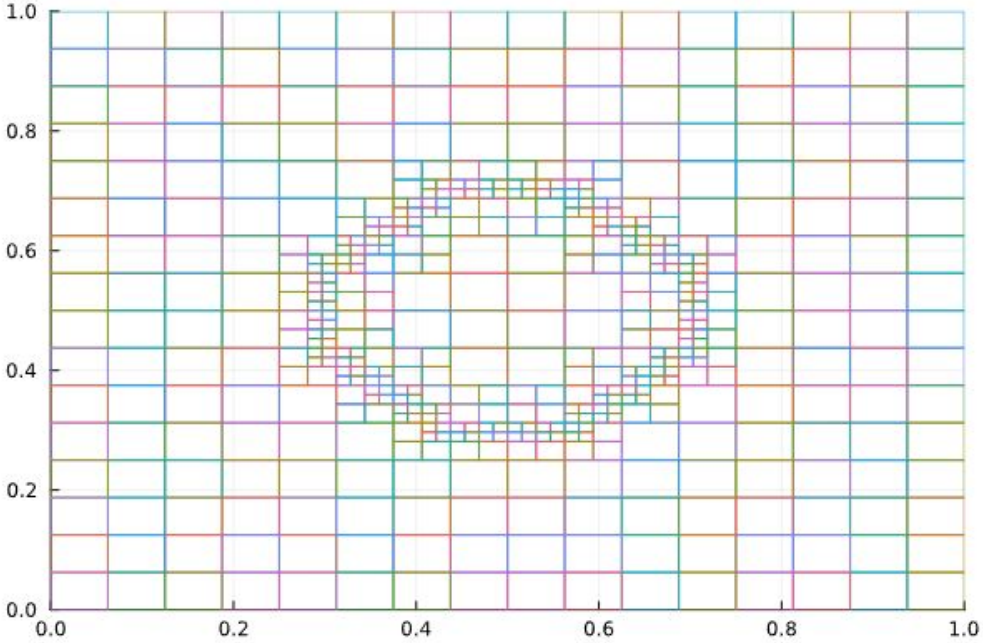


Grid Construction





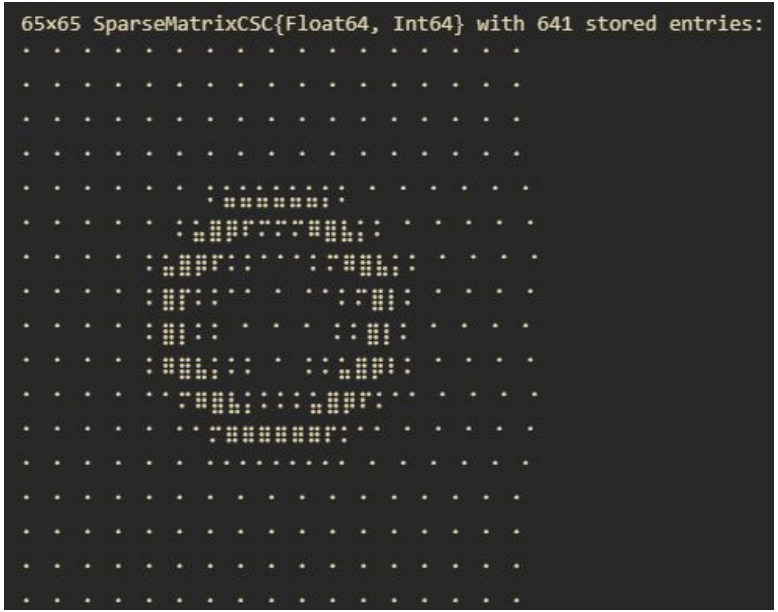
Balancing the Quadtree



Convert to Sparse Array

Used a sparse array at the recommendation of another paper that implemented adaptive grid computations [4]

4225 elements vs. 641



Parallelization Through Domain Partitioning

The data to operate on is large

The operation performed on each data point is relatively simple

Use columns to partition for data locality

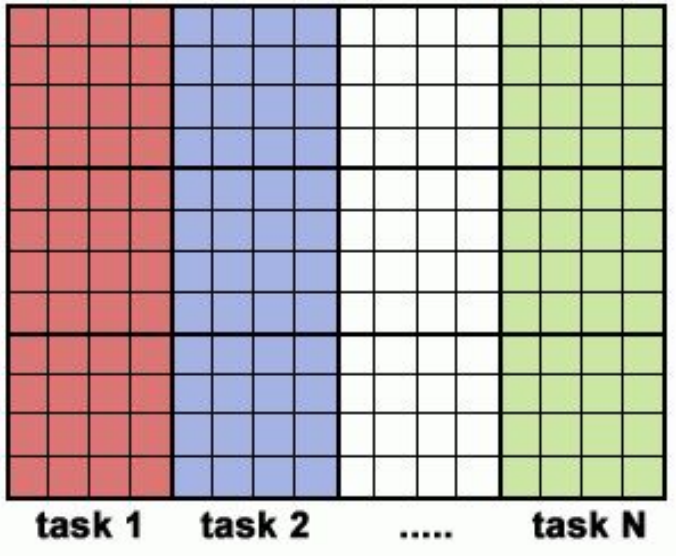


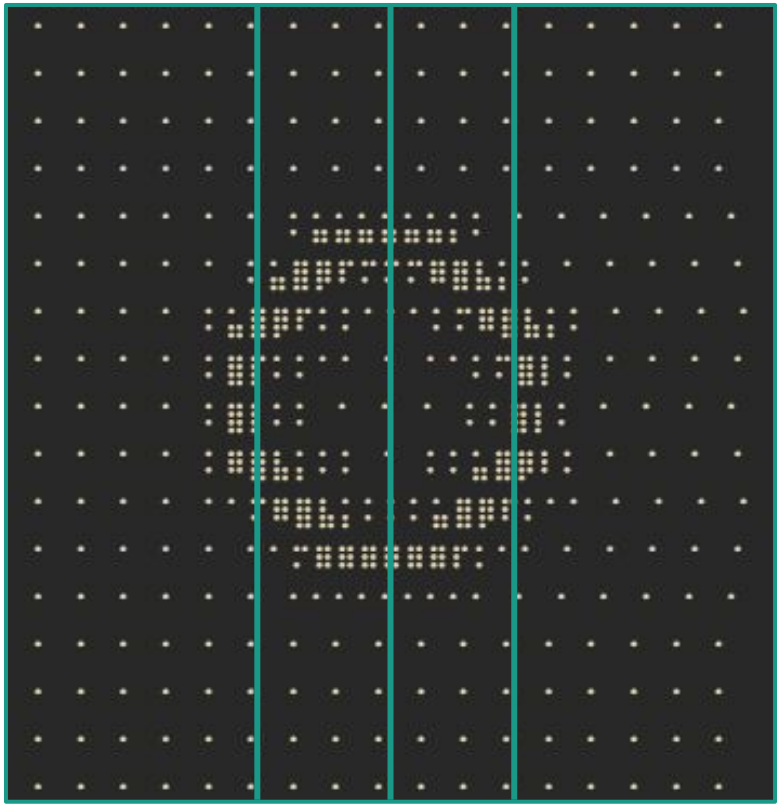
Image Credit: https://hpc.llnl.gov/sites/default/files/styles/no_sidebar_3_up/public/array_proc2.gif?itok=Cict07K0

Load Balancing

Can’t use naive equal column partitions!

Divide total number of elements by the desired number of splits to get the target number of elements per partition

Smaller partitions where the data is denser



Data Dependencies and Communication

Arrays are relatively large

Every iteration, the new values are needed

Most efficient method is to use multiple processes operating on shared memory

Two arrays, read values from one, write to the other

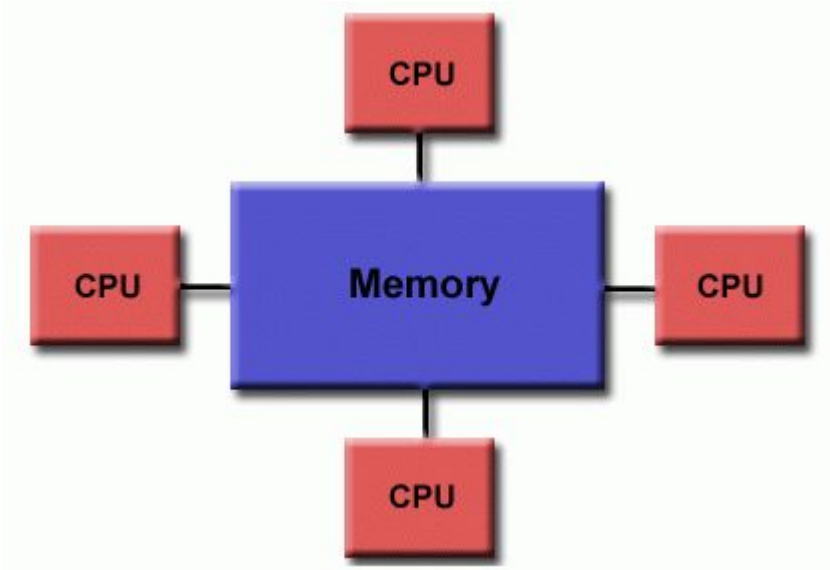


Image Credit: Image Credit: https://hpc.llnl.gov/sites/default/files/styles/no_sidebar_3_up/public/array_proc2.gif?itok=Cict07K0

Parallel Implementation

Used Julia's threaded for loop construct

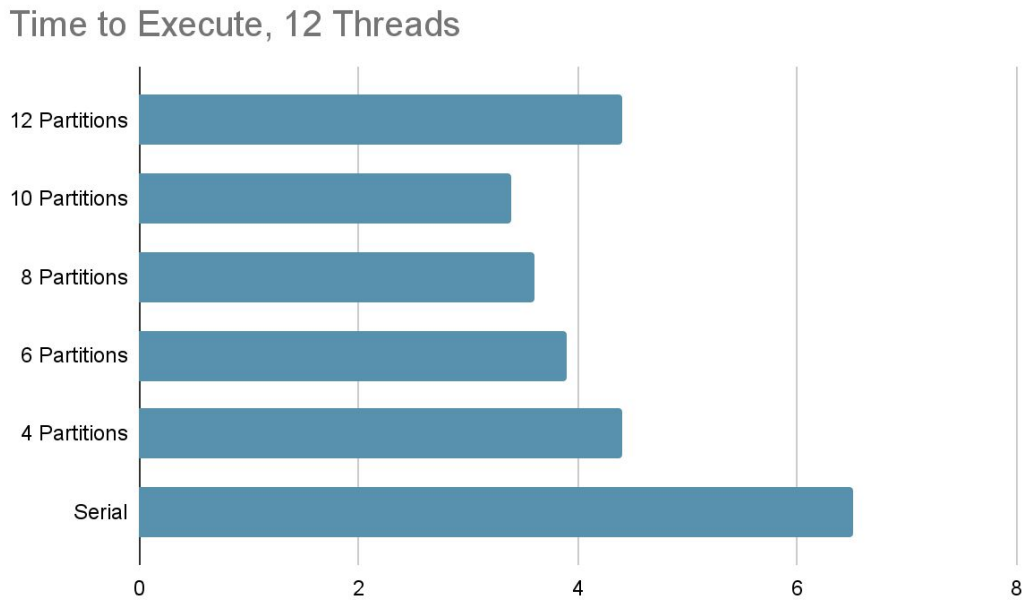
```
while t < tf
    Threads.@threads for i in 1:num_partitions
        rc = partition_heat_pde(my_arr, arr_copy, update_arr2, part_coords[i], Δt)
        if rc == 1
            break
        end
    end
    if rc == 1
        break
    end
    update_arr2 = !update_arr2
    t += Δt
end
```

Comparable C code

This method could be implemented using OpenMP's parallel region and parallel for loop directives

```
#pragma omp parallel
{
  #pragma omp for
  for(int i=0; i < num_partitions; i++)
    partition_heat_pde();
}
```

Runtime Performance



Implementation challenges

Variable resolution grid = no fast matrix operations

Had issues deriving the numerical calculation on a variable resolution grid

$$u^{t+1} = u^t + \Delta t \cdot \left(\frac{\frac{u_e - u}{\Delta x_e} - \frac{u - u_w}{\Delta x_w}}{(\Delta x_e + \Delta x_w)/2} + \frac{\frac{u_n - u}{\Delta x_n} - \frac{u - u_s}{\Delta x_s}}{(\Delta x_n + \Delta x_s)/2} \right)$$

Performance issues

Iterating over a sparse, variable resolution array is difficult, not as efficient as a normal array

Searching for neighbors is computationally expensive

New threads need setup work that reduces the benefits of having an extra thread

Overall slower than a standard method

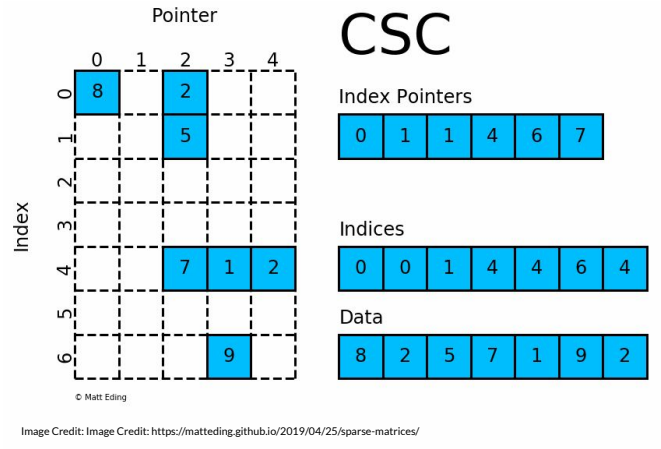


Image Credit: Image Credit: https://matteding.github.io/2019/04/25/sparse-matrices/

```
·        ·        ·        ·        ·        ·        12.0263
·        ·        ·        ·        ·        ·        ·
·        ·        7.32534  ·        24.7338  ·        49.6405
·        ·        ·        ·        ·        ·        172.316
·        ·        24.7338  ·        7.32445  245.003  -281.7
·        ·        ·        ·        245.003  -330.914 545.773
12.0263  ·        49.6405  172.316  -281.7   545.773  -268.923
```

Potential solutions

Refine the algorithms to perform less search work, cache important data

Increase the work per thread so that initialization costs aren't as significant

Conclusions?

The data structures used in a parallel algorithm have significant impact on its performance.

While faster, parallelization is not as effective in this implementation, since the overhead cost is currently too high.

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
9. https://github.com/rdeits/RegionTrees.jl
10. Berg, D. M., Krefeld, V. M., Overmars, M., & Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications*, *Second Edition* (2nd ed.). Springer.

Questions