sciendo

DOI: 10.2478/9788367405805-033

ICAMS
The International Conference on Advanced Materials and Systems

IMPLEMENTING JACOBI ALGORITHM *VERSUS* GAUSS-SEIDEL ALGORITHM IN SOLVING A DISCRETIZED PROBLEM

SEBASTIAN SBÎRNĂ[1], LIANA-SIMONA SBÎRNĂ[2]

[1]*DTU Compute Department, Institute of Applied Mathematics and Computer Science, Technical University of Denmark, Anker Engelunds Vej 1, Building 101A, Kongens Lyngby, Denmark, seby.sbirna@gmail.com*

[2]*Department of Chemistry, Faculty of Sciences, University of Craiova, 107i Bucharest Way, Craiova, Romania, simona.sbirna@gmail.com*

As it is well known that performance of algorithms generally relies both on the hardware that they are run on and on the way that they are implemented and executed the present paper aims to show that the executions' wall-clock time might be affected by parallelizing the code, although this could sometimes become a hard task. Within this work, based on Poisson partial differential equation, we shall present different aspects of a comparison between implementing Jacobi algorithm in solving a discretized problem and implementing Gauss-Seidel algorithm in solving the same. On one hand, the results obtained by implementing the Jacobi algorithm benefited very much from the parallelization, so the first case has shown both good performance and scalability when introducing supplementary threads; as its structure always needed to exhibit a complete data copy computed in its former iterations, it became easier for us to have the code parallelly executed, because there was an insignificant risk of data races' hitting. On the other hand, the results obtained by implementing the Gauss-Seidel algorithm managed to solve the discretized problem, although its performance was not as good as the one of the Jacobi algorithm, so the second case has shown a good execution performance in varying sizes of sequentially executed data, taking into account that, as far as parallelizing was concerned, the performance was unreliable and random.

Keywords: Jacobi algorithm, Gauss-Seidel algorithm, Poisson partial differential equation.

INTRODUCTION

As the Poisson equation lately becomes an interesting topic in describing the heat distribution within a uniform environment, we shall analyze through this paper the model of a cubic room, for which we shall complementarily implement the Jacobi algorithm and the Gauss-Seidel algorithm, in a sequential way, in order to solve a certain discretized problem.

By the end of the paper, we shall be able to express which one of these two algorithms is most appropriate for the chosen model, *i.e.*, which one of them is most suitable for this purpose.

PRELIMINARY STATEMENTS

For calculus' simplicity, the cubic room will be described in the 3D cartesian coordinate system as presented in Figure 1, *i.e.*, centered in the center of it – denoted by $\mathcal{O}$ (0, 0, 0) – and having [-1, 1] as definition domain for all the coordinate axes. The vertices of the cube will be:

$\mathcal{A}$ (1, 1, 1);
$\mathcal{B}$ (–1, 1, 1);
$\mathcal{C}$ (–1, –1, 1);
$\mathcal{D}$ (1, –1, 1);
$\mathcal{E}$ (1, 1, –1);
$\mathcal{F}$ (–1, 1, –1);
$\mathcal{G}$ (–1, –1, –1);
$\mathcal{H}$ (1, –1, –1).

https://doi.org/10.2478/9788367405805-033

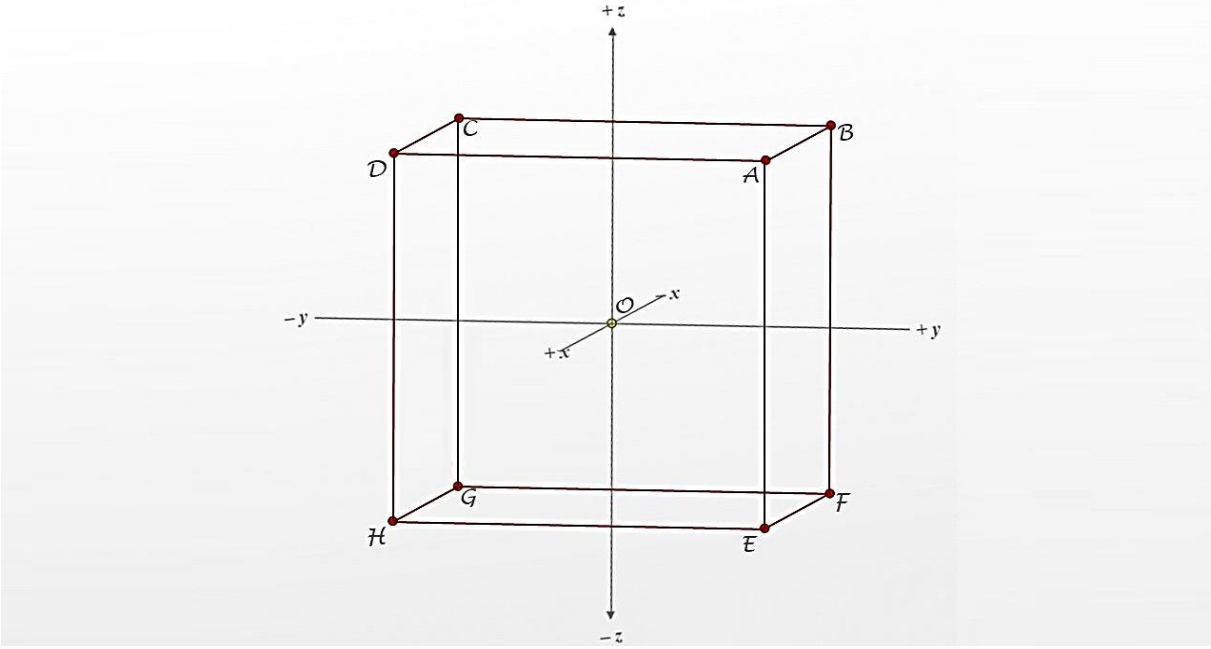

Figure 1. Presenting the position of the cubic room within the 3D cartesian coordinate system

Among the six walls, only one will be set to be cold – having a temperature of 0°C, whereas the rest of them will be set to have room temperature – more exactly, 20 °C.

A radiator which is capable to emit a 200 °C/m^2 radiation shall be placed near to the cold wall, namely at the location specified by the following function:

$$f = \begin{cases} 200, x_1 \le x \le x_2;\ y_1 \le y \le y_2;\ z_1 \le z \le z_2 \\ 0, \text{otherwise} \end{cases} \quad (1)$$

in which: $x_1 = -1, x_2 = -3/8; y_1 = -1, y_2 = -1/2; z_1 = -2/3, z_2 = 0$, as shown below, in Figure 2.

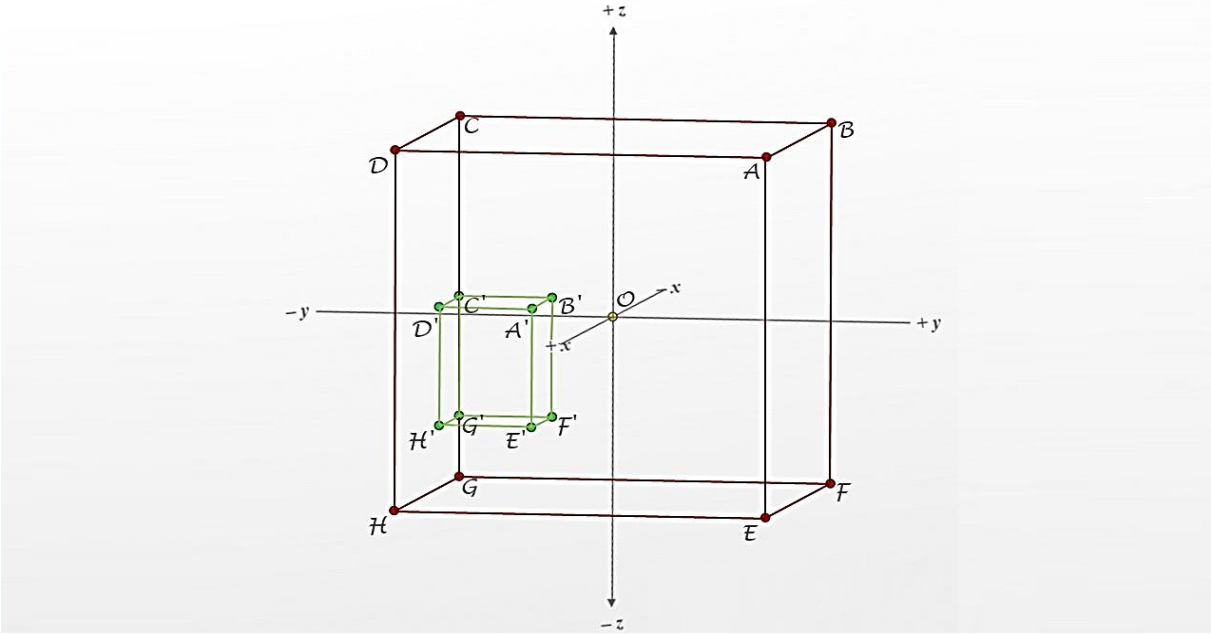

Figure 2. Presenting the position of the radiator inside the cubic room within the 3D cartesian coordinate system

The vertices of the smaller cube (representing the radiator placed inside the cubic room) will be, therefore:

$\mathcal{A}'$ (-3/8, -1/2, 0);

$\mathcal{B}'$ (–1, -1/2, 0);

$\mathcal{C}'$ (–1, –1, 0);

$\mathcal{D}'$ (-3/8, –1, 0);

$\mathcal{E}'$ (-3/8, -1/2, -2/3);

$\mathcal{F}'$ (–1, -1/2, -2/3);

$\mathcal{G}'$ (–1, –1, -2/3);

$\mathcal{H}'$ (-3/8, –1, -2/3).

The dissipation of the heat shall be described by selecting grid points inside the cubic room and estimating the heat values, by using appropriate differential equations.

The cube will fit into three different L-type caches, which will be denoted as L, L' and L".

SPECIFICATIONS OF THE COMPUTING CLUSTER MACHINE USED FOR STUDY

As to execute the tests for this study, a computing cluster machine shall be used, whose hardware and software specifications are gathered in Table 1:

Table 1. Hardware and software specifications of the computing cluster machine used for study

Hardware specifications	Sockets	2
	CPUs/socket	12
	No. of CPUs	24
	CPUs Model	Intel Xeon Processor E5-2650
	Architecture	64-bit
	Cache L	32 kB
	Cache L'	256 kB
	Cache L"	61440 kB
Software specifications	Cubic matrix implementation	C-native way, indexed with triple pointers
	Optimization flags	-O3 and -unroll-loops
	C compiler	GCC v9.2.0

For both algorithms proposed, it would be interesting to express the maximum size of the variable N, denoting the number of grid points chosen inside the cubic room (on each axis),

In the first case (Jacobi algorithm), we will find, for each L-type cache:

$$\mathrm{N} = \sqrt[3]{\mathrm{L}/24} - 2 \tag{2}$$

whereas in the second case (Gauss-Seidel algorithm), we will similarly find, in each case:

$$\mathrm{N} = \sqrt[3]{\mathrm{L}/16} - 2 \tag{3}$$

Using these formulas, we shall calculate and report below – in Table 2 – the maximum size of N for each L-type cache.

Table 2. Maximum size of N for each L-type cache for both Jacobi and Gauss-Seidel algorithms

	Cache	Jacobi algorithm	Gauss-Seidel algorithm
Maximum size of N	L	9	10
	L'	20	23
	L"	134	154

The limits imposed for the dimensions of the grid placed inside the cubic room shown in Figure 1 are established in order to fit this entire room into each of the three caches presented before.

POISSON PARTIAL DIFFERENTIAL EQUATION

Having the limits calculated and presented before, it will become possible to establish the values of N that shall be used for our tests. Namely, the values chosen for N will be equal-to, just-above and just-below these limits, noting that all the rest of the values that N will take through the test will be in-between the limits, as to permit us to determine the behavior of both algorithms, as long as the size of the data will be kept within the above specified L-type cache sizes – as stated by Borawake and Hiwarekar (2024), as well as by Goona *et al*. (2021).

In computational fluid dynamics, Poisson partial differential equation is appropriate to describe the distribution of temperature within a particular space, such as our cubic room.

So, the main problem will be to estimate steady-state distribution of temperatures within the cubic room presented in Figure 1, taking into account the influence of the radiator placed inside it, as shown in Figure 2.

As it is well known, the equation governing this kind of diffusion process is the Poisson partial differential equation, whose general form is:

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} = -f(x, y, z),\ x_1 \le x \le x_2;\ y_1 \le y \le y_2;\ z_1 \le z \le z_2 \quad (4)$$

in which $\Phi = \Phi(x, y, z)$ represents the temperature that diffuses in space and $f(x, y, z)$ is the function already specified in Equation 1 (having there given also the limits of the intervals for x, y, z) – as shown by Amjad and Abdullah (2021), Mohanty and Niranjan (2024), Yu *et al*. (2024).

BEHAVIOR OF JACOBI ALGORITHM

Presenting Jacobi Algorithm

The Jacobi algorithm uses the grid structure of the cub so as the values of f and Φ will be stored in 3D arrays, and the values corresponding to the inner grid of Φ will be initialized to arbitrary values, as presented by Khrapov and Volkov (2024).

Afterwards, a duplicate of Φ shall be introduced, denoted as Φ_old, which will be initialized as identical with Φ. Having both versions of Φ, the Jacobi algorithm will alternately compute the values within the grid until reaching the convergence. The number of cubic data arrays which represent (in the present case) $\Phi = \Phi(x, y, z)$ is two. Consequently, the Jacobi algorithm will iterate over one set of computations, alternating between the two versions of Φ, allowing the algorithm to permanently keep track of latest data, which will be furthermore computed upon, but also stored in a second array, so latest data can never be overwritten until completely used.

For the Jacobi algorithm, there will be triple nested loop iterations, *i.e*., loop iterations through all dimensions corresponding to the cubic arrays (excluding the cube boundary). During each iteration, the new values of Φ will be computed using the equation bellow:

$$\Phi_{i,j,k}^{n+1} = \frac{1}{6}\Big(\Phi\_old_{i-1,j,k}^{(n)} + \Phi\_old_{i+1,j,k}^{(n)} + \Phi\_old_{i,j-1,k}^{(n)} + \Phi\_old_{i,j+1,k}^{(n)} + \Phi\_old_{i,j,k-1}^{(n)} + \Phi\_old_{i,j,k+1}^{(n)} + \Delta^2 f_{i,j,k}\Big) \quad (5)$$

where the Δ symbol appearing in the equation above stands for the distance between the grid points placed in the same dimension.

Indexing will be made in the order: $i \to j \to k$, *i.e.*, the outer-most loop will be run through depth, the middle one – through rows and the inner-most one – through columns.

The algorithm will be ended when the difference between the values obtained for two Φ grids during two consecutive iterations will become so small as to conclude convergence. In other words, the difference between Φ and Φ_old (when compared to a specified threshold) should become insignificant for the algorithm to end.

Implementing Jacobi Algorithm

Implementing Jacobi algorithm will be carried out using a single thread only (procedure usually called "sequential way of implementation"), following the theoretical algorithm.

The first three arguments will be the cubic arrays of f, Φ and Φ_old . The constant N will represent the total number of grid points inside the cube – in all dimension – for each of the cubic data arrays, whereas the final argument will be the tolerance against which the difference between iteration will be taken into comparison. We shall also take into account the maximum number of iterations performed before ending the algorithm (*i.e.*, until reaching the convergence).

BEHAVIOR OF GAUSS-SEIDEL ALGORITHM

Presenting Gauss-Seidel Algorithm

The Gauss-Seidel algorithm is able to approximate the solution of the Poisson differential equation by using the same 3D grid as Jacobi algorithm, as shown by Khrapov and Volkov (2024).

Generally, the difference between these algorithms consists in the number of cubic data arrays which represent (in the present case) $\Phi = \Phi(x, y, z)$ – temperature that diffuses in space.

As far as the Gauss-Seidel algorithm is concerned, the number of cubic data arrays which represent $\Phi = \Phi(x, y, z)$ is reduced to just one (comparatively to two, as already stated for the Jacobi algorithm).

The values for the grid corresponding to the observation of Φ are again initialized by making an arbitrary guess. Consequently, during each iteration, the new values of Φ will be computed using the equation bellow:

$$\Phi_{i,j,k}^{n+1} = \frac{1}{6}\left(\Phi_{i-1,j,k}^{(n)} + \Phi_{i+1,j,k}^{(n)} + \Phi_{i,j-1,k}^{(n)} + \Phi_{i,j+1,k}^{(n)} + \Phi_{i,j,k-1}^{(n)} + \Phi_{i,j,k+1}^{(n)} + \Delta^2 f_{i,j,k}\right) \qquad (6)$$

where the Δ symbol appearing in the equation above stands again for the distance between the grid points placed in the same dimension.

Equation (6) shows that this method of computing new values of Φ implies that, for a certain iteration, the computed values will depend on some other values, previously calculated during the same iteration.

Implementing Gauss-Seidel Algorithm

The core of the Gauss-Seidel algorithm is mostly equivalent to the one of the Jacobi algorithm, the main difference being the absence of the duplicate of Φ, which used to be called Φ_old, so the update calculations shall be directly performed upon the cubic data array Φ.

Compared to the other algorithm, this one does not show any reference to the previous values, at different grid points, so a temporary variable must be used in order to store any new

value. Afterwards, a computation of the squared difference between the two values shall be performed, so that eventually the old one can be replaced by the new one in the cubic data array.

After each iteration, the updated value of Φ must be compared to the grid matrix previously obtained, permanently checking if convergence is achieved, in order to stop the algorithm.

The implementation of Gauss-Seidel algorithm within our study will also be a sequential one, as in the former case.

JACOBI ALGORITHM *VERSUS* GAUSS-SEIDEL ALGORITHM

The performance of an algorithm is relied both on the hardware it runs on, as well as the way it is implemented and executed with respect to said hardware.

In order to ultimately compare the performances of the two algorithms submitted to our study, we shall express a formula giving the lattice updates per second, which can be denoted as $L(N)$.

$$L(N) = \frac{I \cdot N^3}{t} \tag{7}$$

where I stands for the number of iterations performed until achieving convergence, N is – as usually – the number of points inside the grid (considered in all dimensions), whereas t is the total wall-clock time of the algorithm.

We shall also calculate – in both cases – the parameter named floating-point operations per second (denoted as $flop/s$), which represents a measure of computer performance while developing each of the two algorithms. In order to find the expression of $flop/s$, the number of lattice updates per second – $L(N)$ – needs to be multiplied with F, that represents the floating-point operations number in the loop that runs through columns in each algorithm.

$$flop/s = L(N) \cdot F \tag{8}$$

Hence, combining the last two equations, we get the expression for $flop/s$, which will be considered as the main performance indicator for both algorithms:

$$flop/s = \frac{I \cdot N^3 \cdot F}{t} \tag{9}$$

What we can ascertain is that there is a significant performance difference between the two algorithms, in the sense that Jacobi algorithm performance varies proportional to the value of N, whereas Gauss-Seidel algorithm maintains constant performance, not being able to increase more than around one hundred updates of the lattice per second.

CONCLUSION

By making a comparison between Jacobi algorithm and Gauss-Seidel algorithm (using their sequential implementations), we can observe that, focusing on the lattice updates realized per second for different values of the number of points inside the grid, for the same size of the problem, Jacobi algorithm requires a greater number of iterations for the convergence to be achieved than Gauss-Seidel algorithm, but this observation should not automatically lead to the conclusion that the second algorithm performs stronger than the first.

Indeed, assuming that both algorithms submitted to comparison are to be executed for an entire set of values given to the number of points inside the cubic grid, maintaining this as the only changeable parameter (the maximum allowed number of iterations, as well as the tolerance

threshold remaining static during the test), the performance will be mainly expressed by the number of updates of the lattice per second.

In these terms, as previously stated, Jacobi algorithm prevails over Gauss-Seidel algorithm.

It is worth noticing that, in the textile and leather industry, efficient algorithmic performance is critical for processing large datasets involved in quality assessment, pattern recognition, and defect detection. Parallelizing algorithms like Jacobi and Gauss-Seidel in solving Poisson partial differential equations can optimize these processes by reducing computational time for simulations and data analysis. For instance, Jacobi's algorithm, with its favorable scalability in parallelization, can be applied in modeling heat distribution in fabric treatments or predicting wear in leather surfaces. Meanwhile, even though the Gauss-Seidel algorithm may not parallelize as efficiently, its sequential strength makes it valuable for smaller, detailed simulations, such as stitching pattern assessments or localized stress testing in textiles. By improving algorithm performance in these areas, this research supports faster and more reliable quality control in textile and leather production, ultimately enhancing product durability and efficiency of production.

REFERENCES

Amjad, A. & Abdullah, A.A. (2021). The Solution of Poisson Partial Differential Equations via Double Laplace Transform Method. *Partial Differential Equations in Applied Mathematics*, *4*, 100058, 1–4. https://doi.org/10.1016/j.padiff.2021.100058

Borawake, V. & Hiwarekar, A. (2024). Modified Double Laplace Transform of Partial Derivatives and Its Applications. *Gulf Journal of Mathematics*, *16*(2), 353-363. https://doi.org/10.56947/gjom.v16i2.1892

Goona, N.K., Parne, S.R. & Sashidhar, S. (2021). Distributed Source Scheme to Solve the Classical Form of Poisson Equation Using 3-D Finite-Difference Method for Improved Accuracy and Unrestricted Source Position. *Mathematics and Computers in Simulation*, *190*, 965-975. https://doi.org/10.1016/j.matcom.2021.06.025

Khrapov, P.V. & Volkov N.S. (2024). Comparative Analysis of Jacobi and Gauss-Seidel Iterative Methods. *International Journal of Open Information Technologies*, *12*(2), 23-34. https://doi.org/10.48550/arXiv.2307.09809

Mohanty, R.K. & Niranjan, V. (2024). A Class of New Implicit Compact Sixth-Order Approximations for Poisson Equations and the Estimates of Normal Derivatives in Multi-Dimensions. *Results in Applied Mathematics*, *22*, 100454, 1–21. https://doi.org/10.1016/j.rinam.2024.100454

Yu, Z., Wu, L., Zhou, Z. & Zhao, S. (2024). A Differential Monte Carlo Solver for the Poisson Equation. *ACM SIGGRAPH 2024 Conference Papers*, no. 11, 1–10. https://doi.org/10.1145/3641519.3657460