Efficiencies of Linear-System-Solution Algorithms in Finding Equilibrium Positions of a 2D Mass-Spring System

Charlie Carver

Systems of Linear Equations

System of Linear Equations

- Collection of interrelated, linear equations
- Solving allows us to find the N unknowns

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \ldots + a_{1,N}x_N &= b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + \ldots + a_{2,N}x_N &= b_2 \\
\ldots & \\
a_{M,1}x_1 + a_{M,2}x_2 + a_{M,3}x_3 + \ldots + a_{M,N}x_N &= b_M
\end{aligned}
$$

Matrix Representation

- Effectively represented as a matrix
  - **A** is an N x M matrix of coefficients
  - **x** is a column vector of N unknowns
  - **b** is a columns vector of M solutions
- Goal: solve for the column matrix **x**

$$\boldsymbol{A}=\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,N} \\ a_{2,1} & a_{2,2} & \dots & a_{2,N} \\ \dots & \dots & \dots & \dots \\ a_{M,1} & a_{M,2} & \dots & a_{M,N} \end{bmatrix} \quad \boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{bmatrix} \quad \boldsymbol{b}=\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_M \end{bmatrix} \quad \boldsymbol{Ax=b}$$

Methods for Solving Linear System Matrices

- Cramer's rule
  - Computationally impractical, determinant is an expensive calculation (O(N!) for Laplace expansion!)

$$a_1 = \frac{\begin{vmatrix} b_{1,1} & A_{1,2} & \dots \\ b_{2,1} & A_{2,2} & \dots \\ \dots & \dots & \dots \end{vmatrix}}{|A|}, \quad a_2 = \dots, \quad a_3 = \dots$$

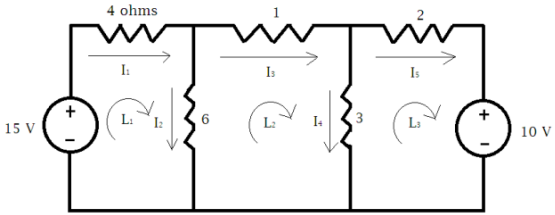
- Augmented matrix row reduction
  - Slow for large matrices
- Matrix inversion
  - Slow if inverse is not required

$$x = A^{-1} b$$

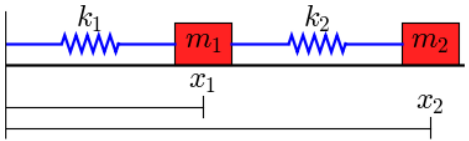
Applications

Physical Applications

- Kirchhoff's laws
  - Find currents passing through resistors



- 1D mass-spring system
  - Find equilibrium positions of many masses connected by many springs

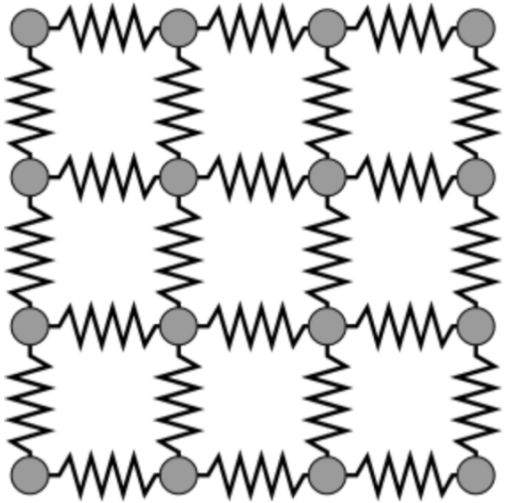


2D Mass-Spring System

- A “sheet” of masses connected by springs
- N2L and Hooke’s Law

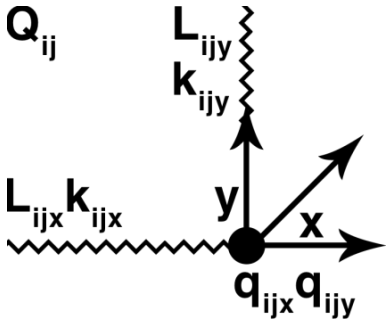
$$\sum F = \sum k_n \Delta X_n = 0$$

- Solution
  - Begin with 1D solution
    - Increase number of masses to see trend
    - General solution for a given row/column
  - Move up to 2D solution



Quadrant System and Data Generation

- Matrix of quadrants
  - Each quadrant has two vectors
    - **L**: natural spring lengths
    - **k**: spring constants



$$\begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1N} \\ Q_{21} & Q_{22} & \cdots & Q_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ Q_{N1} & Q_{N2} & \cdots & Q_{NN} \end{bmatrix}, \quad \boldsymbol{Q}_{ij} = \begin{bmatrix} L_{ij} \\ k_{ij} \end{bmatrix}, \quad \overrightarrow{L_{ij}} = [L_{ijx}\ L_{ijy}], \quad \overrightarrow{k_{ij}} = \ldots$$

- Data is generated for each quadrant
  - Equally spaced masses for tests

2D Matrix Representation

- There are N masses/row and N masses/column
  - n runs from 1 to N
  - Solution for any row can be generalized
  - Solution has swapped indices for each column

$$\boldsymbol{M}_{nx}=\begin{bmatrix} -k_{n1x}-k_{n2x} & k_{n2x} & 0 & \dots \\ k_{n2x} & -k_{n2x}-k_{n3x} & k_{n3x} & \dots \\ 0 & k_{n3x} & -k_{n3x}-k_{n4x} & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix},\quad \boldsymbol{s}_{nx}=\begin{bmatrix} k_{n2x}L_{n2x}-k_{n1x}L_{n2x} \\ k_{n3x}L_{n3x}-k_{n2x}L_{n2x} \\ \dots \\ k_{n(N+1)x}(L_{n(N+1)x}-D_x)-k_{nNx}L_{nNx} \end{bmatrix},\quad \boldsymbol{q}_{nx}=\begin{bmatrix} q_{n1x} \\ q_{n2x} \\ \dots \\ q_{nNx} \end{bmatrix}$$

2D Matrix Representation

- Arranged into a diagonal matrix and column matrices
- Matrix **A** is:
  - (2*N^2) x (2*N^2) in size
  - Square
  - Sparse
  - Diagonal
  - Nonsingular

$$A=\begin{bmatrix}\begin{bmatrix} M_{1x} & 0 & \dots \\ 0 & M_{2x} & \dots \\ \dots & \dots & M_{Nx}\end{bmatrix} & 0 \\ 0 & \begin{bmatrix} M_{1y} & 0 & \dots \\ 0 & M_{2y} & \dots \\ \dots & \dots & M_{Ny}\end{bmatrix}\end{bmatrix}, \quad x=\begin{bmatrix}\begin{bmatrix} q_{1x} \\ \dots \\ q_{Nx}\end{bmatrix} \\ \begin{bmatrix} q_{1y} \\ \dots \\ q_{Ny}\end{bmatrix}\end{bmatrix}, \quad b=\begin{bmatrix}\begin{bmatrix} s_{1x} \\ \dots \\ s_{Nx}\end{bmatrix} \\ \begin{bmatrix} s_{1y} \\ \dots \\ s_{Ny}\end{bmatrix}\end{bmatrix}$$

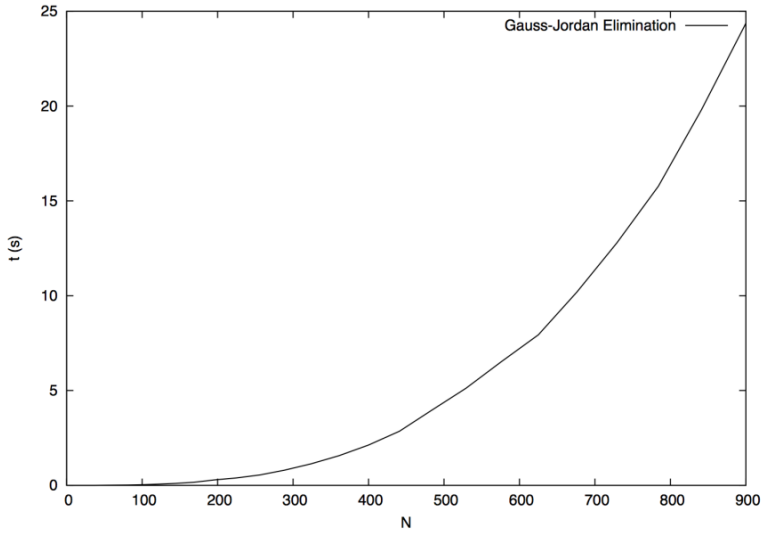
$$A=\begin{bmatrix} M_{1x} & 0 & \dots & 0 & 0 & 0 \\ 0 & M_{2x} & \dots & 0 & 0 & 0 \\ \dots & \dots & M_{Nx} & 0 & 0 & 0 \\ 0 & 0 & 0 & M_{1y} & 0 & \dots \\ 0 & 0 & 0 & 0 & M_{2y} & \dots \\ 0 & 0 & 0 & \dots & \dots & M_{Ny}\end{bmatrix}, \quad x=\begin{bmatrix} q_{1x} \\ \dots \\ q_{Nx} \\ q_{1y} \\ \dots \\ q_{Ny}\end{bmatrix}, \quad b=\begin{bmatrix} s_{1x} \\ \dots \\ s_{Nx} \\ s_{1y} \\ \dots \\ s_{Ny}\end{bmatrix}$$

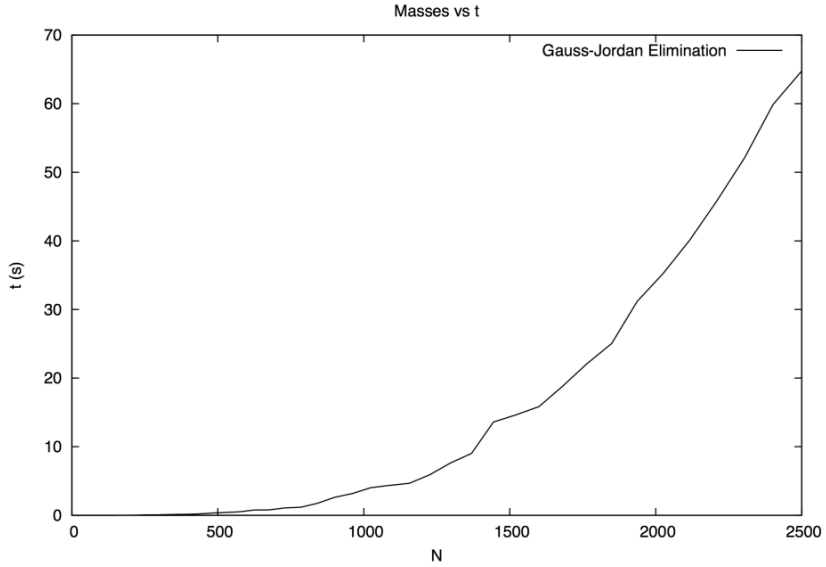
Linear-System-Solution Algorithms

Gauss-Jordan Elimination

- Method
  - Add/subtract linear combinations
  - Left with one unknown per equation
  - Inefficient for large matrices if inverse is not required
  - Complexity is O(N^3)
- Implementation
  - Code from lecture
  - Reduce matrix to an upper triangle and solve the system from the bottom up

Gauss-Jordan Elimination Results





Cubic complexity, over a minute for large matrix of 50^2 masses

Thomas Algorithm

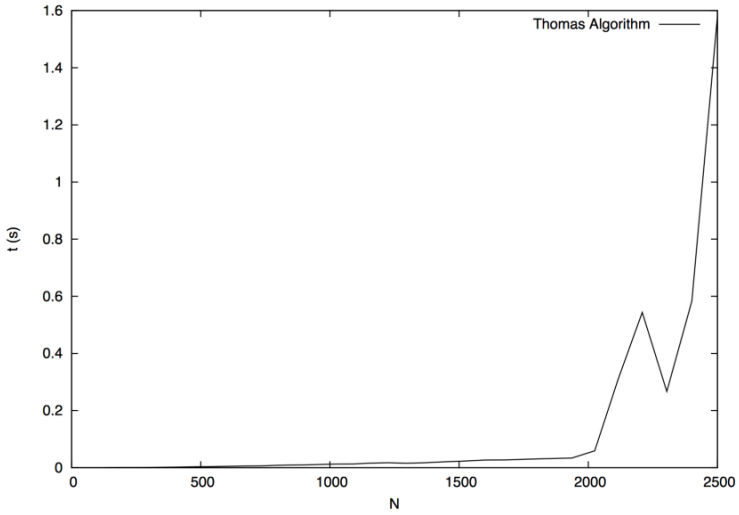
- Method
  - **A** is a sparse, diagonal matrix
  - Can use tridiagonal matrix algorithm
  - Simplified Gaussian elimination
  - Complexity is O(N)
- Implementation
  - Basic algorithm
    - Modify middle band based on top and bottom band
    - Modify solution based on top, middle, and bottom bands
    - Back substitute to solve for x
  - Only needs to store diagonals of size *N* and *N*-1
  - Two loops, one for modifications and one for back substitution

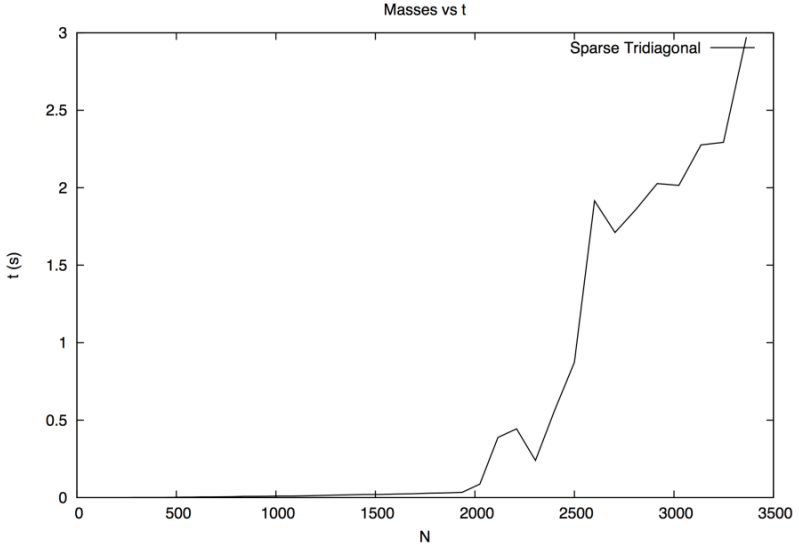
$$m = \frac{a_k}{b_{k-1}}$$
$$b_k = b_k - mc_{k-1}$$
$$d_k = d_k - md_{k-1}$$

$$x_n = \frac{d_n}{b_n}$$

$$x_k = \frac{d_k - c_k x_{k+1}}{b_k}$$

Thomas Algorithm Results





3 seconds for larger matrix, jaggedness not that strange in these types of algorithms

Strassen Algorithm

- Method
  - Matrices can be multiplied using 7 multiplications instead of 8
  - Matrices can be inverted by finding the inverse of only two sub-matrices
  - Consequently, complexity is O(N^2.807)

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \cdot \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix}$$

$$\begin{aligned} Q_0 &\equiv (a_{00} + a_{11}) \times (b_{00} + b_{11}) \\ Q_1 &\equiv (a_{10} + a_{11}) \times b_{00} \\ Q_2 &\equiv a_{00} \times (b_{01} - b_{11}) \\ Q_3 &\equiv a_{11} \times (-b_{00} + b_{10}) \\ Q_4 &\equiv (a_{00} + a_{01}) \times b_{11} \\ Q_5 &\equiv (-a_{00} + a_{10}) \times (b_{00} + b_{01}) \\ Q_6 &\equiv (a_{01} - a_{11}) \times (b_{10} + b_{11}) \end{aligned}$$

$$\begin{aligned} c_{00} &= Q_0 + Q_3 - Q_4 + Q_6 \\ c_{10} &= Q_1 + Q_3 \\ c_{01} &= Q_2 + Q_4 \\ c_{11} &= Q_0 + Q_2 - Q_1 + Q_5 \end{aligned}$$

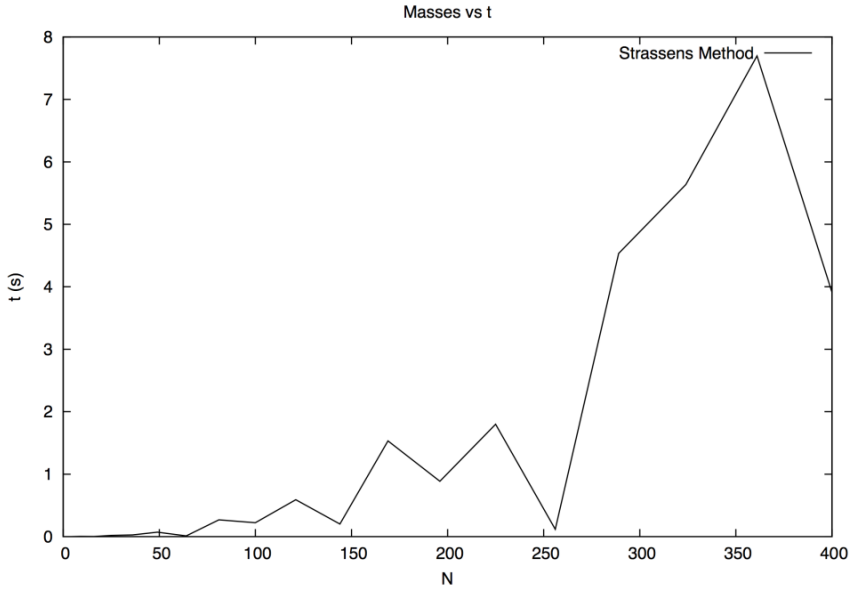
Strassen Method

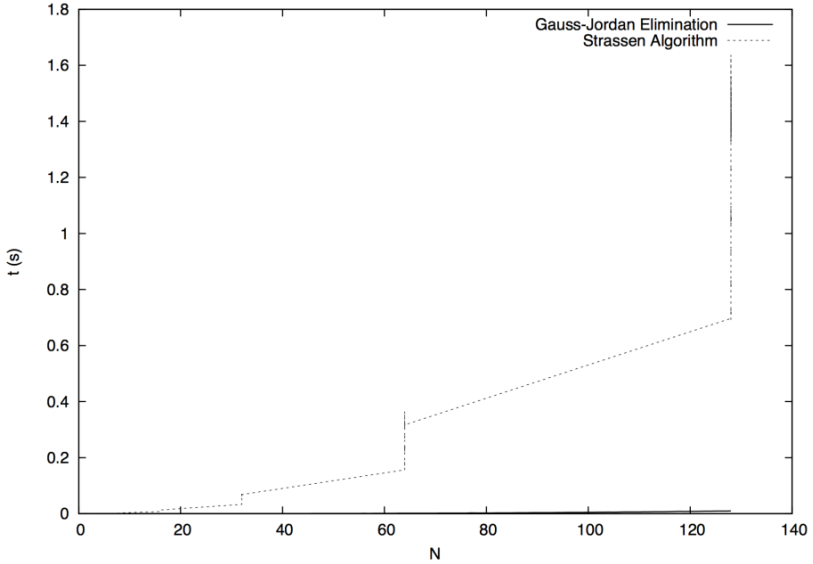
- Padding
  - Pad matrix to a power of 2 for division into square submatrices (like FFT)
- Recursion
  - Recursively partition matrix into square submatrices until the inverse is simply the reciprocal
  - Build back up and move on to multiplication
- Multiplication
  - Strassen multiplication between sub matrices
  - Quick check for zero multiplication
- Recursion
  - Recursively partition matrices into square matrices until multiplication is between scalars
  - Build back up and move on to next step

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \text{ and } \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix}$$

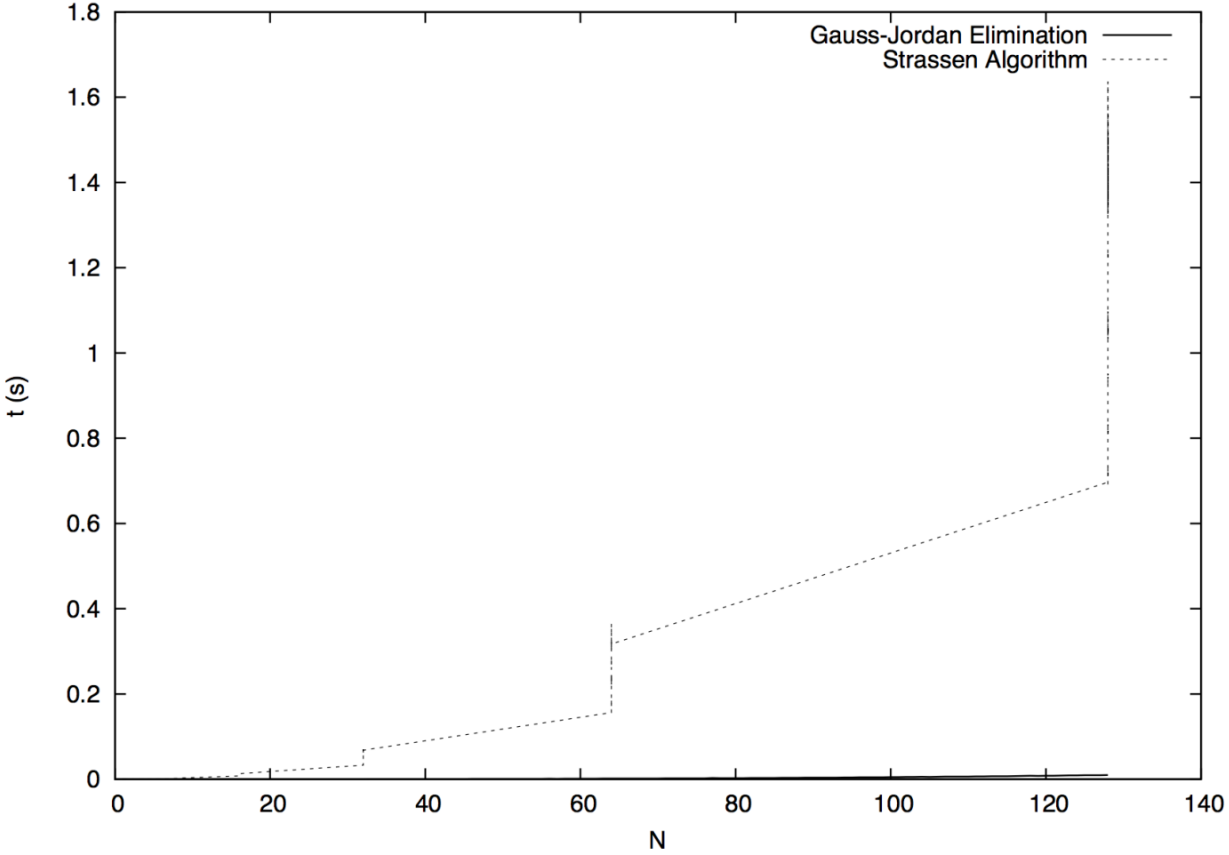
$$R_0 = \text{Inverse}(a_{00})$$
$$R_1 = a_{10} \times R_0$$
$$R_2 = R_0 \times a_{01}$$
$$R_3 = a_{10} \times R_2$$
$$R_4 = R_3 - a_{11}$$
$$R_5 = \text{Inverse}(R_4)$$
$$c_{01} = R_2 \times R_5$$
$$c_{10} = R_5 \times R_1$$
$$R_6 = R_2 \times c_{10}$$
$$c_{00} = R_0 - R_6$$
$$c_{11} = -R_5$$

Strassen Algorithm Results





Slowest method due to padding (overhead in setup), but probably not at large matrix sizes


Gauss-Jordan Elimination
Strassen Algorithm
t (s)
N
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
0
20
40
60
80
100
120
140

Strassen Method Improvements

- Targeted Padding
  - On the fly padding
  - Don’t pad to power of 2
- Algorithm switching
  - Switch to a more efficient method after size threshold (Gauss-Jordan elimination for small matrices)
- Code improvements
  - Make the most out of memory management
  - Use efficient methods for storing and retrieving data
- Parallel computing
  - Run computations in parallel if possible

Conclusions

General Conclusions

- Diagonalized data generation is simple
  - Trivial but room for error
  - Majority of issues were due to off by one errors
- Optimal data modeling is crucial
  - How the diagonal matrix **A** was implemented has an effect on efficiency
- Physics is meticulous
  - The Physics behind the 2D system was simple
  - Took time to find correct pattern
- No one-size-fits-all solution to solving system of linear equations
  - Analyze matrix first then implement appropriate method
  - Place emphasis on matrix size and orientation of matrix
- 2D mass-spring system can be generalized to N dimensions

References

- 1D Spring System. Digital image. Web.
  - http://scipy-cookbook.readthedocs.io/_images/two_springs_diagram.png
- Kirchoff's Matrix. Digital image. Web.
  - http://i.stack.imgur.com/DNVkI.png
- Lecture Notes
- Press, William H. *Numerical Recipes*. Cambridge: Cambridge U Pr., 1992. Print.
- Spring Sheet. Digital image. Web.
  - https://people.eecs.berkeley.edu/~sequin/CS184/TOPICS/SpringMass/Spring_mass_2D.gif
- "Tridiagonal Matrix Algorithm - TDMA (Thomas Algorithm)." *Tridiagonal Matrix Algorithm - TDMA (Thomas Algorithm)*.