Parallelization of Successive Over-Relaxation Method

JERED DOMINGUEZ-TRUJILLO

Introduction

Objective:

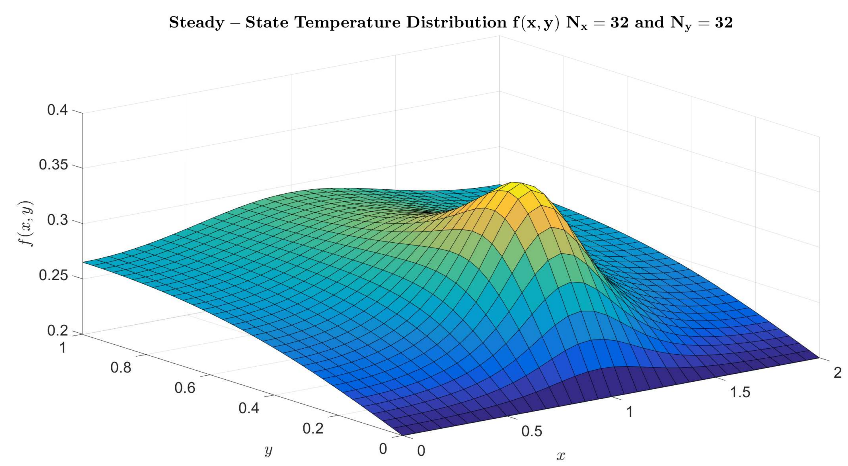
- *Develop a Program that Implements a Parallel Successive Over-Relaxation (SOR) Method to Solve the Steady 2-Dimensional Heat Equation on a Rectangular Grid*

Motivation:

- To Reduce Computation Times
- Enable Larger, More Refined Models to be Computed

Applications:

- Mechanical Engineering
  - Computational Fluid Dynamic Simulations
  - Heat Transfer (Conduction, Convection, Radiation) Problems

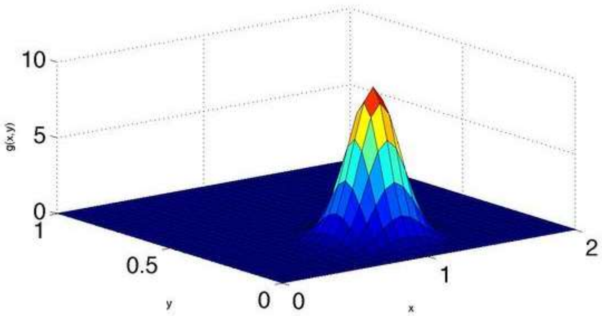


Successive Over-Relaxation is an Iterative Method of Solving Partial Differential Equations

Methods: Heat Equation

Steady Problems:

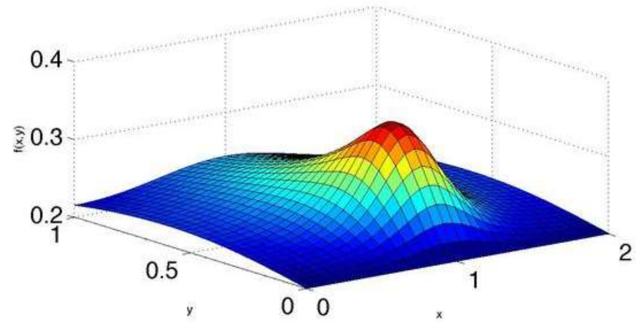
- 2^{nd} Order Partial Differential Equation
- $k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) + S(x, y) = 0$
  - $k = thermal\ conductivity$
  - $T(x, y) = Temperature$
  - $S(x,\ y) = Heat\ Source$



Given Heat Source: S(x, y)

Solving:

- Apply Boundary Conditions
  - Periodic, Constant Temperature/Heat Flux
- Discretize Region
- Discretize Equation
  - $Where\ \frac{\partial^2 T_i}{\partial x^2} \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}$
- Use Numerical (Iterative) Method to Solve



Solution (Steady-State): T(x, y)

Methods: Successive Over-Relaxation

General Equation:

- $x_i^{k+1} = (1-\omega)x_i^k + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j>i} a_{ij}x_j^k - \sum_{j<i} a_{ij}x_j^{k+1}\right), i = 1,2,\ldots,n$

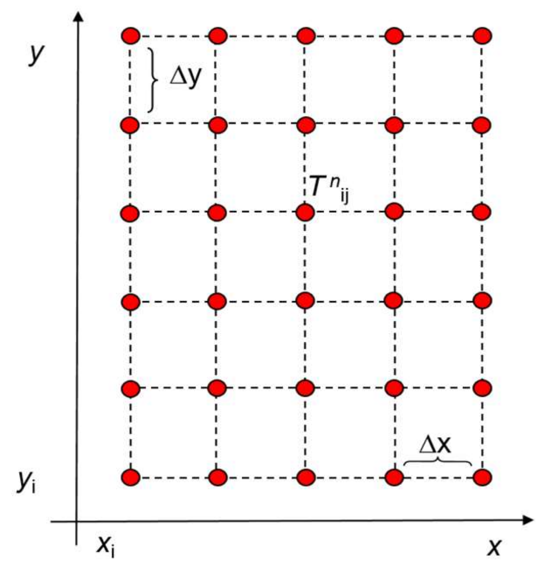
Implementation for Heat Equation:

$$f(x_i, y_j)^{(k+1)} = \frac{2*\left[g(x_i,y_j)*\Delta x^2 + \left[f(x_{i-1},y_j)^{(k+1)} + f(x_{i+1},y_j)^{(k)}\right] + \left[f(x_i,y_{j-1})^{(k)} + f(x_i,y_{j+1})^{(k)}\right]*\frac{\Delta x^2}{\Delta y^2}\right]}{1+\frac{\Delta x^2}{\Delta y^2}}$$

For $i = 1, \ldots, n_x$ and $j = 1, 2, \ldots, n_y$

Where k represents the iteration number, $f(x,y)$ represents $T(x,y)$

And $g(x,y)$ represents $S(x,y)$



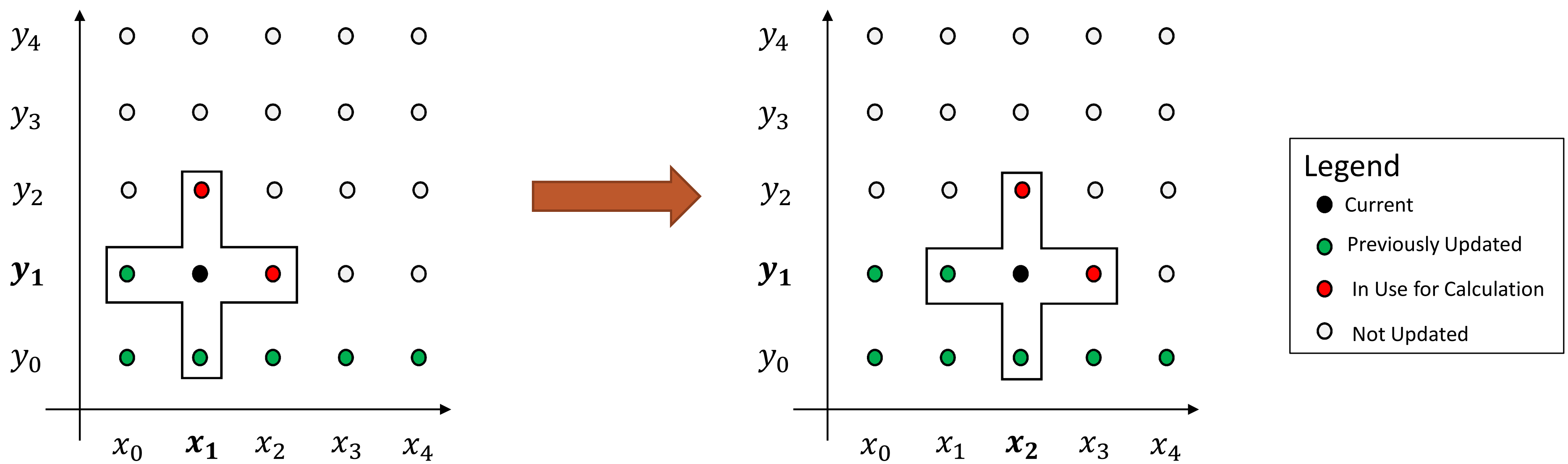
Approach

OpenMP, Multithreading on a Single Process

- Simple, Practical, and Effective

Challenges

- **Data Dependency** within the Grid
- i.e. Each Step through Each Iteration Requires Updated Grid Data



Implementation

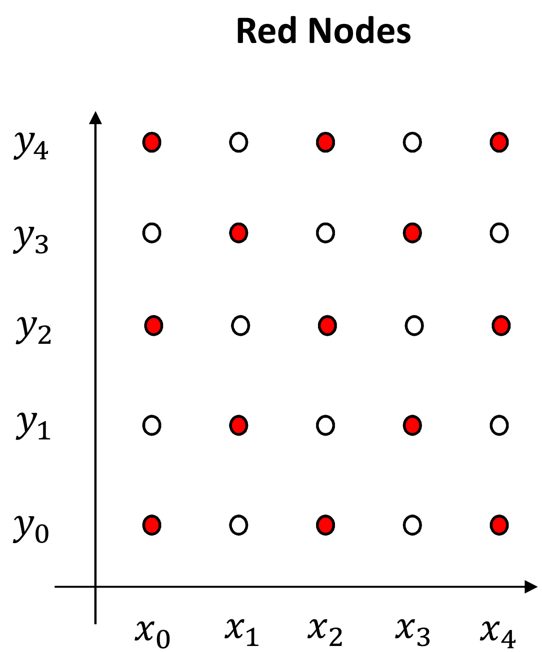
Data-Dependence Solution:

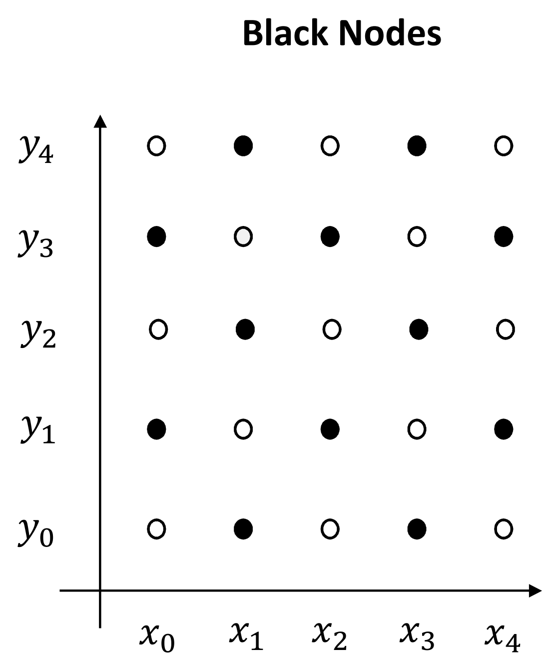
- Red-Black Ordering Method

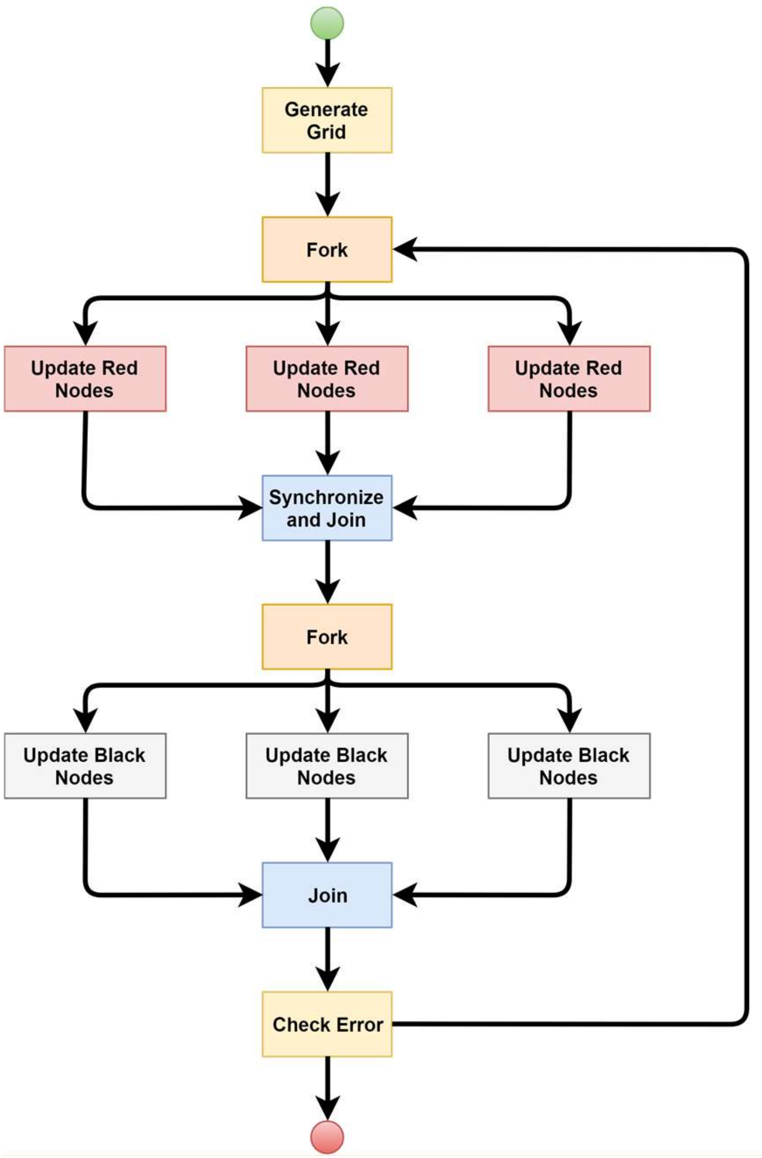
Red-Black Ordering Method:

- First, Update Red Nodes
- Then, Update Black Nodes
- Repeat Until Error is Smaller than Specified Tolerance

Eliminates Data Dependence, Allowing Parallelism to be Implemented







Implementation

Each Iteration has 2 Parallel Regions:

1. Parallel Region 1
- Spawn Threads
- Statically Assign Nodes
- Update Red Nodes

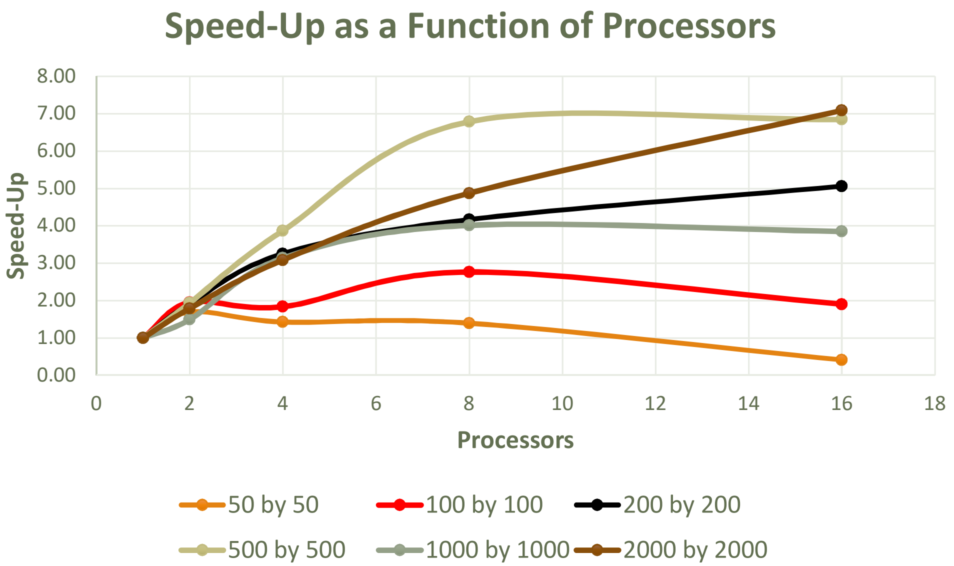
2. Join and Sync Threads with a Thread Barrier

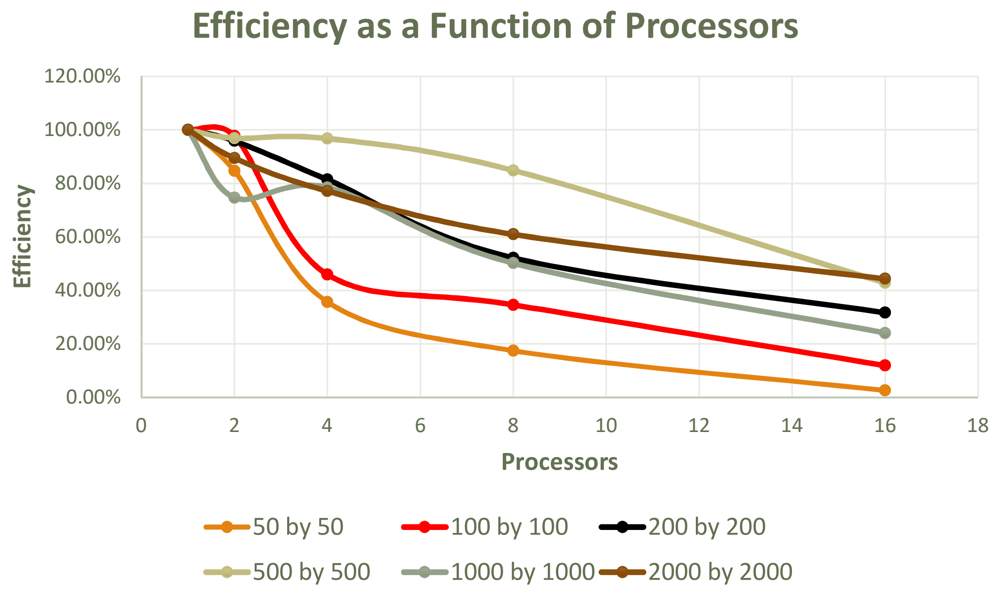
3. Parallel Region 2
- Spawn Threads
- Statically Assign Nodes
- Update Black Nodes

4. Join Threads

5. Iterate Until Error is Below Specified Tolerance

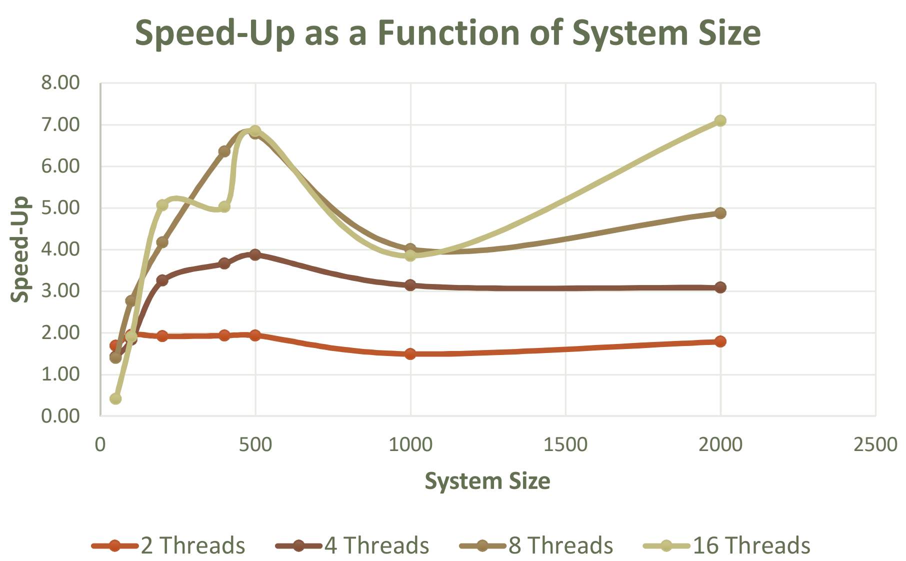
Results: Speed-Up and Efficiency

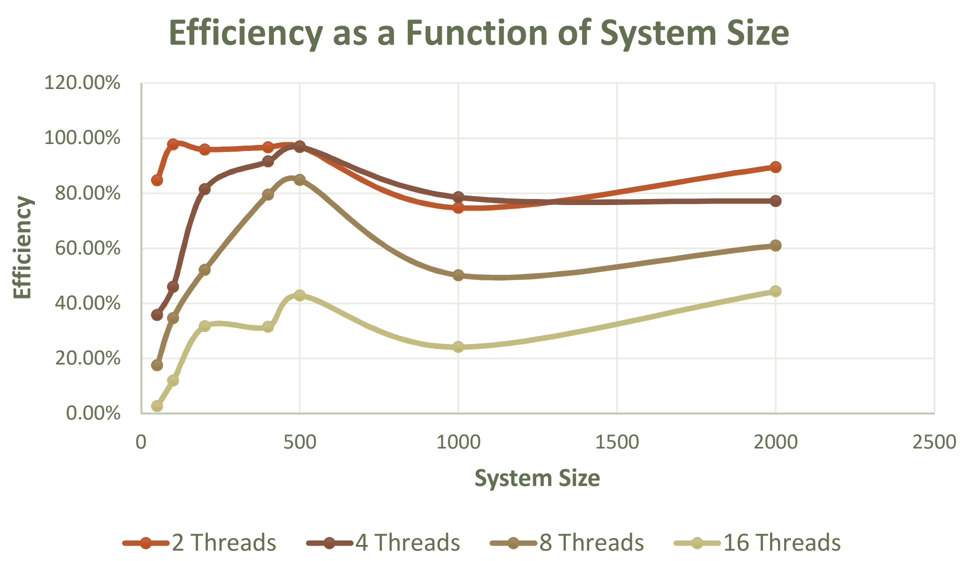




Comments: *Generally, Speed-Up Leveled Off at 16 Threads and Parallel Efficiency Decreased with Additional Threads*

Results: Speed-Up and Efficiency





Comments: *Anomaly at 1000 x 1000 System due to 1.5x Increase in Required Iterations*

Conclusions

Computation Time Increases Quadratically Along with Grid Size

Long Computation Times for Large Systems

- Upwards of an hour

Parallelism Decreased Computation Times to 15 Minutes for a 2000 x 2000 System

Trade-Offs Between

- System Size
- Number of Threads
- Speed-Up
- Efficiency

Must Have Balance for Optimal Performance