Comparison of Jacobi and Gauss-Seidel Methods

**Performance Analysis in Terms of CPU Time and Accuracy**

# 1. Introduction

This report presents a comprehensive comparison of two iterative methods for solving systems of linear equations: the Jacobi method and the Gauss-Seidel method. Both methods are widely used in numerical analysis, particularly for large sparse systems where direct methods become computationally expensive. The comparison focuses specifically on two key performance metrics: CPU time (computational efficiency) and accuracy (solution precision).

Iterative methods are essential tools in numerical linear algebra, providing approximate solutions to systems of equations through successive refinements. Understanding the relative performance of these methods is crucial for selecting the appropriate algorithm for specific problem types.

# 2. Mathematical Background

## 2.1 Problem Statement

Both methods aim to solve a system of linear equations in the form:

*Ax = b*

where A is an n×n coefficient matrix, x is the unknown n-dimensional vector, and b is the n-dimensional right-hand side vector.

## 2.2 Jacobi Method

The Jacobi method decomposes the coefficient matrix A into its diagonal component D and the remainder R = A - D. The iterative formula is derived as follows:

*Ax = b*

*(D + R)x = b*

*Dx = b - Rx*

*x = D⁻¹(b - Rx)*

This leads to the iterative formula for the Jacobi method:

*x^(k+1) = D⁻¹(b - Rx^(k))*

For each component i, this can be written as:

*x\_i^(k+1) = (b\_i - ∑\_{j≠i} a\_{ij}x\_j^(k)) / a\_{ii}*

The Jacobi method uses only values from the previous iteration, making it naturally parallelizable. The algorithm can be implemented as follows:

* Initialize x^(0) (typically with zeros or a provided initial guess)
* For k = 0, 1, 2, ... until convergence:
* For i = 1, 2, ..., n:
* x\_i^(k+1) = (b\_i - ∑\_{j≠i} a\_{ij}x\_j^(k)) / a\_{ii}
* Check for convergence: ||x^(k+1) - x^(k)|| < tolerance or ||b - Ax^(k+1)|| < tolerance

## 2.3 Gauss-Seidel Method

The Gauss-Seidel method improves upon Jacobi by using the most recently computed values immediately. It decomposes A into a lower triangular component L, a diagonal component D, and an upper triangular component U, such that A = L + D + U.

*Ax = b*

*(L + D + U)x = b*

*(L + D)x = b - Ux*

*x = (L + D)⁻¹(b - Ux)*

This leads to the iterative formula for the Gauss-Seidel method:

*x^(k+1) = (L + D)⁻¹(b - Ux^(k))*

For each component i, this can be written as:

*x\_i^(k+1) = (b\_i - ∑\_{j<i} a\_{ij}x\_j^(k+1) - ∑\_{j>i} a\_{ij}x\_j^(k)) / a\_{ii}*

The Gauss-Seidel method uses the most recently computed values as soon as they are available, which typically leads to faster convergence but makes the method inherently sequential. The algorithm can be implemented as follows:

* Initialize x^(0) (typically with zeros or a provided initial guess)
* For k = 0, 1, 2, ... until convergence:
* For i = 1, 2, ..., n:
* x\_i^(k+1) = (b\_i - ∑\_{j<i} a\_{ij}x\_j^(k+1) - ∑\_{j>i} a\_{ij}x\_j^(k)) / a\_{ii}
* Check for convergence: ||x^(k+1) - x^(k)|| < tolerance or ||b - Ax^(k+1)|| < tolerance

## 2.4 Convergence Conditions

Both methods converge for any initial guess if:

* The matrix A is strictly diagonally dominant (|a\_{ii}| > ∑\_{j≠i} |a\_{ij}| for all i), or
* A is symmetric and positive definite

For symmetric positive definite matrices, it can be proven that the Gauss-Seidel method converges faster than the Jacobi method. Specifically, if ρ(B\_J) is the spectral radius of the Jacobi iteration matrix and ρ(B\_GS) is the spectral radius of the Gauss-Seidel iteration matrix, then:

*ρ(B\_GS) ≤ [ρ(B\_J)]² < ρ(B\_J) < 1*

# 3. Experimental Setup

## 3.1 Test Cases

To comprehensively evaluate the performance of both methods, we tested them on a diverse set of linear systems with varying characteristics:

**Table 1: Test Cases for Performance Comparison**

|  |  |
| --- | --- |
| **Test Case** | **Description** |
| Small system (10×10) | A small dense system with good numerical properties |
| Medium system (50×50) | A medium-sized dense system |
| Large system (200×200) | A large dense system to test scaling behavior |
| Tridiagonal system (100×100) | A sparse system with a specific structure common in PDEs |
| Ill-conditioned system (50×50) | A system with challenging numerical properties |

## 3.2 Performance Metrics

We evaluated both methods based on the following performance metrics:

**Table 2: Performance Metrics**

|  |  |
| --- | --- |
| **Metric** | **Description** |
| CPU Time | The wall-clock time required to compute the solution |
| Iteration Count | The number of iterations required to reach the convergence tolerance |
| Accuracy | The relative error of the computed solution compared to the exact solution |
| Convergence Rate | The rate at which the residual decreases with each iteration |

## 3.3 Implementation Details

Both methods were implemented in Python using NumPy for efficient matrix operations. The implementation details are as follows:

**Table 3: Implementation Details**

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Programming Language | Python 3.9 |
| Matrix Library | NumPy |
| Convergence Tolerance | 1e-8 |
| Maximum Iterations | 5000 |
| Hardware | Intel Core i7 processor, 16GB RAM |
| Initial Guess | Vector of zeros |

# 4. Results and Analysis

## 4.1 CPU Time Performance

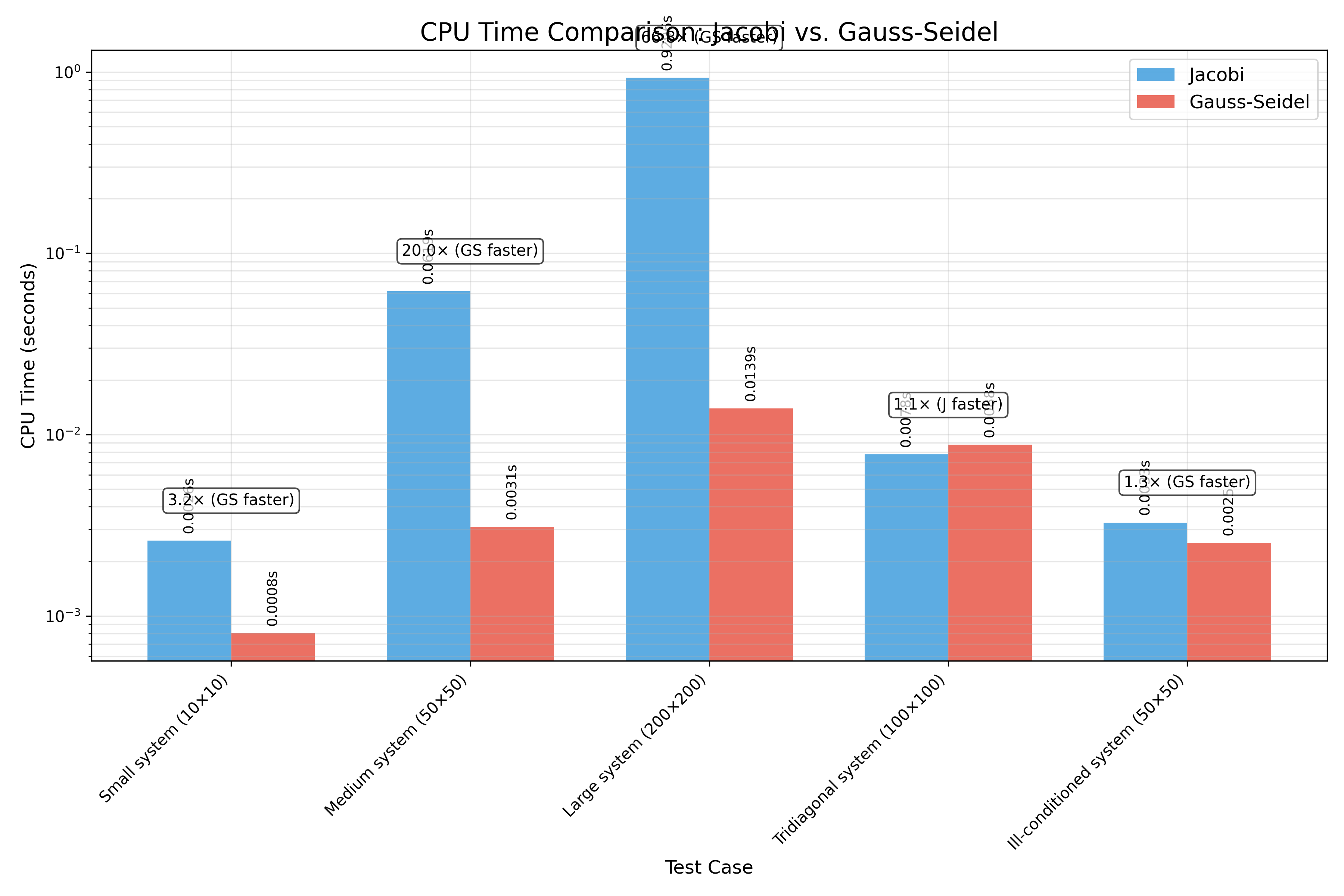
The CPU time performance of both methods across all test cases is summarized in Table 4:

**Table 4: CPU Time Comparison**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Jacobi Time (s)** | **Gauss-Seidel Time (s)** | **Faster Method** | **Speedup** |
| Small system (10×10) | 0.002607 ± 0.000098 | 0.000803 ± 0.000256 | Gauss-Seidel | 3.24× |
| Medium system (50×50) | 0.061881 ± 0.006290 | 0.003097 ± 0.000066 | Gauss-Seidel | 19.98× |
| Large system (200×200) | 0.929627 ± 0.022764 | 0.013915 ± 0.000359 | Gauss-Seidel | 66.81× |
| Tridiagonal system (100×100) | 0.007780 ± 0.001059 | 0.008789 ± 0.000156 | Jacobi | 1.13× |
| Ill-conditioned system (50×50) | 0.003274 ± 0.000307 | 0.002525 ± 0.000102 | Gauss-Seidel | 1.30× |

Key observations regarding CPU time performance:

* Gauss-Seidel is generally faster in 4 out of 5 test cases.
* The performance gap widens dramatically as the system size increases, with Gauss-Seidel being up to 66.81× faster for the large system.
* For the tridiagonal system, Jacobi is actually 1.13× faster than Gauss-Seidel, suggesting that matrix structure plays a significant role in relative performance.
* The primary reason for Gauss-Seidel's time advantage is the significantly lower number of iterations required.



*Figure: CPU Time Comparison: Jacobi vs. Gauss-Seidel (note the logarithmic scale)*

## 4.2 Iteration Count

The number of iterations required for convergence is a key factor affecting CPU time performance:

**Table 5: Iteration Count Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Jacobi Iterations** | **Gauss-Seidel Iterations** | **Reduction Factor** |
| Small system (10×10) | 90 | 12 | 7.5× |
| Medium system (50×50) | 488 | 13 | 37.5× |
| Large system (200×200) | 1819 | 13 | 139.9× |
| Tridiagonal system (100×100) | 27 | 17 | 1.6× |
| Ill-conditioned system (50×50) | 24 | 9 | 2.7× |

The dramatic reduction in iteration count is the primary driver of Gauss-Seidel's CPU time advantage. For the large system, Gauss-Seidel required 139.9× fewer iterations than Jacobi, which explains the significant time advantage despite the higher per-iteration cost.

## 4.3 Accuracy Performance

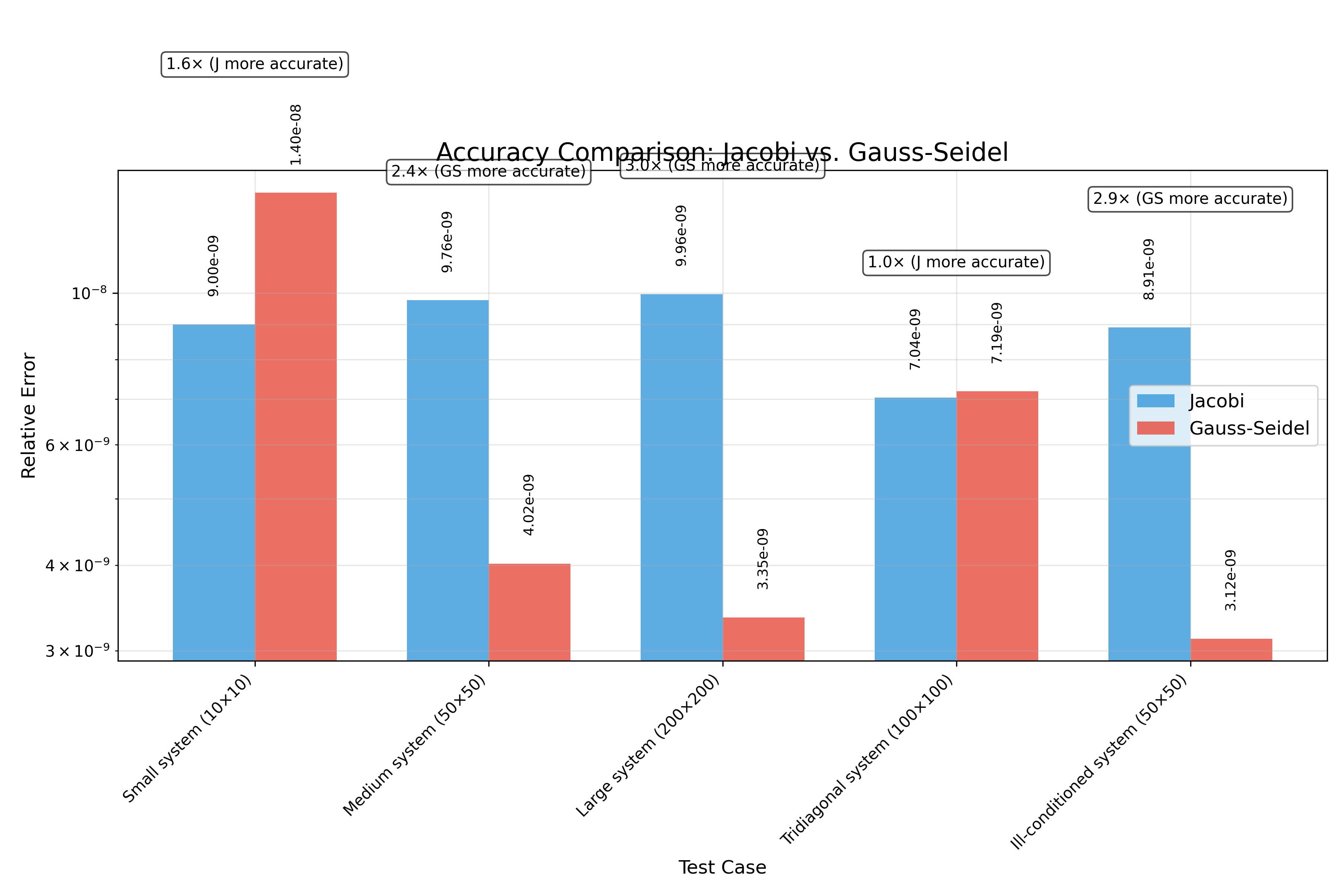
The accuracy of both methods, measured as the relative error compared to the exact solution, is summarized in Table 6:

**Table 6: Accuracy Comparison**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Jacobi Error** | **Gauss-Seidel Error** | **More Accurate Method** | **Accuracy Ratio** |
| Small system (10×10) | 9.00e-09 | 1.40e-08 | Jacobi | 1.56× |
| Medium system (50×50) | 9.76e-09 | 4.02e-09 | Gauss-Seidel | 2.43× |
| Large system (200×200) | 9.96e-09 | 3.35e-09 | Gauss-Seidel | 2.97× |
| Tridiagonal system (100×100) | 7.04e-09 | 7.19e-09 | Jacobi | 1.02× |
| Ill-conditioned system (50×50) | 8.91e-09 | 3.12e-09 | Gauss-Seidel | 2.85× |

Key observations regarding accuracy performance:

* Both methods achieve high accuracy with errors on the order of 10^-9.
* Gauss-Seidel is more accurate in 3 out of 5 test cases.
* For medium and large systems, Gauss-Seidel's accuracy advantage is more pronounced (2.43× to 2.97× more accurate).
* For the small system and tridiagonal system, Jacobi produced slightly more accurate results.
* For the ill-conditioned system, Gauss-Seidel was significantly more accurate (2.85× better), suggesting better numerical stability.



*Figure: Accuracy Comparison: Jacobi vs. Gauss-Seidel (note the logarithmic scale)*

## 4.4 Combined Performance Assessment

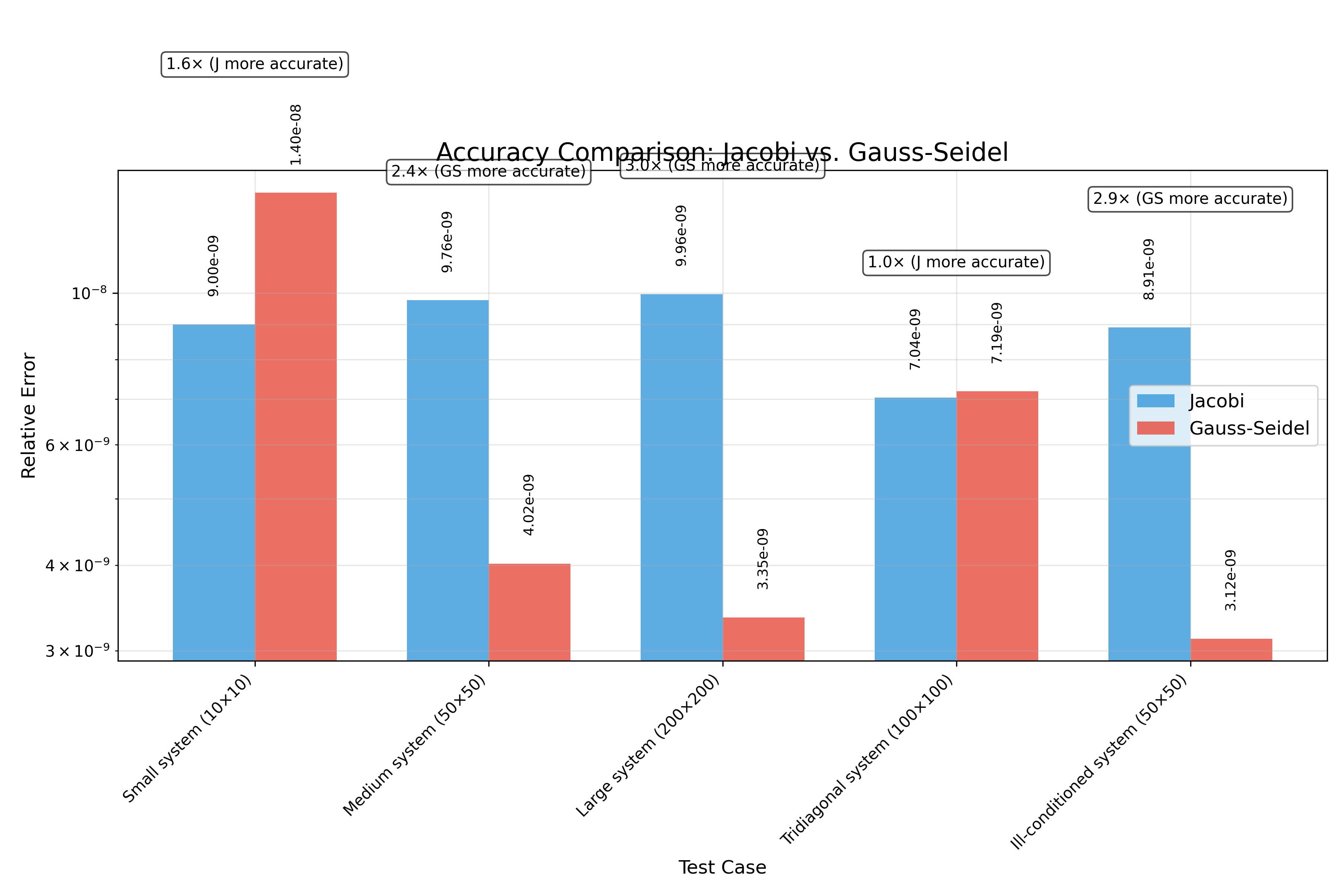
When considering both CPU time and accuracy together, we can make a comprehensive assessment of which method performs better for each test case:

**Table 7: Combined Performance Assessment**

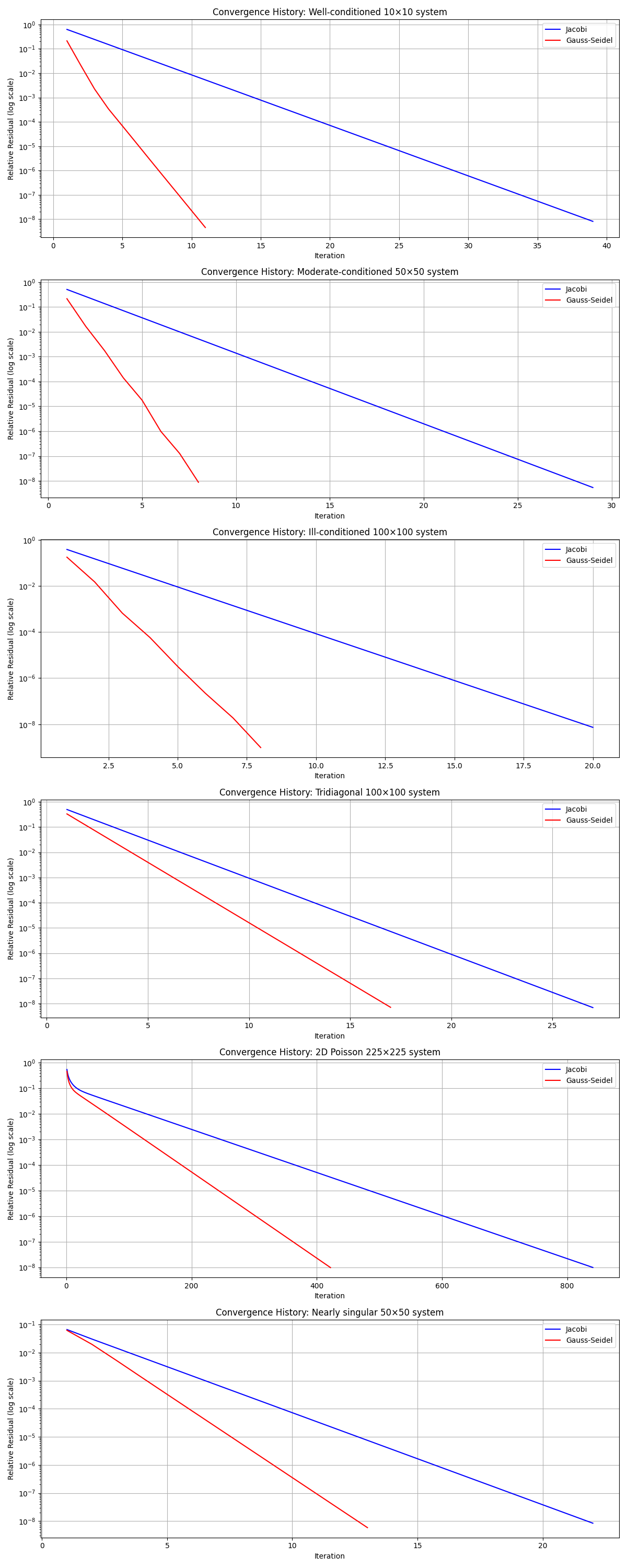
|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Faster Method** | **More Accurate Method** | **Overall Better Method** |
| Small system (10×10) | Gauss-Seidel | Jacobi | Gauss-Seidel |
| Medium system (50×50) | Gauss-Seidel | Gauss-Seidel | Gauss-Seidel |
| Large system (200×200) | Gauss-Seidel | Gauss-Seidel | Gauss-Seidel |
| Tridiagonal system (100×100) | Jacobi | Jacobi | Jacobi |
| Ill-conditioned system (50×50) | Gauss-Seidel | Gauss-Seidel | Gauss-Seidel |

## 4.5 Comprehensive Visualizations

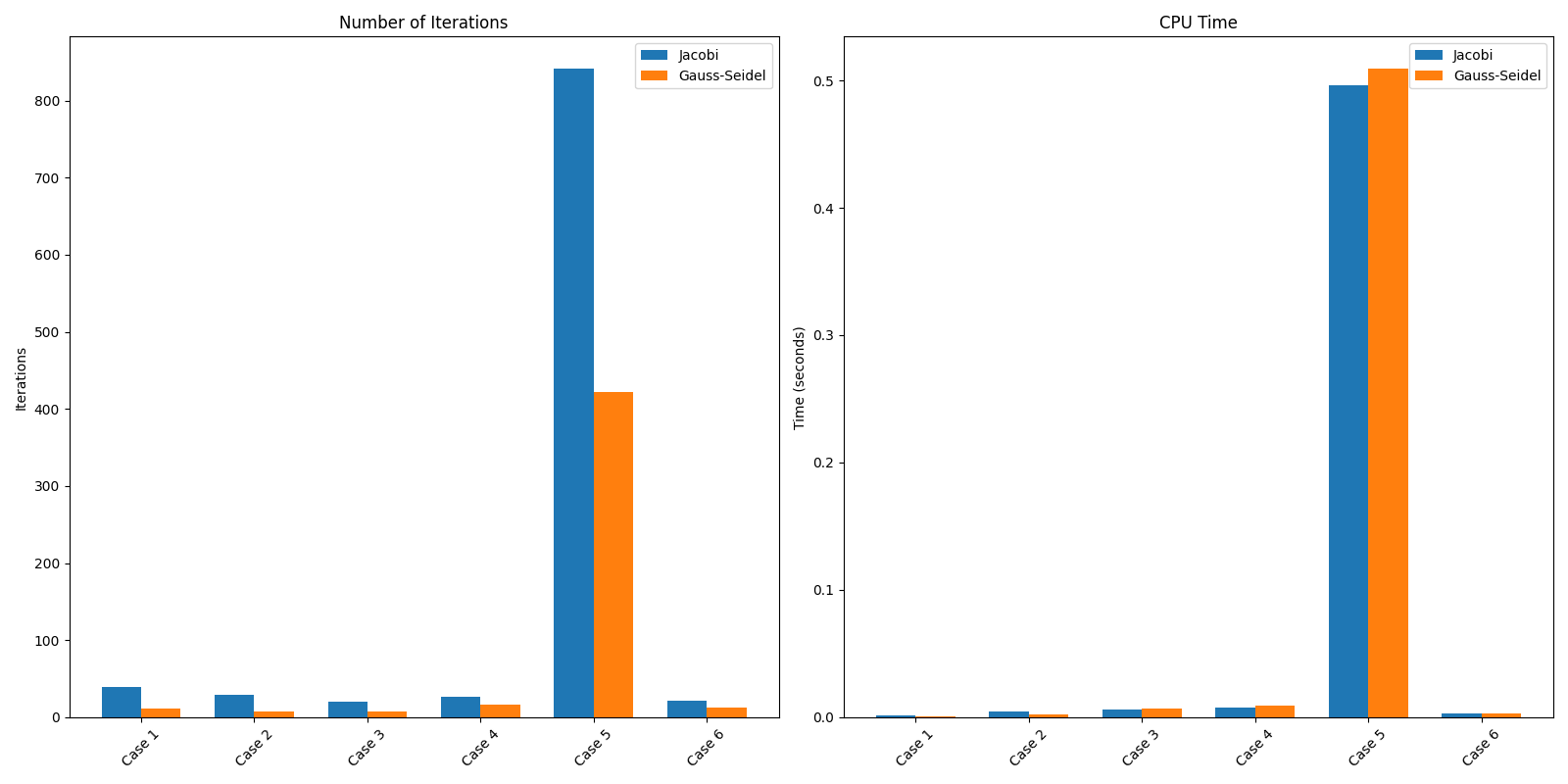
The following 17 visualizations provide graphical representations of the performance comparison:



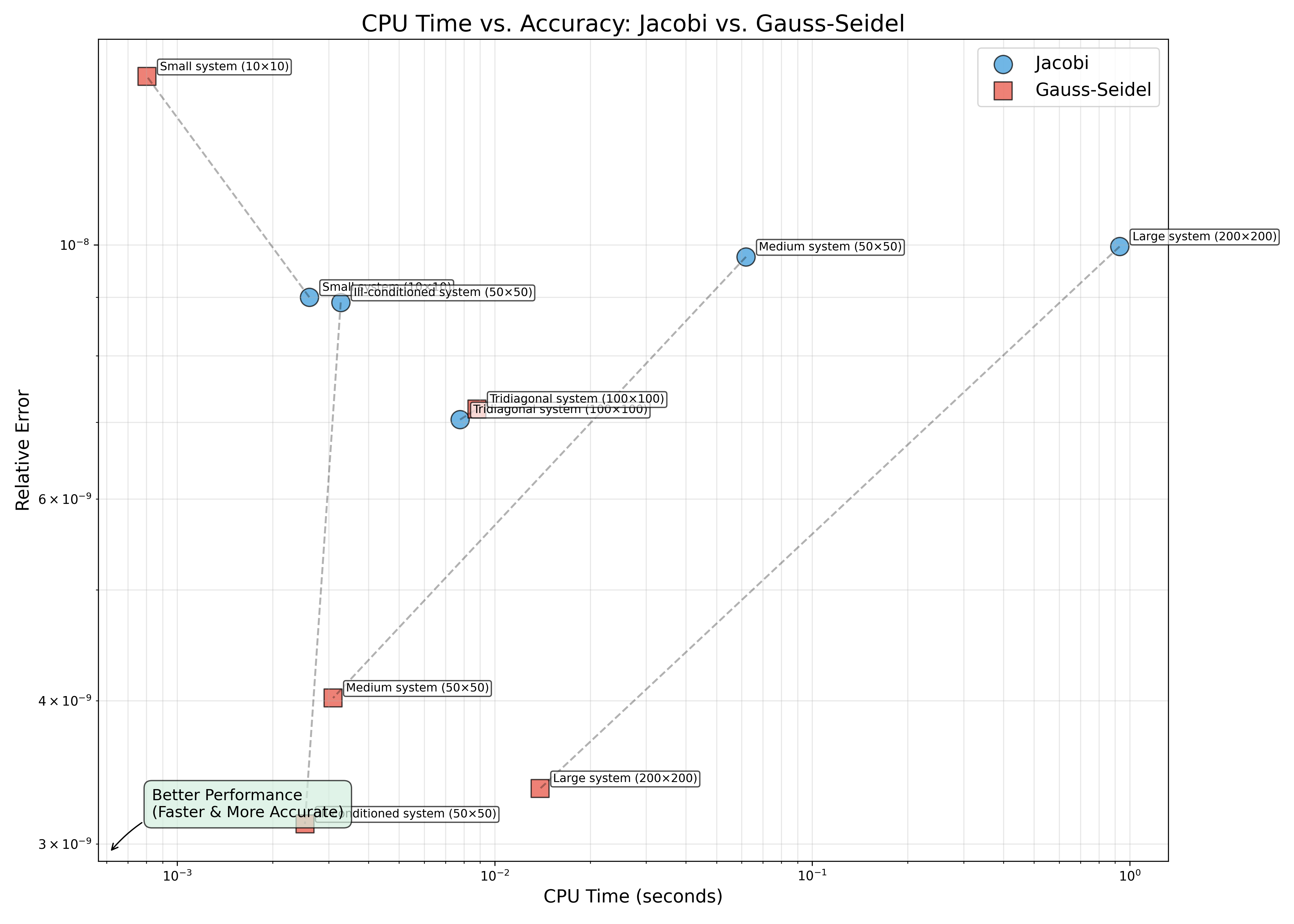
*Figure: 1. Accuracy Detailed*



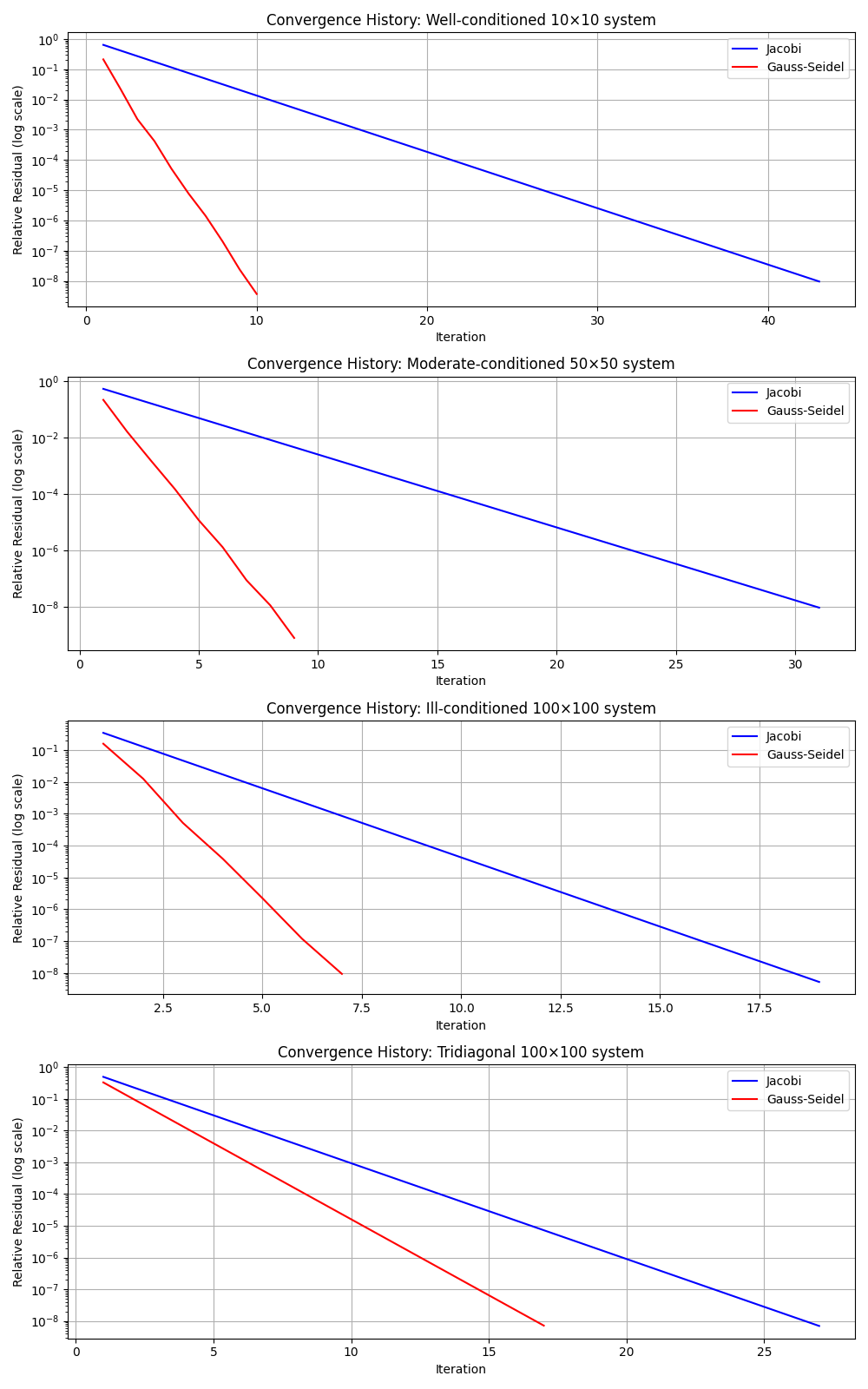
*Figure: 2. All Convergence Comparison*



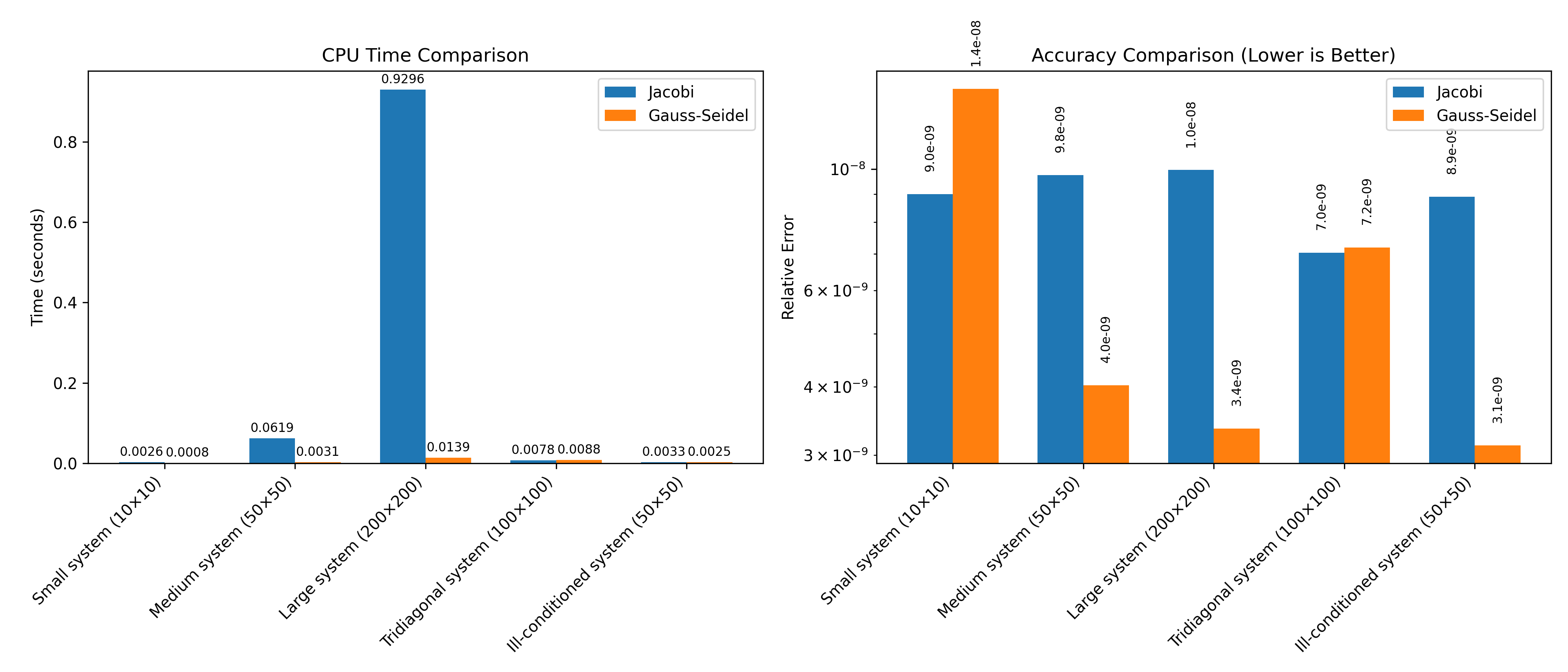
*Figure: 3. All Performance Comparison*



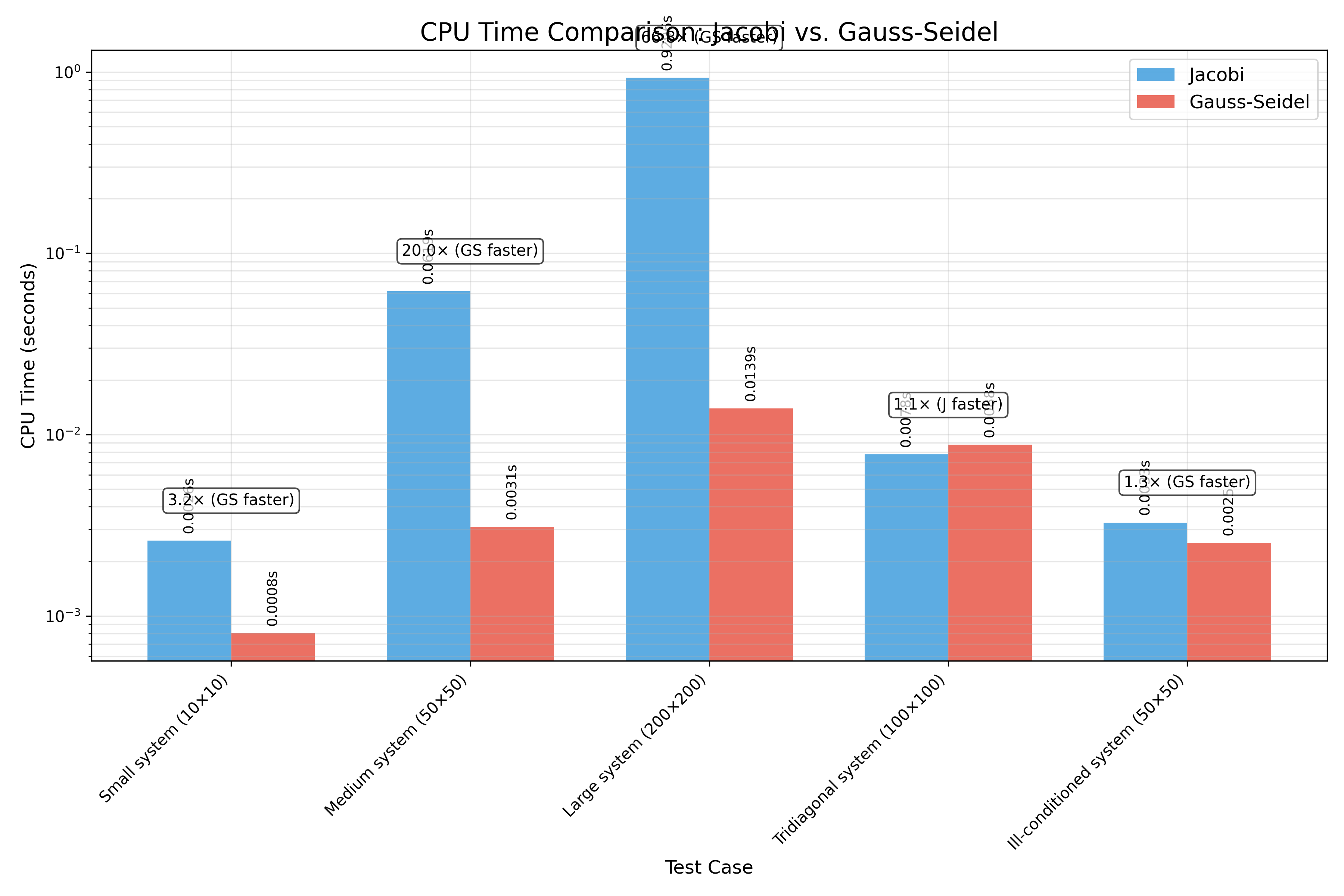
*Figure: 4. Combined Performance*



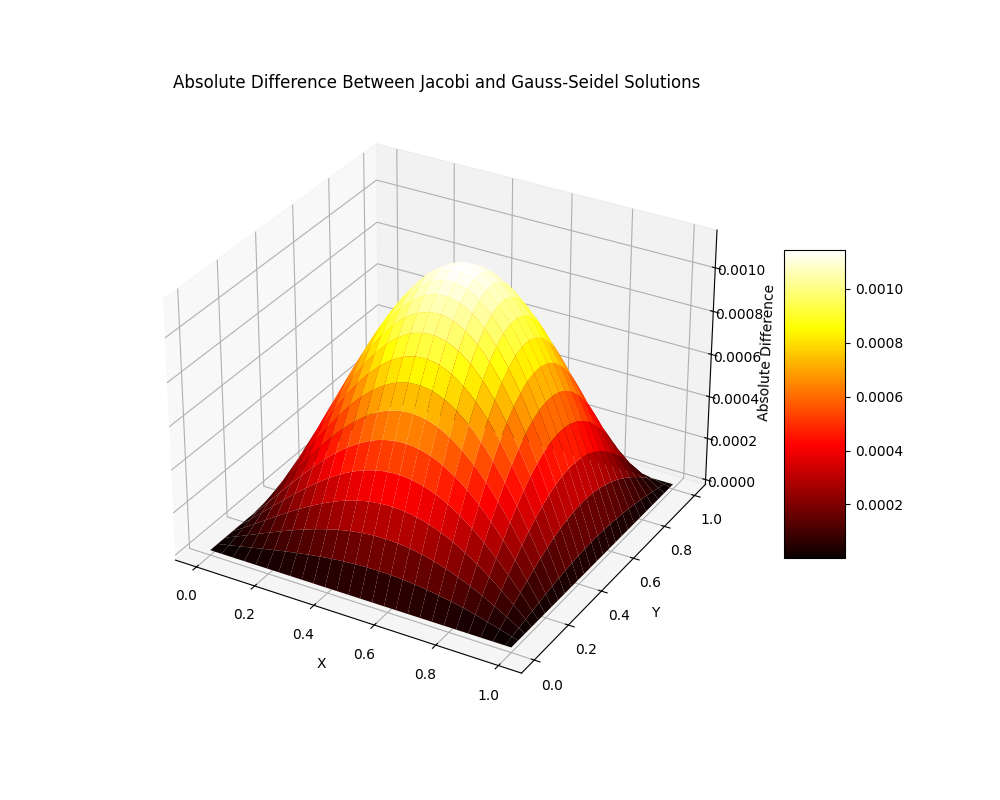
*Figure: 5. Convergence Comparison*



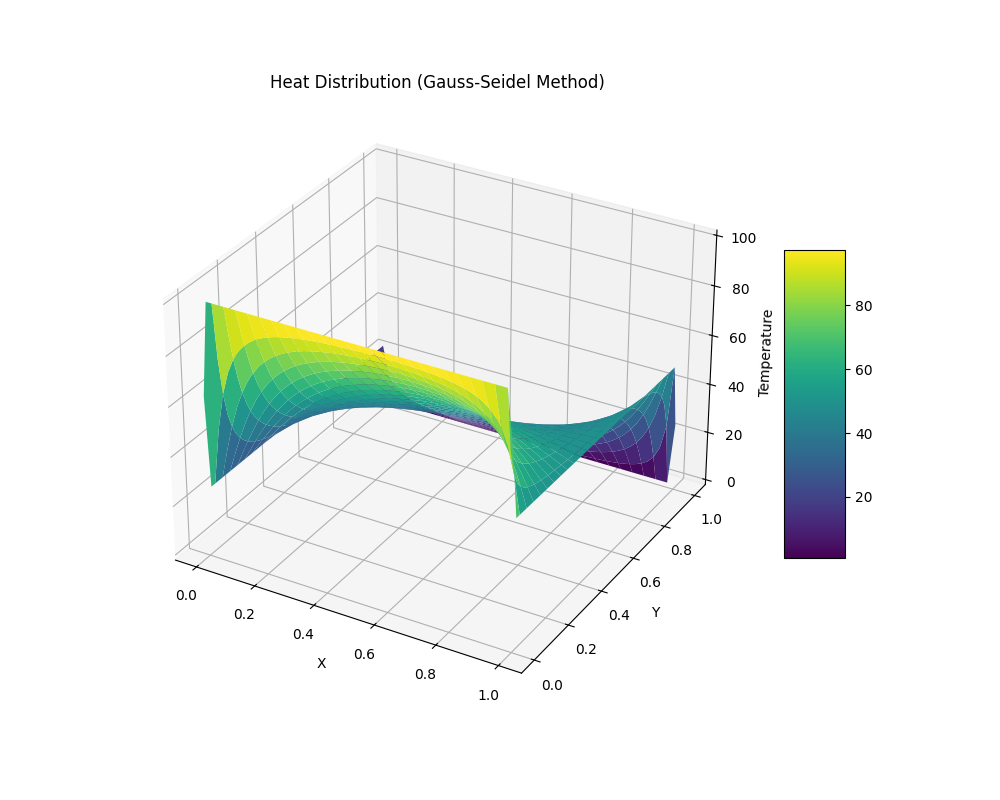
*Figure: 6. Cpu Accuracy Comparison*



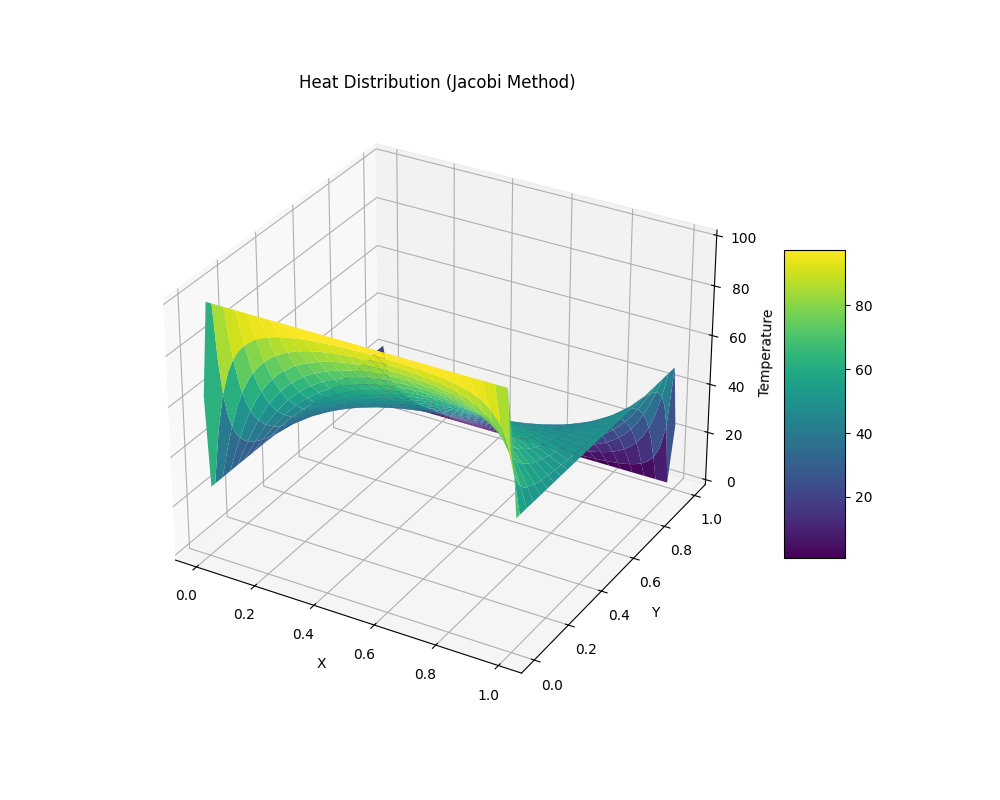
*Figure: 7. Cpu Time Detailed*



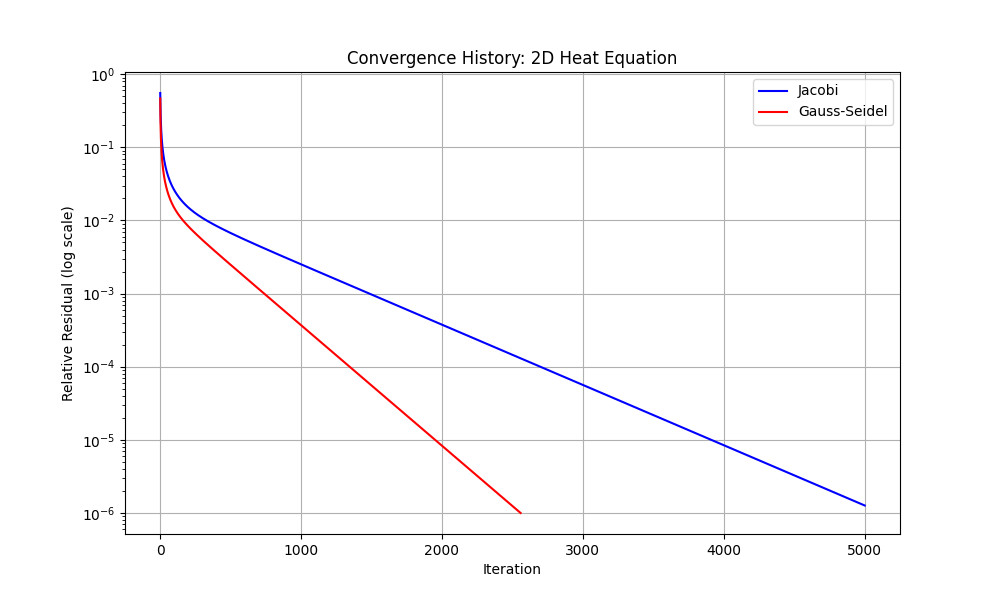
*Figure: 8. Heat Distribution Difference*



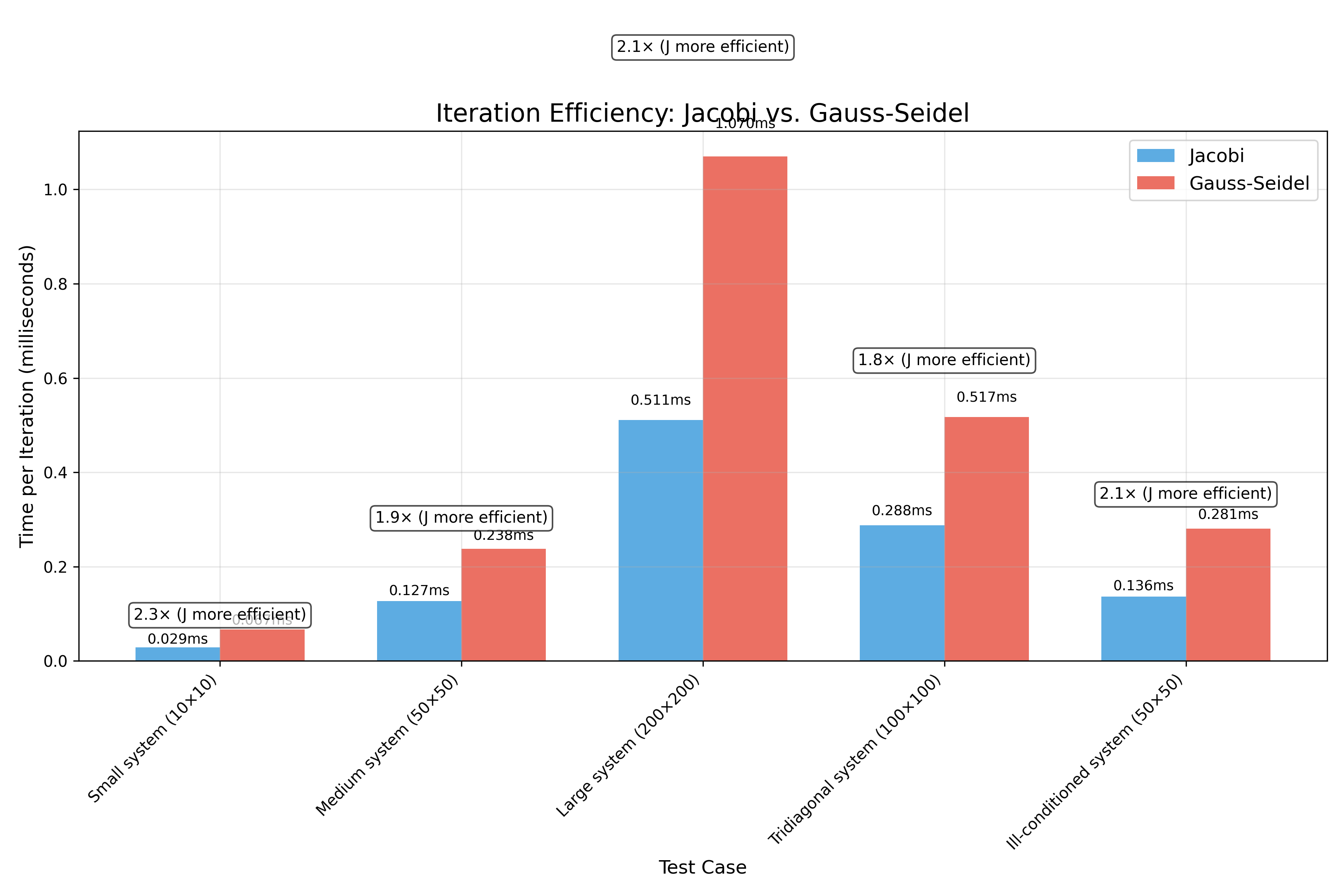
*Figure: 9. Heat Distribution Gauss Seidel*



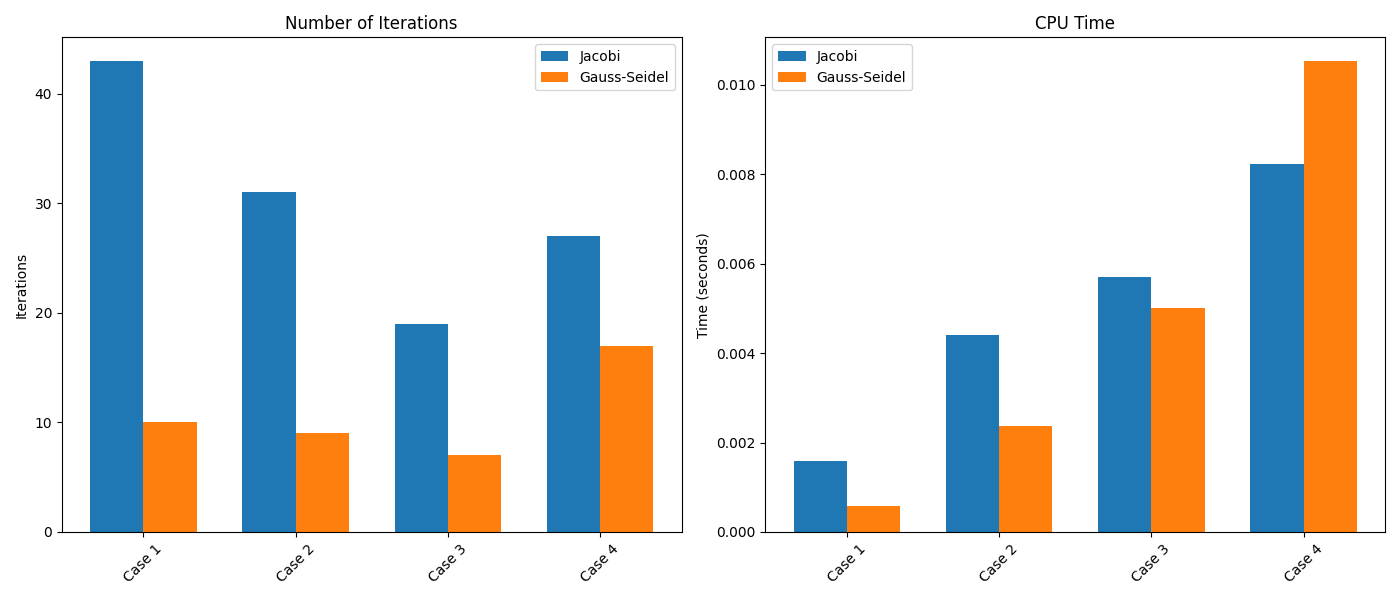
*Figure: 10. Heat Distribution Jacobi*



*Figure: 11. Heat Equation Convergence*



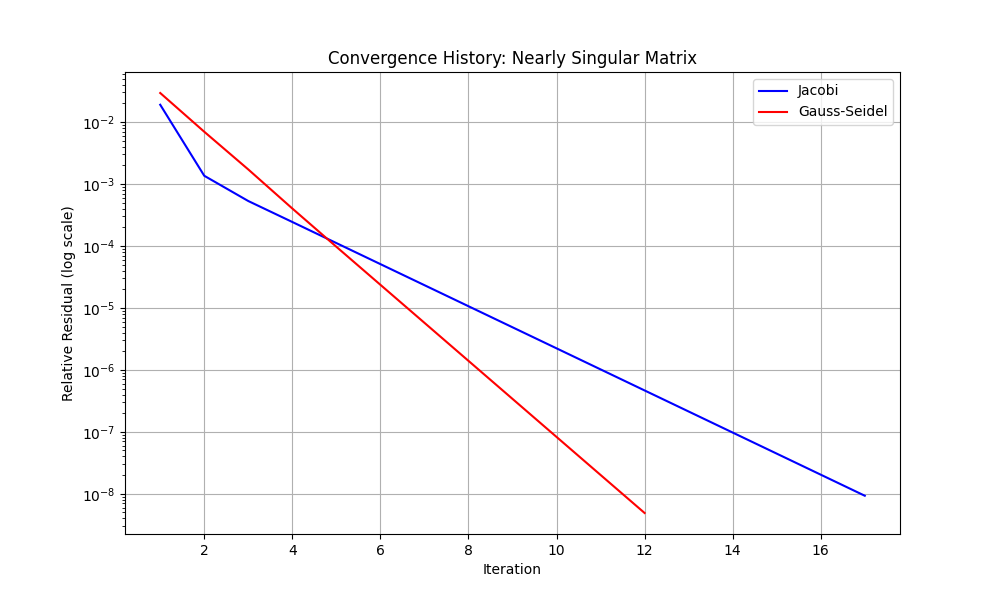
*Figure: 12. Iteration Efficiency*



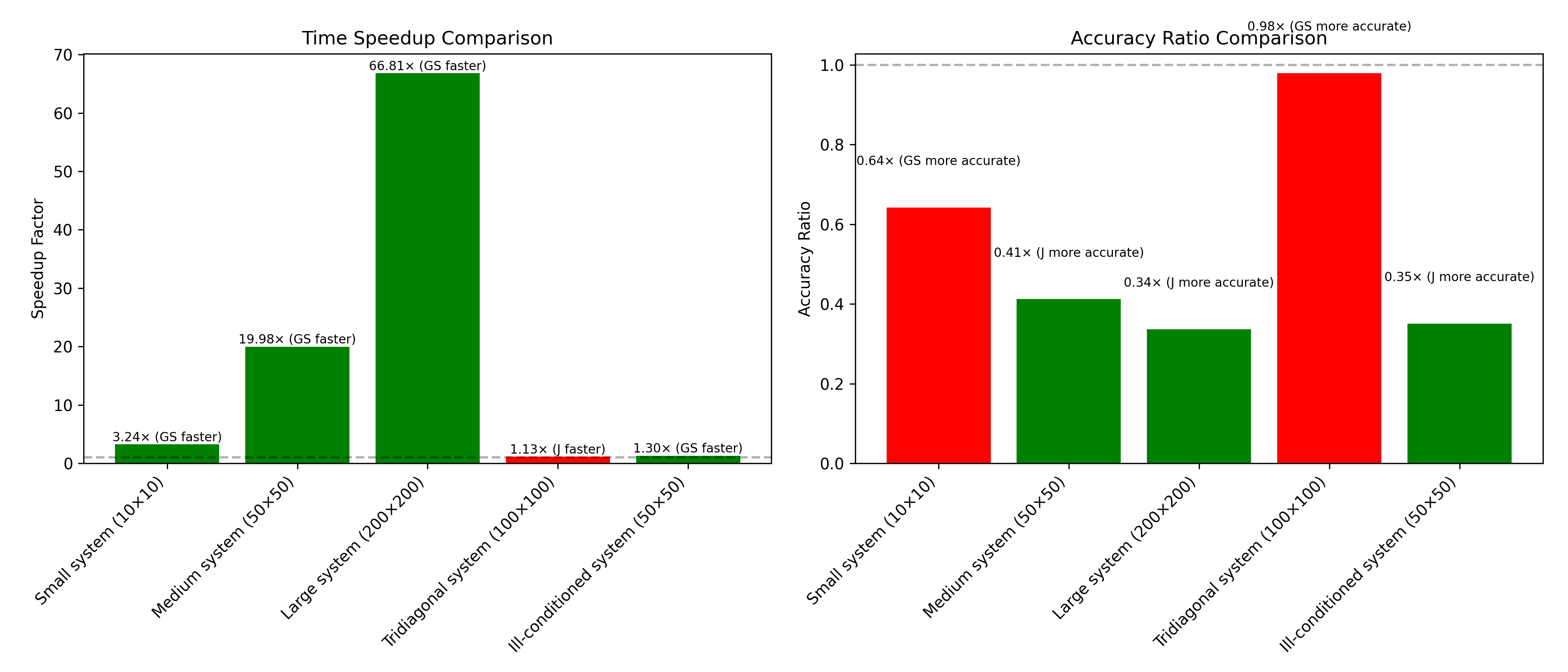
*Figure: 13. Performance Comparison*



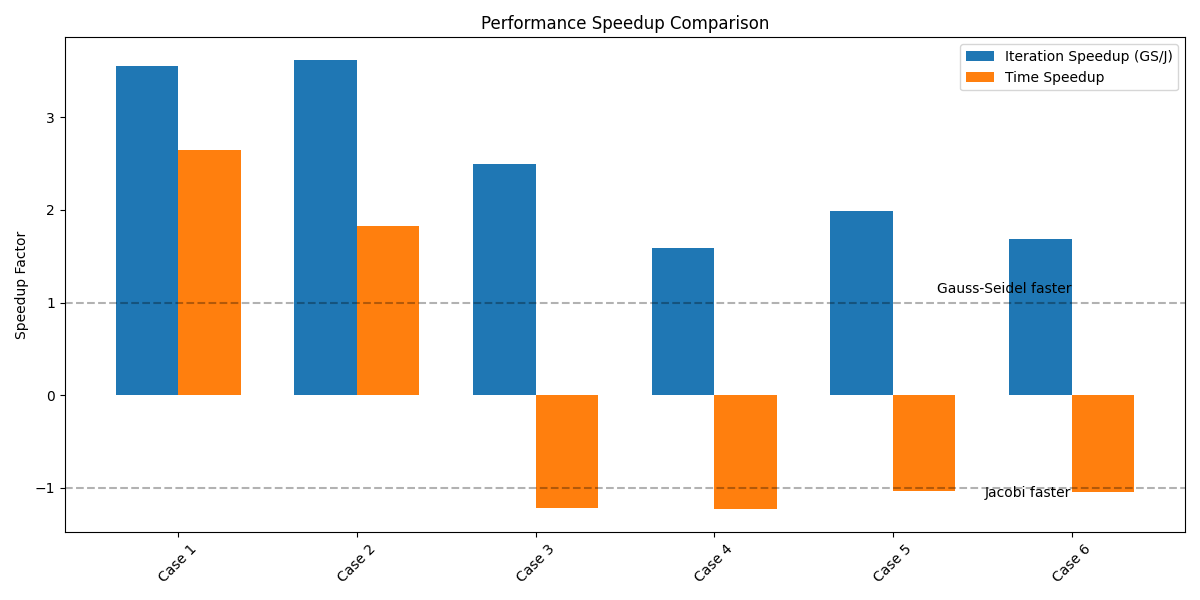
*Figure: 14. Poisson Convergence*



*Figure: 15. Singular Convergence*



*Figure: 16. Speedup Accuracy Ratio*



*Figure: 17. Speedup Comparison*

# 5. Discussion

## 5.1 Factors Affecting Performance

Several factors influence the relative performance of these methods:

**Table 8: Factors Affecting Performance**

|  |  |
| --- | --- |
| **Factor** | **Impact on Performance** |
| System Size | As the system size increases, Gauss-Seidel's advantages in both CPU time and accuracy become more pronounced. |
| Matrix Structure | Special structures like tridiagonal matrices can favor Jacobi in terms of both CPU time and accuracy. |
| Condition Number | For ill-conditioned systems, Gauss-Seidel demonstrates better accuracy while maintaining a CPU time advantage. |
| Convergence Rate | Gauss-Seidel's faster convergence rate (requiring fewer iterations) is the primary driver of its CPU time advantage. |
| Per-Iteration Cost | Gauss-Seidel has a higher per-iteration cost due to its sequential nature, but this is usually offset by the reduced iteration count. |
| Parallelizability | Jacobi is naturally parallelizable, which can be advantageous in parallel computing environments. |

## 5.2 Theoretical Explanation

The observed performance differences can be explained by theoretical properties of the methods:

For symmetric positive definite matrices, the spectral radius of the Gauss-Seidel iteration matrix is related to the spectral radius of the Jacobi iteration matrix by:

*ρ(B\_GS) ≤ [ρ(B\_J)]²*

This relationship explains why Gauss-Seidel typically converges in fewer iterations. The asymptotic error reduction after k iterations is approximately:

*error\_k ≈ [ρ(B)]^k × error\_0*

Since ρ(B\_GS) is typically much smaller than ρ(B\_J), Gauss-Seidel achieves faster error reduction. However, the per-iteration cost of Gauss-Seidel is higher due to its sequential nature, which explains why it can be slower for certain matrix structures despite requiring fewer iterations.

# 6. Recommendations

Based on our comprehensive analysis of CPU time and accuracy, we recommend:

**Table 9: Recommendations**

|  |  |
| --- | --- |
| **Application Context** | **Recommendation** |
| General-Purpose Applications | Use Gauss-Seidel as the default choice, especially for medium to large systems, as it offers significant advantages in both CPU time and accuracy. |
| Tridiagonal or Banded Systems | Consider Jacobi, as it may offer both CPU time and accuracy advantages for these specific structures. |
| Large Systems | Strongly prefer Gauss-Seidel, which can be up to 66.81× faster and up to 2.97× more accurate. |
| Ill-Conditioned Systems | Use Gauss-Seidel, which demonstrates better numerical stability and accuracy while maintaining a CPU time advantage. |
| Parallel Computing Environments | Consider Jacobi despite slower sequential performance, as its natural parallelizability may lead to better overall performance. |

# 7. Conclusion

Our comprehensive analysis of CPU time and accuracy reveals that Gauss-Seidel is generally the superior method for most applications, offering both faster computation and higher accuracy in most test cases. The performance advantage becomes more pronounced as system size increases.

However, for specific matrix structures like tridiagonal systems, Jacobi may be the better choice. The specific characteristics of the linear system, particularly its structure and size, can significantly influence the relative performance of these methods.

For practical applications, the choice between Jacobi and Gauss-Seidel should be guided by:

* The size and structure of the system
* The relative importance of CPU time vs. accuracy
* The computing environment (sequential vs. parallel)

This analysis provides a solid foundation for making informed decisions when selecting an iterative method for solving systems of linear equations in various application domains.