May 20, 2021

PARALLEL IMPLEMENTATION OF STEEPEST DESCENT AND CONJUGATE GRADIENT METHODS

Kishore Ram Sathia
ME18B085
Indian Institute of Technology Madras

Ameya Rajesh Ainchwar
ME18B122
Indian Institute of Technology Madras

Guides:
Prof. Kameswararao Anupindi
Prof. Rupesh Nasre

ABSTRACT

Solving a system of linear equations is an essential task in most scientific computations. For several engineering applications such as solving elliptical PDEs numerically, a large sparse system of equations has to be solved simultaneously. Direct methods such as Gaussian elimination or LU decomposition pose several disadvantages to solving such sparse systems compared to iterative methods such as Gauss-Jacobi or Gauss-Seidel. However, even these traditional iterative schemes do not come close to the speed required to solve such large systems of equations. Advances in the mathematical field of optimization have led to the development of several more complicated iterative algorithms, which are much more efficient in solving such systems. These advances have not only led to solving large sparse systems, but to solving large dense systems iteratively too.

This study explores two such algorithms, the Steepest Descent Method and the Conjugate Gradient Method. These are used to solve linear systems where the coefficient matrix is positive definite (i.e. symmetric and diagonally dominant). Serial programs for these two methods in the C language are implemented first, followed by the implementation of the same two algorithms in parallel. Three different parallel programming paradigms are used - OpenMP, MPI and OpenACC.

Finally, the performance of these algorithms in the serial and parallel implementations is compared. The Conjugate Gradient Method, considered a slight modification of the Steepest Descent Method, theoretically converges faster. This statement is verified. The parallel paradigm that performs the best is also found.

INTRODUCTION

The traditional methods to solve a system of linear equations pose some issues related to the size of the system and the speed of solving the system. The iterative methods like Gauss-Jacobi and Gauss-Seidel methods, although outperform the direct methods of Gaussian Elimination and LU Decomposition, are not desirable for very large systems. All these methods are slow and require many iterations to converge to the solution for large systems of equations. Hence, there is clearly a need for more sophisticated and efficient algorithms.

The Steepest Descent Method and Conjugate Gradient Method are designed to overcome exactly these kinds of problems. These methods are suitable for a symmetric and positive-definite matrix. Although the Steepest Descent method does not provide a big improvement over the traditional methods, it is considered a stepping stone for more sophisticated methods like Conjugate Gradient. The Conjugate Gradient method quickly converges to the solution even for a large system of equations and requires fewer number of iterations.

The efficiency and speed of these algorithms can be further improved by using parallel computing constructs. The algorithms can be parallelized by using shared memory systems like OpenMP, distributed memory systems like MPI or hybrid architectures using both CPU and GPU such as OpenACC. Each of these architectures come with their own set of advantages and disadvantages and specific applications.

OpenMP is relatively easy to implement and can be used for most functions, including recursive functions. However it suffers from memory limitations in memory intensive problems. MPI is suitable for problems which involve large memory and can be easily scaled. However, the programmer is responsible for data transfer among threads and synchronization among tasks. OpenACC is used to accelerate the programs at a high level and

Copyright © by ASME

in a platform independent way. It is relatively easy to implement and portable, but not very performance intensive. Hence each paradigm has its own merits. This report aims to compare the performance of these three parallel programming constructs by solving a system of linear equations using the Steepest Descent Method and the Conjugate Gradient Method.

Positive Definite and Diagonally Dominant Matrices

Consider a symmetric (square) matrix M with real entries. M is said to be a positive definite matrix if $x^T Mx > 0$ for all non-zero vectors x.

It can be shown that if M has all its eigenvalues > 0, then M is positive definite. Hence, positive definite matrices are always invertible.

A strictly diagonally dominant matrix is a square matix in which for every row of the matrix, the magnitude of the diagonal entry in a row is larger than the sum of the magnitudes of all the other (non-diagonal) entries in that row.

It can be shown that a strictly diagonally dominant matrix is positive definite. [1]
This is used to generate positive definite matrices in the code to test the two algorithms:

rand() is used to generate integer values (upto 20) in the upper triangular half of an $N \times N$ matrix A.
The matrix is then made symmetric by equating $A_{ij} = A_{ji}$.
Finally, $20N$ is added to the diagonal elements of A to make it diagonally dominant.

Steepest Descent Method

The Steepest Descent method [2] is an iterative algorithm to solve a system of linear equations of the form:

$$Ax = b$$

where A is an $n \times n$ matrix and b is a vector $\in R^n$.
For this algorithm, A has to be a positive definite matrix.

Define a quadratic function $F(x) = \frac{1}{2}x^T Ax - x^T b$

Simplifying the LHS of the following:
$\frac{1}{2}(x - A^{-1}b) \cdot A(x - A^{-1}b) = \frac{1}{2}x \cdot Ax - b \cdot x + \frac{1}{2}b \cdot A^{-1}b$

we notice that we can write $J(x)$ as:
$J(x) = \frac{1}{2}(x - A^{-1}b) \cdot A(x - A^{-1}b) - \frac{1}{2}b \cdot A^{-1}b$

Since we are dealing with positive definite matrices, the solution to the system of equations is unique, say $A^{-1}b = y_0$.
We then have $J(y_0) = -\frac{1}{2}b \cdot A^{-1}b$

We also know that $(x - A^{-1}b) \cdot A(x - A^{-1}b)$ is always positive (by definition of SPD matrix). Hence y_0 would be attained when $J(x)$ is minimized with respect to x. Our goal now is to minimize $J(x)$.

Start at some point x_0, find the direction of the steepest descent of the value of $J(x)$ and move in that direction as long as the value of J(x) descends.
At this point (when $J(x)$ stops descending), find the new direction of the steepest descent and repeat the whole process.

Let the initial guess vector be x_0.
Our initial direction is given by $x_0 + td_0$
where $d_0 = -\nabla F(x_0) = -(Ax_0 - b)$

We can simplify $F(x_0 + td_0)$ as :
$F(x_0 + td_0) = F(x_0) + td_0 \cdot (Ax_0 - b) + \frac{t^2}{2}d_0 \cdot Ad_0$

We wish to stop when F is no longer descending, i.e. when F reaches minimum wrt t. Differentiating wrt t, we get that:
$t_0 = |d_0|^2/(d_0 \cdot Ad_0)$

Hence the first stopping point x_1 is given by:
$x_1 = x_0 + |d_0|^2 d_0/(d_0 \cdot Ad_0)$

In general, we have:

$$x_{n+1} = x_n + |d_n|^2 d_n/(d_n \cdot Ad_n)$$

The solution is said to have converged when the relative error:
$||x_{n+1} - x_n||/||x_n|| <$ tolerance
We choose the 2-norm and a tolerance of 10^{-7}.

Summary:

1. Input: A, b, guess vector x_0
2. Perform $x_{n+1} = x_n + |d_n|^2 d_n/(d_n \cdot Ad_n)$
   where $d_n = -(Ax_n - b)$
   Repeat until convergence criterion has been reached.

Comments:

1. As the condition number of the matrix A becomes higher, more iterations are necessary for convergence.
   That is, for a fixed number of iterations, the accuracy of the solution decreases as the condition number of the matrix increases.
2. The two main computational advantages of the steepest descent algorithm is the ease with which a computer algorithm can be implemented and the low storage requirements necessary, $\mathcal{O}(n)$

.

Conjugate Gradient Method

The steepest descent method makes fast progress initially when the guess is away from the optimum.
However, this method tends to slow down as iterations progress.
It can be shown mathematically that there exist better descent directions than the negative of the gradient direction.
One such approach where the convergence to the solution is faster, is conjugate gradient method [3].

Copyright © by ASME

Motivation: In the Steepest Gradient Method, when computing the direction in which to proceed, it was only information about the gradient at the current iterate x_n that was being used, not any of the previously computed gradient values.
The Conjugate Gradient uses a combination of the gradient at the current iterate x_n and all the previous computed gradient values as well to compute the new direction.

The Conjugate Gradient method is also iterative algorithm to solve a system of linear equations of the form:

$$Ax = b$$

where A is an $n \times n$ matrix and b is a vector $\in R^n$.
For this algorithm, A has to be a positive definite matrix.

Define a quadratic function $F(x) = \frac{1}{2}x^T Ax - x^T b$

Let the initial guess vector be x_0.
Our initial direction, similar to Steepest Descent is given by d_0 where $d_0 = -\nabla F(x_0) = -(Ax_0 - b)$

However, for the k^{th} step, instead of using the direction as d_k, we enforce that the directions p_k be "conjugate" to each other.
The conjugation constraint used is an orthonormal-type constraint and similar to Gram-Schmidt orthonormalization. Applying this, we have:
This gives the following expression:

$$p_k = d_k - \sum_{i<k} \frac{p_i^{\mathsf{T}} A d_k}{p_i^{\mathsf{T}} A p_i} p_i$$

Repeating the mathematical steps elaborated in the Steepest Descent Method, we get:

$$x_{k+1} = x_k + \alpha_k p_k$$

with $\alpha_k = \dfrac{p_k^{\mathsf{T}} d_k}{p_k^{\mathsf{T}} A p_k}$

The solution is said to have converged when the relative error: $||x_{n+1} - x_n||/||x_n|| <$ tolerance
We choose the 2-norm and a tolerance of 10^{-7}.

Additionally, it can be shown mathematically, that replacing p_k^T with d_k^T in the numerator of α_k does not affect the method, so for ease of computation, we make this substitution.

Summary:

1. Input: A, b, guess vector x_0
2. Perform $x_{n+1} = x_n + \alpha p_n$
   where $\alpha_n = \dfrac{d_n^{\mathsf{T}} d_n}{p_n^{\mathsf{T}} A p_n}$
   Repeat until convergence criterion has been reached.

Comments:

1. As the condition number of the matrix A becomes higher, more iterations are necessary for convergence.
   That is, for a fixed number of iterations, the accuracy of the solution decreases as the condition number of the matrix increases.
2. Since the Conjugate Method primarily differs from the Steepest Descent only with respect to the computation of the direction, the major computational advantages are the same in both cases i.e. the ease with which a computer algorithm can be implemented and the low storage requirements necessary, $\mathcal{O}(n)$.
   The significant advantage of the Conjugate Gradient compared to the Steepest Descent is the number of iterations required for convergence to the solution.

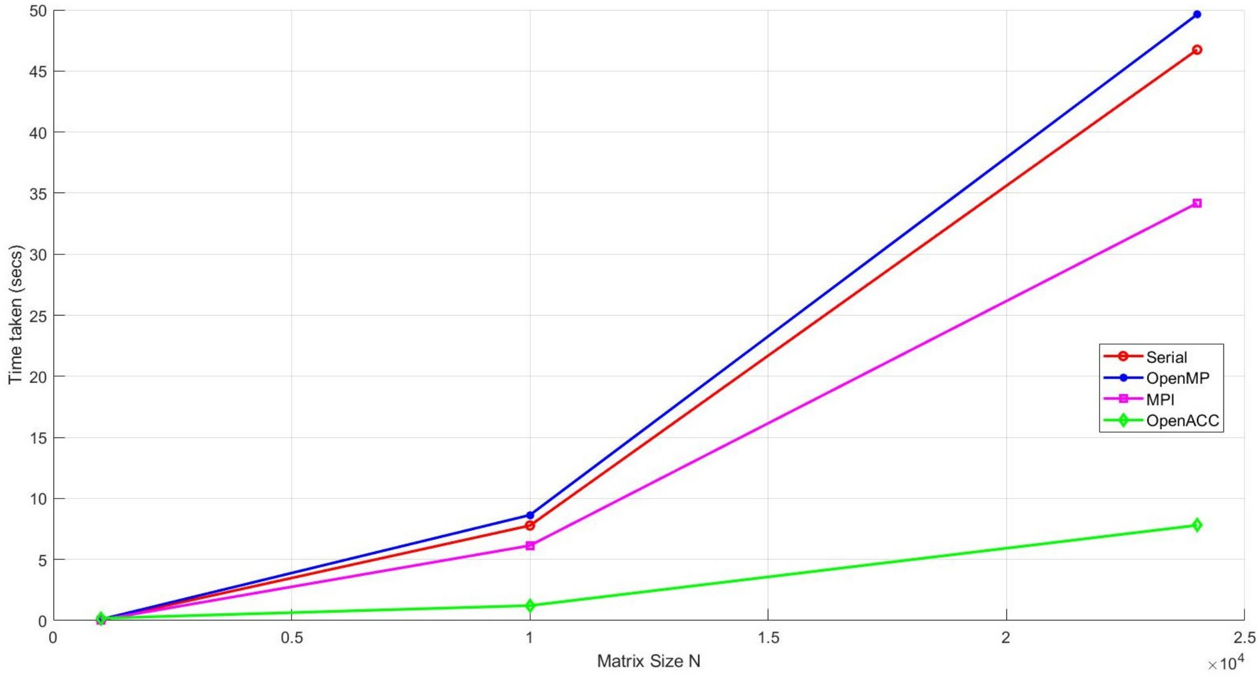
.

Results and Implementation

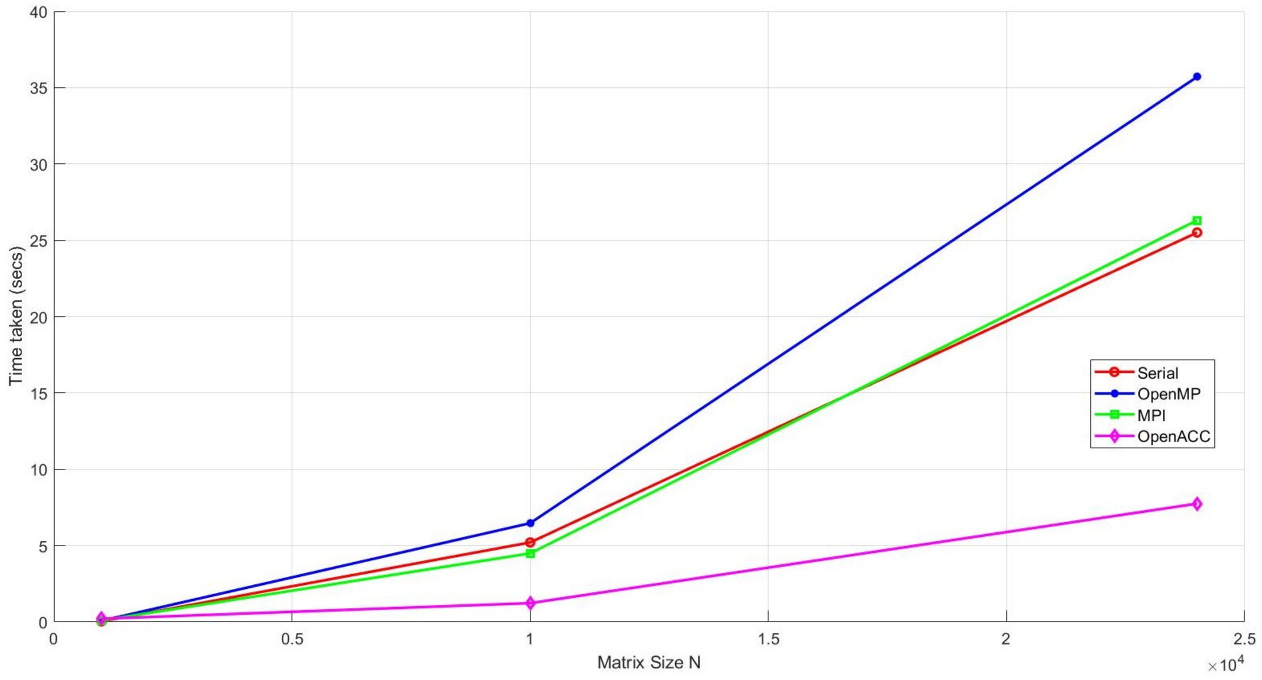
Conclusions:

1. As N is increased, the number of iterations required for convergence does not increase.
   For Steepest Descent, for all tested N, the number of iterations is 8-11.
   For Conjugate Gradient, for all tested N, the number of iterations is 5-6.
2. OpenMP takes the most run-time, even more than that taken by serial code. However it is expected that as the size N increases to much larger values, OpenMP becomes faster than serial code.
3. MPI is considerably faster than the serial code. The rate of decrease of speed is also higher, and it is expected to have more than 2x speedup for larger N such as $N = 100,000$.
4. OpenACC clearly has the fastest run-time for all N, being considerably faster than the serial programs and the other 2 parallel programs for both Steepest Descent and Conjugate Gradient.
   This is expected since most of the calculations in these algorithms involve matrix multiplications, which can be run in a highly optimized parallelized manner on a GPU.
5. For both OpenMP and MPI, as the number of threads/processes is doubled, the run time is roughly halved, until a number of threads q, beyond which the run-time stagnates to a constant value.
   This number q increases as N is increased.

Copyright © by ASME

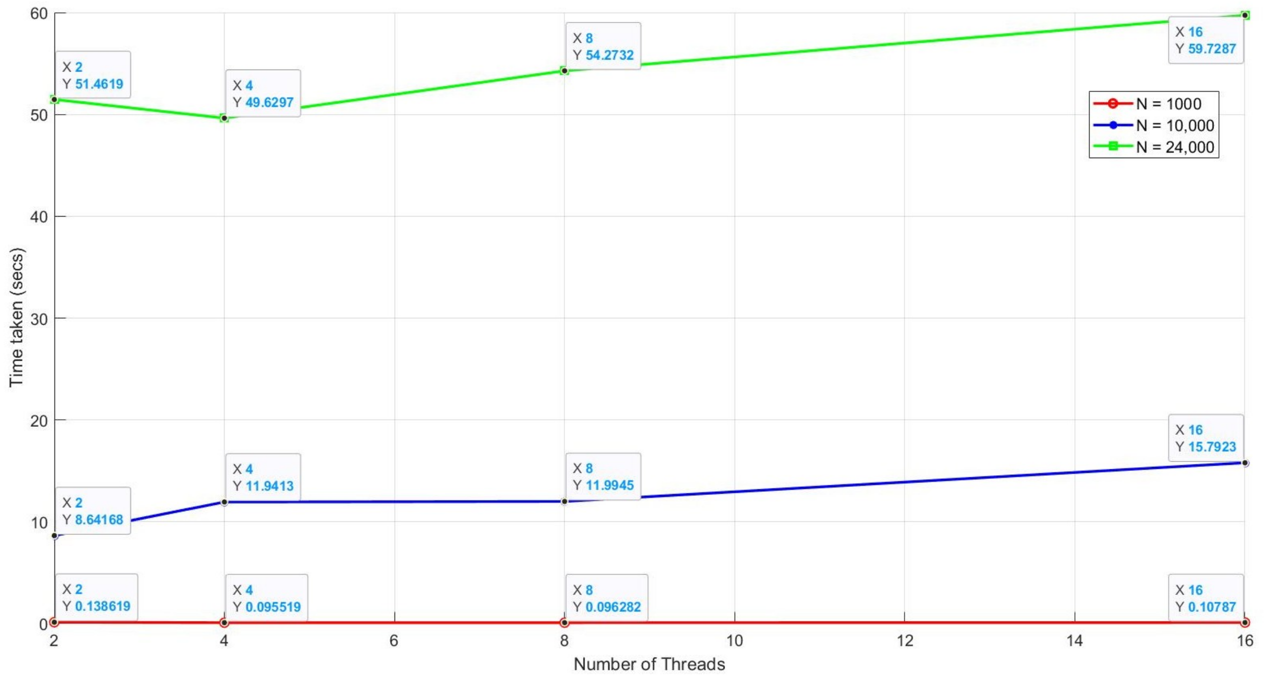


RUN-TIME FOR VARIOUS N FOR STEEPEST DESCENT METHOD

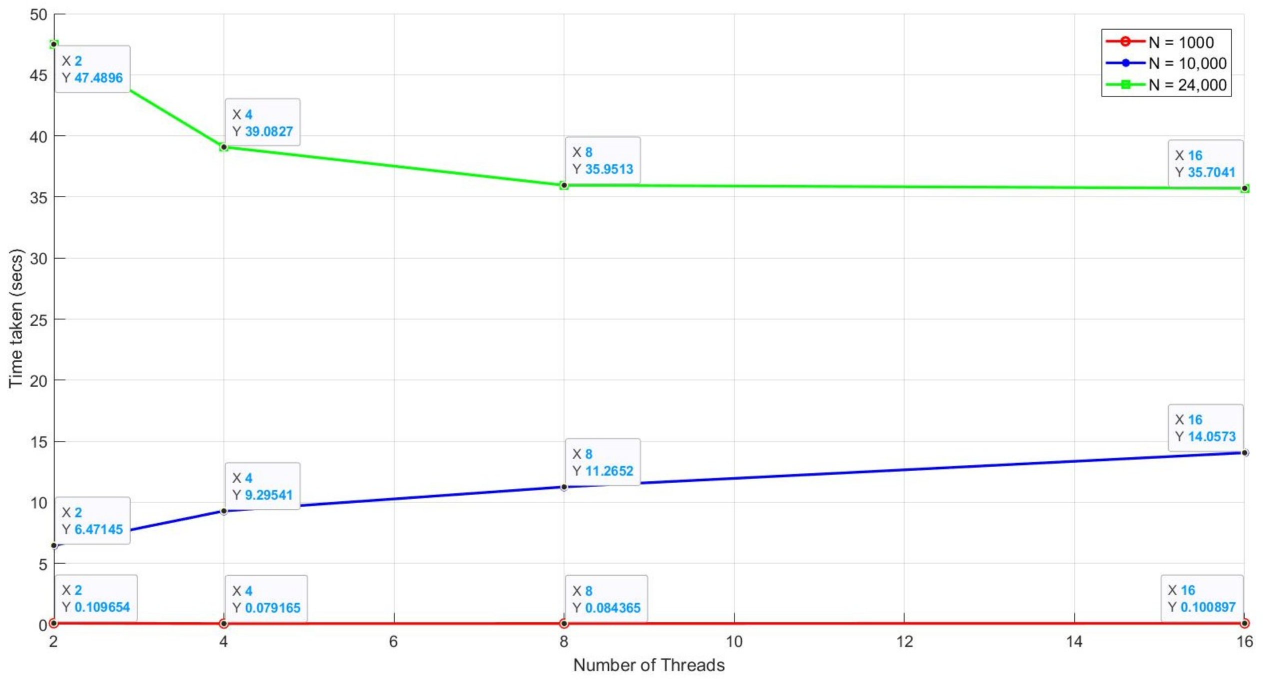


RUN-TIME FOR VARIOUS N FOR CONJUGATE GRADIENT METHOD

Copyright © by ASME

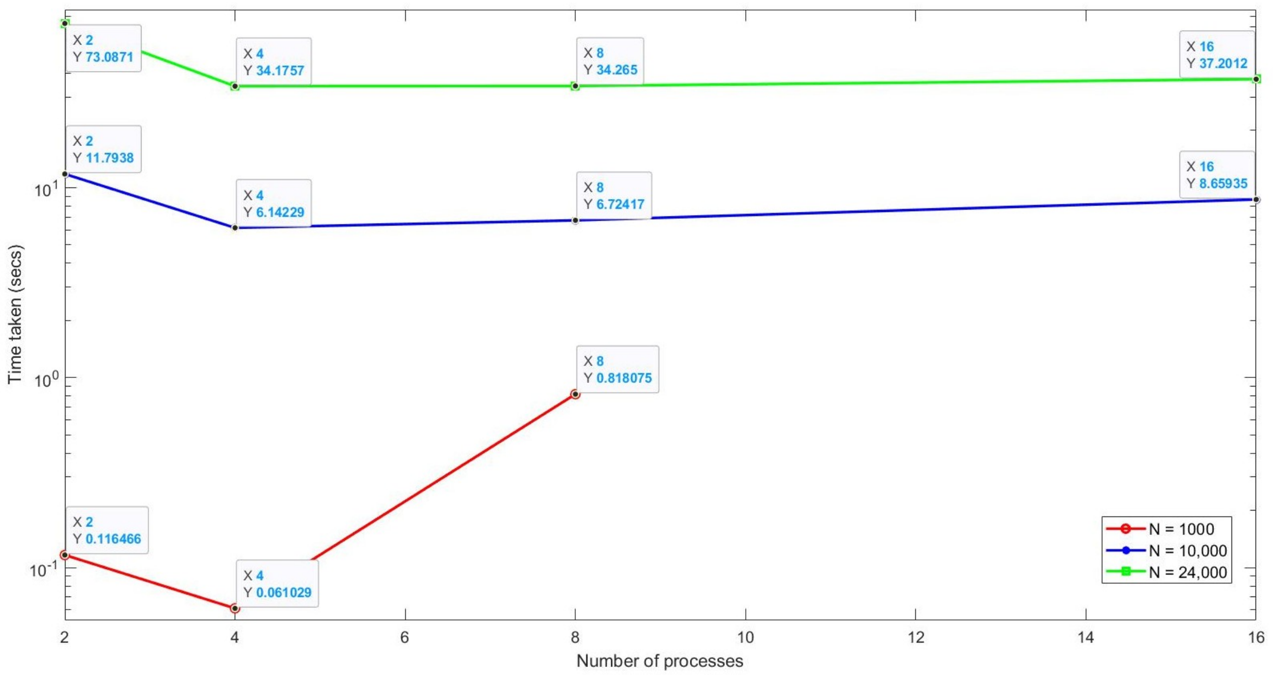


RUN-TIME VS NUMBER OF THREADS FOR VARIOUS N
(FOR STEEPEST DESCENT METHOD PARALLELIZED USING OpenMP)

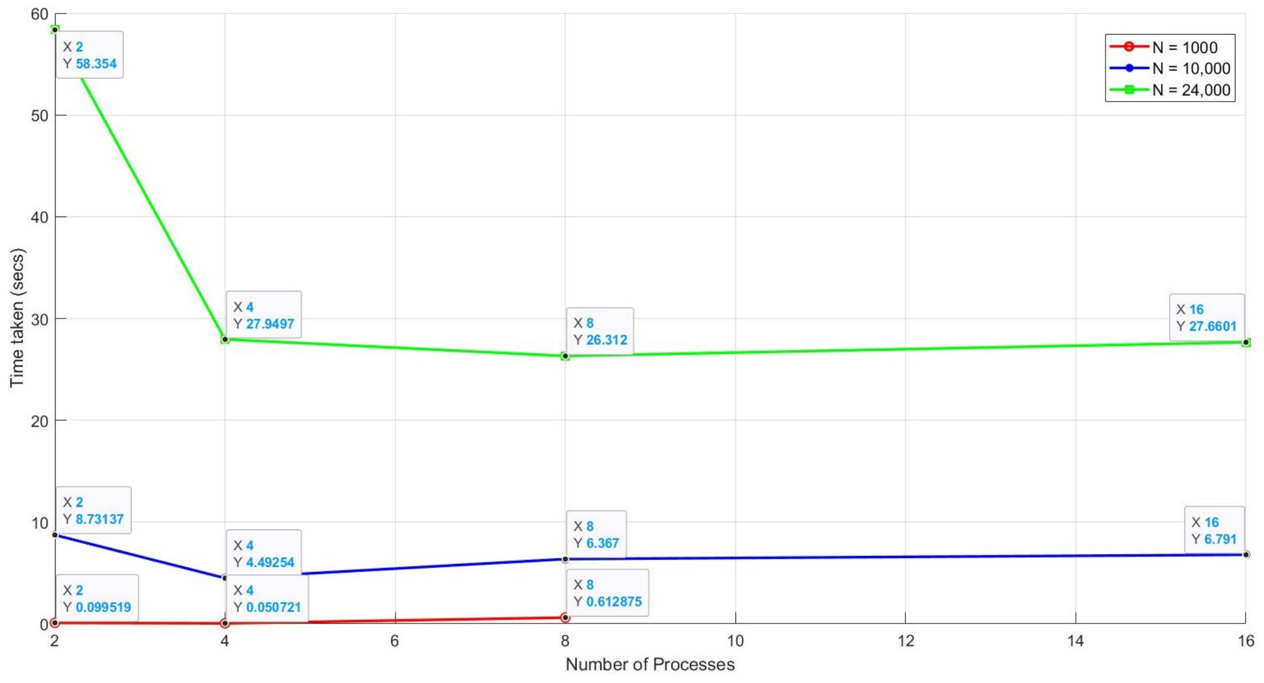


RUN-TIME VS NUMBER OF THREADS FOR VARIOUS N
(FOR CONJUGATE GRADIENT METHOD PARALLELIZED USING OpenMP)

Copyright © by ASME



RUN-TIME VS NUMBER OF THREADS FOR VARIOUS N
(FOR STEEPEST DESCENT METHOD PARALLELIZED USING MPI)



RUN-TIME VS NUMBER OF THREADS FOR VARIOUS N
(FOR CONJUGATE GRADIENT METHOD PARALLELIZED USING MPI)

Copyright © by ASME

Comments:

1. A simple implementation of the serial program on MPI would simply split A column-wise evenly by number of processes p. However, if N is not divisible by p, then the program cannot run.
   This highlights a very distinct difference between MPI and OpenMP/OpenACC - parallelising the serial code for MPI usually takes longer than with OpenMP or OpenACC. Accommodating N which are not divisible by p leads to furthur complicating the code.
   Hence data could not be displayed for $N = 1000, p = 16$ for MPI parallelised code.
2. For large N such as $N = 50,000$ or $N = 100,000$, using static arrays generates segmentation faults when running on the IIT Madras Aqua cluster.
   This problem is abated by using dynamic arrays (eg. using malloc in C) instead of using static arrays. This was done for the serial code, and the run times were obtained for $N = 100,000$ as :
   Steepest Descent: 871.28 secs (9 iterations)
   Conjugate Gradient: 446.43 secs (5 iterations)
   However, due to time constraints, the parallel code has not been modified to use dynamic arrays. Hence the maximum N used for comparison is $N = 24,000$

CONCLUSION

Solving a large system of linear equations is an essential task in many scientific computations. Parallelization has significantly reduced the time required for the operation. This report has successfully obtained the time required by various parallel programming paradigms to execute this task. It can be clearly seen that as the problem size increases, serial implementations require more time than some of the parallel implementations. The performance of the parallel implementations have also been compared with each other. OpenACC has given the best performance for both the algorithms. The optimum number of threads/processes for each problem size in the OpenMP and MPI codes have also been identified. Hence, the various parallel methods were compared and their performance was analyzed for the steepest descent and conjugate gradient methods.

REFERENCES

[1] Diagonally Dominant Matrix, Wolfram Math World `https://mathworld.wolfram.com/DiagonallyDominantMatrix.html#:~:text=A%20strictly%20diagonally%20dominant%20matrix,diagonal%20entries%20is%20positive%20semidefinite`

[2] NPTEL Online Course, Advanced Numerical Analysis, Lecture Notes `https://nptel.ac.in/content/storage2/courses/103101111/downloads/Lecture-notes/Module_4_Solving_Ax=b.pdf`

[3] Stephen Boyd: Convex Optimization, `https://stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`

Additional References

1. `https://advancesindifferenceequations.springeropen.com/articles/10.1186/s13662-020-02715-9#:~:text=The%20steepest%20descent%20of%20gradient%2Dbased%20iterative%20algorithm%20for%20solving,f(x(i)).`
2. `https://antonleykin.math.gatech.edu/math2605spr10/PROJECTS/SteepestDescent.pdf`
3. NPTEL Online Course, Numerical Optimization `https://nptel.ac.in/courses/106/108/106108056/`
4. `https://people.sc.fsu.edu/~jburkardt/c_src/cg/cg.html`