Penerbit Akademia Baru

Journal of Advanced Research Design

Journal homepage:
https://akademiabaru.com/submit/index.php/ard
ISSN: 2289-7984

Efficient Application of Modified TOR Iterative Method with GPU Acceleration in Robotics and Image Processing

Farhah Athirah Musli[1], Siti Hasnah Tanalol[2], Asni Tahir[2], Andang Sunarto[3], Azali Saudi[1,*]

[1] Faculty of Computing and Informatics, Universiti Malaysia Sabah, Kota Kinabalu, Malaysia
[2] Preparatory Centre for Science and Technology, Universiti Malaysia Sabah, Kota Kinabalu, Malaysia
[3] Tadris Matematika, Universitas Islam Negeri Fatmawati Sukarno, Bengkulu, Indonesia

ARTICLE INFO

Article history:
Received 24 January 2025
Received in revised form 3 March 2025
Accepted 2 May 2025
Available online 16 May 2025

Keywords:
Modified TOR; GPU acceleration; harmonic functions; path planning; image processing

ABSTRACT

This paper presents the application of the Modified TOR (MTOR) method, implemented using CUDA, for solving Poisson's equation through the iterative Finite Difference (FD) method. The study aims to provide a detailed description of the MTOR method for solving Partial Differential Equations (PDEs) on both standard and rotated meshes, employing CUDA for parallel processing. The MTOR method uses a red-black ordering strategy during iteration; consequently, the calculation of red cells utilizes the updated values of their four neighbouring black cells, and vice versa. To evaluate the performance of these iterative methods, robot path planning and image blending problems were tested. Our results highlight the advantages of employing parallel methods on Graphics Processing Units (GPUs) in terms of iterations and computational time, compared to their sequential counterparts. In the path planning simulation, the parallel implementations exhibited over 20 times faster execution compared to their corresponding sequential versions. Similarly, in the image blending problem, the parallel implementations achieved more than a 6-fold improvement in speed compared to the sequential methods.

1. Introduction

In this paper, we develop a parallel numerical technique on a Graphics Processing Unit (GPU) device and provide a comprehensive description of discretization and parallelization for solving partial differential equations. Parallel computing is crucial for handling large-scale problems in fields like weather forecasting, planetary sciences, engineering, and more, where reducing computing time and achieving speedup are critical. Recent studies focus on solving numerical models using various approaches on GPUs, driven by advancements in GPU cards taken from the previous studies [1-3]. Large-scale problems involving sets of PDEs and parameters require high resolutions and significant computational resources. Finite Difference (FD) schemes are widely used in numerical models for engineering and applied science. This work focuses on developing parallel algorithms based on the

* *Corresponding author.*
E-mail address: azali@ums.edu.my

https://doi.org/10.37934/ard.131.1.2646

Modified Two-Parameter Over-Relaxation (MTOR) schemes to solve numerical models. We apply these parallel solvers to solve robot path planning problems using Laplace's equation and image composition problems using Poisson's equation. CUDA C++ and FD schemes are employed to solve the two-dimensional Poisson's equation in parallel, leveraging their effectiveness and simplicity. We implement sequential and parallel versions of these iterative methods, including their modified variants, on standard and rotated meshes. The efficiency and accuracy of the proposed methods are evaluated through their application to robot path planning and image blending problems.

While significant advancements have been made in leveraging GPU-based parallel computing to solve large-scale problems involving partial differential equations (PDEs), challenges remain in optimizing computational efficiency and accuracy for specific applications such as robot path planning and image processing. Existing studies primarily focus on general GPU acceleration techniques, but there is limited exploration of parallelized iterative methods, especially the MTOR schemes, tailored for solving PDEs like Laplace's and Poisson's equations on various mesh configurations. Furthermore, the integration of these methods for practical applications, such as real-time robot navigation and seamless image composition, remains underexplored.

This study addresses these gaps by developing a GPU-based parallel numerical technique employing MTOR schemes to solve PDEs efficiently. The contributions include:

i. Algorithm Development: Design and implementation of parallel algorithms for MTOR schemes on both standard and rotated meshes to solve Laplace's and Poisson's equations.
ii. Application to Real-World Problems: Demonstration of the effectiveness of the proposed methods in solving robot path planning problems using Laplace's equation and image blending tasks using Poisson's equation.
iii. Performance Evaluation: Comprehensive evaluation of the proposed parallel methods in terms of computational efficiency, speedup, and accuracy, using CUDA C++ for high-performance implementations.
iv. Comparative Analysis: Presentation of sequential and parallel versions of the iterative methods, highlighting the benefits of parallelization and the role of mesh configurations.

By addressing computational challenges and demonstrating practical applications, this study advances the use of parallel computing techniques for solving PDE-based problems in engineering and applied sciences.

This paper is organized as follows. In Section 2, both the modeling problem and the two-dimensional decomposition method on the standard and rotated meshes are introduced. The sequential and parallel implementations of the proposed iterative methods are presented, as well as the applications of the Laplace's and Poisson's equations to model the respective path planning and image blending problems are also explained. Section 3 presents the results of numerical experiments on path planning and image blending problems. Finally, the conclusions are provided in Section 4.

2. Methodology

Numerical methods based on PDE have been widely used in robotics and image processing applications. These methods involve solving linear systems of equations generated from the PDE solution on a given mesh. In the previous works [4-9], the established sequential Successive Over-Relaxation (SOR), Accelerated Over-Relaxation (AOR), and Two-Parameter Over-Relaxation (TOR) methods, as well as the modified versions MSOR, MAOR and MTOR are implemented to solve various linear problems. All these iterative methods are implemented on a standard mesh by applying the

standard Full-Sweep (FS) iteration, thus the standard versions are also known as FSSOR, FSAOR and FSTOR, whereas the modified versions are called FSMSOR, FSMAOR, and FSMTOR which extracted from prior studies [9]. The modified versions support parallel processing, and their corresponding implementations are known as Parallel FSMSOR (P-FSMSOR), Parallel FSMSOR (P-FSMAOR) and Parallel FSMTOR (P-FSMTOR) methods, respectively. On a rotated mesh, the Half-Sweep (HS) iteration procedure is applied to reduce computational complexity. Previously, the iterative algorithms implemented on a rotated mesh were utilized to solve a wide range of problems [10-14]. It was shown that the implementation of HS iteration procedure on a rotated mesh drastically speed up the computational time with the same order of accuracies. In the next sections, we present the development of sequential rotated variants, namely HSSOR, HSAOR and HSTOR found in prior studies [10,13,15], whilst the rotated modified variants are called HSMSOR, HSMAOR, and HSMTOR methods drawn from preceding studies [9,14,16].

2.1 Modeling Problem and Discretization

The two-dimensional Poisson's equation is solved in the rectangular mesh $\Omega = [0 \leq x \leq M] \times [0 \leq y \leq N]$ and is written as in Eq. (1).

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y), \tag{1}$$

with boundary conditions in Eq. (2).

$$U(x, y) = g(x, y), (x, y) \in \partial\Omega. \tag{2}$$

The area is divided uniformly into several cells with mesh size h, where and $x_i = ih$ and $y_j = jh$ with $i = 1, 2, 3, \ldots, M$ and $j = 1, 2, 3, \ldots, N$.

Assuming a homogenous mesh, each second derivative in Eq. (1) is approximated with centered FD expressed as in Eq. (3) and (4).

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U(x-h,y)-2U(x,y)+U(x+h,y)}{h^2} \tag{3}$$

$$\frac{\partial^2 U}{\partial y^2} \approx \frac{U(x,y-h)-2U(x,y)+U(x,y+h)}{h^2} \tag{4}$$

Replacing both Eqs. (3) and (4) into Eq. (1) we obtain Eq. (5)

$$U(x-h,y) + U(x+h,y) + U(x,y-h) + U(x,y+h) - 4U(x,y) = h^2 f(x,y) \tag{5}$$

Eq. (5) can be summarized as Eq. (6).

$$U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j} = h^2 f_{i,j} \tag{6}$$

On a rotated mesh [10], the centered FD approximation is given as Eq. (7) and (8)

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U(x-h,y-h)-2U(x,y)+U(x+h,y+h)}{h^2} \tag{7}$$

$$\frac{\partial^2 U}{\partial y^2} \approx \frac{U(x-h,y+h)-2U(x,y)+U(x+h,y-h)}{h^2} \tag{8}$$

Similarly, by substituting Eqs. (7) and (8) into Eq. (1), we obtain Eq. (9).

$$U(x-h,y-h)+U(x+h,y-h)+U(x-h,y+h)+U(x+h,y+h)-4U(x,y) = 2h^2 f(x,y) \tag{9}$$

where it can be rewritten as in Eq. (10).

$$U_{i-1,j-1}+U_{i+1,j-1}+U_{i-1,j+1}+U_{i+1,j+1}-4U_{i,j}=2h^2 f_{i,j}. \tag{10}$$

Eqs. (6) and (10) represent the discretized Poisson's equation for the standard and rotated meshes, respectively. The distance between each cell for the respective standard and rotated meshes are h and $\sqrt{2h}$.

2.2 Sequential Iterative Methods

We should first discuss the sequential iterative methods that are executed on a single CPU before moving on to their corresponding GPU parallel versions.

2.2.1 Full-sweep iteration

The classic Gauss-Seidel (GS) iterative scheme of Eq. (6) can be written as Eq. (11).

$$U_{i,j}^{(k+1)} = \frac{1}{4}\left[U_{i-1,j}^{(k+1)}+U_{i+1,j}^{(k)}+U_{i,j-1}^{(k+1)}+U_{i,j+1}^{(k)}-h^2 f_{i,j}\right] \tag{11}$$

From Eq. (11), we can obtain the SOR iterative scheme that applies a relaxation factor and is written as Eq. (12).

$$U_{i,j}^{(k+1)} = \frac{\omega}{4}\left[U_{i-1,j}^{(k+1)}+U_{i+1,j}^{(k)}+U_{i,j-1}^{(k+1)}+U_{i,j+1}^{(k)}-h^2 f_{i,j}\right]+(1-\omega)U_{i,j}^{(k)} \tag{12}$$

In [5], the AOR iterative scheme was developed and given as in Eq. (13).

$$U_{i,j}^{(k+1)} = \frac{\omega}{4}\left[U_{i-1,j}^{(k)}+U_{i+1,j}^{(k)}+U_{i,j-1}^{(k)}+U_{i,j+1}^{(k)}-h^2 f_{i,j}\right]+\frac{\alpha}{4}\left[U_{i-1,j}^{(k+1)}-U_{i-1,j}^{(k)}+U_{i,j-1}^{(k+1)}-U_{i,j-1}^{(k)}\right]+(1-\omega)U_{i,j}^{(k)} \tag{13}$$

Where, an accelerated parameter α is added. Note that if $\alpha=\omega$, the original SOR is obtained. An extension to the AOR is the TOR method [6] that utilizes two accelerated parameters α and β to provide more tuning options during the iteration procedure. The TOR iterative scheme is given as in Eq. (14).

$$U_{i,j}^{(k+1)} = \frac{\omega}{4}\left[U_{i-1,j}^{(k)}+U_{i+1,j}^{(k)}+U_{i,j-1}^{(k)}+U_{i,j+1}^{(k)}-h^2 f_{i,j}\right]+\frac{\alpha}{4}\left[U_{i-1,j}^{(k+1)}-U_{i-1,j}^{(k)}\right]+\frac{\beta}{4}\left[U_{i,j-1}^{(k+1)}-U_{i,j-1}^{(k)}\right]+(1-\omega)U_{i,j}^{(k)} \tag{14}$$

Obviously, if $\alpha = \beta$, then the TOR will reduce to the AOR method as mentioned in previous study [5]. Eqs. (12), (13), and (14) represent the iterative schemes for FSSOR, FSAOR and FSTOR, respectively.

2.2.2 Half-sweep iteration

Furthermore, the GS iterative scheme on a rotated mesh can be derived from Eq. (10) and given as shown in Eq. (15).

$$U_{i,j}^{(k+1)} = \frac{1}{4}\left[U_{i-1,j-1}^{(k+1)} + U_{i+1,j-1}^{(k)} + U_{i-1,j+1}^{(k+1)} + U_{i+,j+1}^{(k)} - 2h^2 f_{i,j}\right] \tag{15}$$

Based on Eq. (15), the respective iterative schemes for HSSOR, HSAOR and HSTOR methods are given in Eqs. (16), (17) and (18), respectively.

$$U_{i,j}^{(k+1)} = \frac{\omega}{4}\left[U_{i-1,j-1}^{(k+1)} + U_{i+1,j-1}^{(k+1)} + U_{i-1,j+1}^{(k)} + U_{i+,j+1}^{(k)} - 2h^2 f_{i,j}\right] + (1-\omega)U_{i,j}^{(k)} \tag{16}$$

$$U_{i,j}^{(k+1)} = \frac{\omega}{4}\left[U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k)} + U_{i-1,j+1}^{(k)} + U_{i+1,j+1}^{(k)} - 2h^2 f_{i,j}\right] + \frac{\alpha}{4}\left[U_{i-1,j-1}^{(k+1)} - U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k+1)} - U_{i+1,j-1}^{(k)}\right] + (1-\omega)U_{i,j}^{(k)} \tag{17}$$

$$U_{i,j}^{(k+1)} = \frac{\omega}{4}\left[U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k)} + U_{i-1,j+1}^{(k)} + U_{i+1,j+1}^{(k)} - 2h^2 f_{i,j}\right] + \frac{\alpha}{4}\left[U_{i-1,j-1}^{(k+1)} - U_{i-1,j-1}^{(k)}\right] + \frac{\beta}{4}\left[U_{i+1,j-1}^{(k+1)} - U_{i+1,j-1}^{(k)}\right] + (1-\omega)U_{i,j}^{(k)} \tag{18}$$

The HS iteration procedure involves only black cells. During the iteration, all white cells are ignored and are only calculated after the convergence criterion is met using the Gauss-Seidel formula, Eq. (11).

2.2.3 Full-sweep iteration with modified variants

As an extension to the FSSOR, FSAOR and FSTOR methods, their modified variants FSMSOR, FSMAOR and FSMTOR methods were developed by employing Red-Black ordering strategy, as shown in Figure 1 (left), and applying two different relaxation parameters, ω_r and ω_b, for the respective red and black cells extracted from previous studies [7-9]. Modifying the SOR scheme Eq. (12), the iterative scheme for red and black cells of FSMSOR can be written as Eqs. (19) and (20), respectively.

$$U_{i,j}^{(k+1)} = \frac{\omega_r}{4}\left[U_{i-1,j}^{(k)} + U_{i+1,j}^{(k)} + U_{i,j-1}^{(k)} + U_{i,j+1}^{(k)} - h^2 f_{i,j}\right] + (1-\omega_r)U_{i,j}^{(k)}, \tag{19}$$

$$U_{i,j}^{(k+1)} = \frac{\omega_b}{4}\left[U_{i-1,j}^{(k+1)} + U_{i+1,j}^{(k+1)} + U_{i,j-1}^{(k+1)} + U_{i,j+1}^{(k+1)} - h^2 f_{i,j}\right] + (1-\omega_b)U_{i,j}^{(k)}. \tag{20}$$

From Eqs. (13) and (14), the respective iterative schemes of FSMAOR and FSMTOR methods for black cells can be expressed as Eqs. (21) and (22).

$$U_{i,j}^{(k+1)} = \frac{\omega_b}{4}\left[U_{i-1,j}^{(k)} + U_{i+1,j}^{(k)} + U_{i,j-1}^{(k)} + U_{i,j+1}^{(k)} - h^2 f_{i,j}\right] + (1-\omega_b)U_{i,j}^{(k)} + \frac{\alpha}{4}\left[U_{i-1,j}^{(k+1)} - U_{i-1,j}^{(k)} + U_{i,j-1}^{(k+1)} - U_{i,j-1}^{(k)} + U_{i+1,j}^{(k+1)} - U_{i+1,j}^{(k)} + U_{i,j+1}^{(k+1)} - U_{i,j+1}^{(k)}\right] \tag{21}$$

and

$$U_{i,j}^{(k+1)} = \frac{\omega_b}{4}\left[U_{i-1,j}^{(k)} + U_{i+1,j}^{(k)} + U_{i,j-1}^{(k)} + U_{i,j+1}^{(k)} - h^2 f_{i,j}\right] + (1-\omega_b)U_{i,j}^{(k)} + \frac{\alpha}{4}\left[U_{i-1,j}^{(k+1)} - U_{i-1,j}^{(k)} + U_{i,j-1}^{(k+1)} - U_{i,j-1}^{(k)}\right] + \frac{\beta}{4}\left[U_{i+1,j}^{(k+1)} - U_{i+1,j}^{(k)} + U_{i,j+1}^{(k+1)} - U_{i,j+1}^{(k)}\right]. \quad (22)$$

Where, ω_b is the relaxation parameter, α and β are the accelerated parameters. To calculate the red cells of FSMAOR and FSMTOR methods, Eq. (19) is used, which is the same equation employed by the FSMSOR method. This equation only involves the previous values of black cells obtained at the k-th iteration, as depicted in Figure 2(a). The calculation of black cells, on the other hand, employs the updated values of red cells, as shown in Figure 2(b).

2.2.4 Half-sweep iteration with modified variants

On a rotated mesh, where the HS iteration procedure is employed, the modified variants HSMSOR, HSMAOR and HSMTOR apply a Red-Black ordering strategy with a 3-color scheme, as shown in Figure 1. Only red and black cells are calculated in the iteration process. The calculation of the remaining white cells is performed after the iteration has converged using the rotated GS formula, Eq. (15). As depicted in Figures 2(c) and 2(d), on the rotated mesh, red and black cells rely on the updated values of the opposite color of their four neighboring cells. The HSMSOR, HSMAOR and HSMTOR methods are developed based on the rotated schemes as in Eqs. (16), (17) and (18), respectively. These three methods employ the same Eq. (23) to calculate red cells, and the iterative scheme for red cells is given as below.

$$U_{i,j}^{(k+1)} = \frac{\omega_r}{4}\left[U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k)} + U_{i-1,j-1}^{(k)} + U_{i+1,j+1}^{(k)} - 2h^2 f_{i,j}\right] + (1-\omega_r)U_{i,j}^{(k)}. \quad (23)$$

Correspondingly, the respective iterative schemes of these variants to calculate black cells are given as in Eqs. (24) – (26).

$$U_{i,j}^{(k+1)} = \frac{\omega_b}{4}\left[U_{i-1,j-1}^{(k+1)} + U_{i+1,j-1}^{(k+1)} + U_{i-1,j+1}^{(k+1)} + U_{i+,j+1}^{(k+1)} - 2h^2 f_{i,j}\right] + (1-\omega_b)U_{i,j}^{(k)}, \quad (24)$$

$$U_{i,j}^{(k+1)} = \frac{\omega_b}{4}\left[U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k)} + U_{i-1,j+1}^{(k)} + U_{i+1,j+1}^{(k)} - 2h^2 f_{i,j}\right] + (1-\omega_b)U_{i,j}^{(k)} + \frac{\alpha}{4}\left[U_{i-1,j-1}^{(k+1)} - U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k+1)} - U_{i+1,j-1}^{(k)} + U_{i-1,j+1}^{(k+1)} - U_{i-1,j+1}^{(k)} + U_{i+1,j+1}^{(k+1)} - U_{i+1,j+1}^{(k)}\right], \quad (25)$$

$$U_{i,j}^{(k+1)} = \frac{\omega_b}{4}\left[U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k)} + U_{i-1,j+1}^{(k)} + U_{i+1,j+1}^{(k)} - 2h^2 f_{i,j}\right] + (1-\omega_b)U_{i,j}^{(k)} + \frac{\alpha}{4}\left[U_{i-1,j-1}^{(k+1)} - U_{i-1,j-1}^{(k)} + U_{i+1,j-1}^{(k+1)} - U_{i+1,j-1}^{(k)}\right] + \frac{\beta}{4}\left[U_{i-1,j+1}^{(k+1)} - U_{i-1,j+1}^{(k)} + U_{i+1,j+1}^{(k+1)} - U_{i+1,j+1}^{(k)}\right]. \quad (26)$$

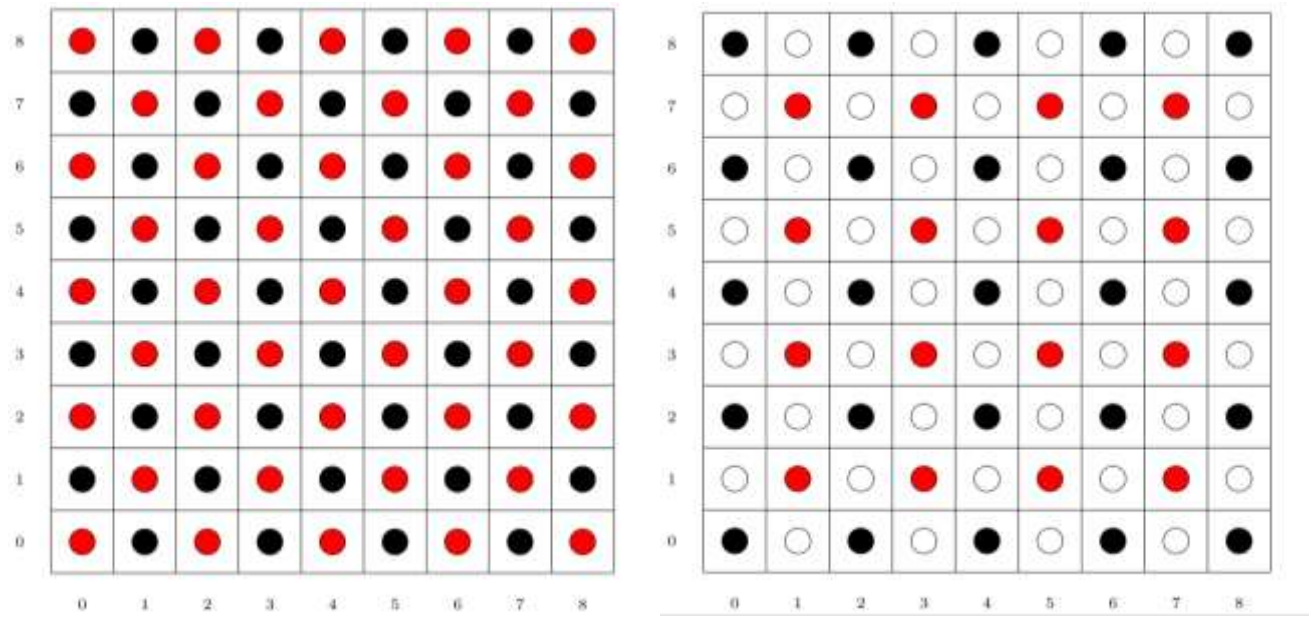
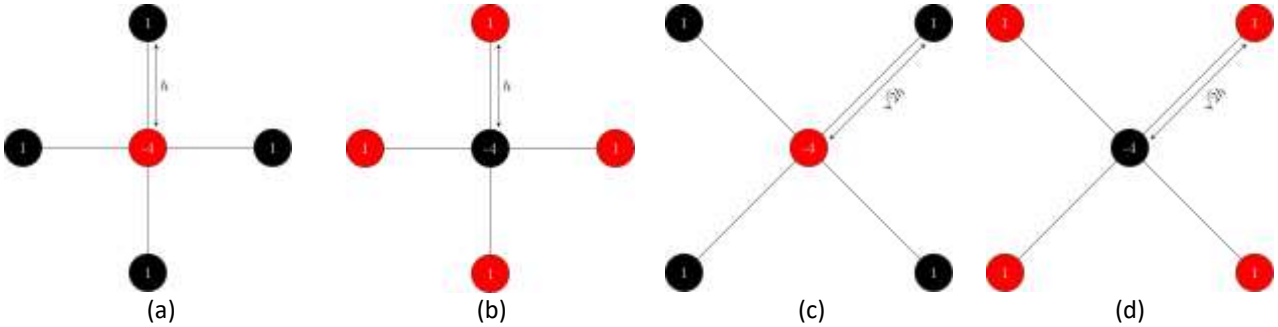


Fig. 1. All red and black cells are calculated alternately in the standard mesh (left). With rotated mesh (right), only red and black are calculated during the iteration procedure, the remaining white cells are calculated after the convergence is achieved



(a) (b) (c) (d)

Fig. 2. The computational molecules for (a) red and (b) black cells on the standard mesh, and the respective molecules for (c) red and (b) black cells on the rotated mesh. The computational molecules for red and black cells are symmetrical, where the black node applies the updated values of its four neighboring red cells and *vice versa*

Algorithm 1 provides the details of the HSMTOR method, where if the parameters are set as $\alpha = \beta$ or $\alpha = \beta = \omega$, the respective implementation of HSMAOR or HSMSOR are obtained. The conditions in Line 6 and 11 are applied to identify the red and black cells, respectively. The loop in Line 19 to 23 is performed to calculate the remaining white cells that are ignored during the main iteration, where the condition in Line 20 is used to identify the white cells.

Algorithm 1: The function to calculate cells using the Half-Sweep Modified variants

function CALCULATECELLSWITHHSMTOR (X, Y, N, ω_r, ω_b, α, β, ϵ)
  $t_s \leftarrow$ recordStartTime()
  $k \leftarrow 0$
  repeat
    for $i, j \leftarrow 1$ to $N - 2$ **do**
      if i and j are odd **then** ▷ Red cells
        $Y_{i,j} \leftarrow \frac{\omega_r}{4}[X_A+X_B+X_C+X_D-T]+(1+\omega_r)X_{i,j}$

```
      end if
    end for
    for i, j ← 1 to N − 2 do
      if i and j are even then                                   ▷ Black cells
        Y_{i,j} ← (ω_b/4) [X_A+X_B+X_C+X_D-T]+(1+ω_r)Xi,j+ (α/4) [Y_A-X_A+Y_B-X_B]+ (β/4) [Y_C-X_C+Y_D-X_D]
      end if
    end for
    E ← calculateError(X, Y )
    k← k+1
  until E ¡ ∈
  for i, j ← 1 to N − 2 do          ▷Calculate the remaining white cells using direct method
    if i + j is odd then                                         ▷ White cells
      Y_{i,j} ← (1/4)[Y_A+Y_B+Y_C+Y_D-T]
    end if
  end for
  t_e ← recordElapseTime(t_s)
  return Y, E, k, t_e
end function
```

2.3 Parallel Iterative Methods

The GPU, initially designed for graphics rendering, has become a valuable tool for general-purpose computing due to its unique hardware design. It consists of multiple simpler CPUs with faster memory access and concurrent computational power. GPU computing is well-suited for highly parallel applications with simple workflows, low memory requirements, and infrequent communication. CUDA, a parallel computing architecture, utilizes the GPU's parallel computing structure to solve complex computing problems. For further information on CUDA programming, refer to [17]. Parallel solutions to Eq. (1) can only be implemented using Red-Black ordering. Therefore, only the Modified variants, as described in Sections 2.1.3 and 2.1.4, are developed for parallel implementations.

Let array variables $U = U^{(k)}$ and $V = U^{(k+1)}$ store the host memory values at iterations (k) and $(k+1)$, respectively. Similarly, array variables U_d and V_d are the respective previous (k) and updated $(k+1)$ device memory values. Thus, unlike the sequential CPU iteration procedure, initial values of the solutions need to be copied from host memory U and V to device memory U_d and V_d, before the iteration begins using the CUDA API function **cuMemcpyHtoD**. After the convergence is achieved, the results in the device memory $V_d$are copied back to the host memory Vusing CUDA API function **cuMemcpyDtoH**, for further processing. This procedure is described in Algorithm 2. In Line 3 and 4, the contents of host memory U and V are copied to device memory U_d and uv, respectively. In the main iteration, the CUDA kernel functions calculateRedCells and calculateBlackCells are called to calculate the respective red and black cells alternately. This iteration continues until the condition $E \leq \varepsilon$ is achieved. If the tested iterative method P is of Half-Sweep type, the calculation of the remaining white cells in Line 13 is performed. The final updated results are obtained by copying the contents of device memory V_d to host memory V as shown in Line 15.

Algorithm 2: GPU Iteration procedure

procedure ITERATEWITHGPU(P, U, U_d , V, V_d , N, $\in$)
▷ P is the solver
  $t_s \leftarrow$ recordStartTime()
  cuMemcpyHtoD(U_d, U, i) ▷ Copy from host U to device memory U_d
  cuMemcpyHtoD(V_d , V, N)
  $k \leftarrow 0$
  while E > $\in$ **do** ▷ E is the maximum error
    calculateRedCells(U_d , V_d, N, ω_r)
    calculateBlackCells(U_d ,V_d , N, ω_b, α, β)
    $E \leftarrow$ calculateError(U_d , V_d, N)
    $k \leftarrow k$+1
  end while
  if isHalfSweep(P) **then**
    calculateWhiteCells(U_d , V_d, N) ▷ Calculate the remaining white cells
  end if
  cuMemcpyDtoH(V, V_d, N) ▷ Copy from device V_d to host memory V
  $t_e \leftarrow$ recordElapseTime(t_s)
  return V, k, t_e , E
end procedure

The red and black cell calculation kernels, presented in Algorithms 3 and 4, respectively, are implemented as **__global__** functions. In CUDA, built-in variables like **blockIdx.x**, **blockIdx.y**, **blockDim.x**, **blockDim.y**, **threadIdx.x**, and **threadIdx.y** are used to obtain block and thread indices. These indices are used to assign device memory locations to stream processors. The neighboring cell indices are denoted by A, B, C and D, while T represents a constant, and X and Y store the previous and updated solution values. Relaxation parameters ω_r and ω_b are used for red and black cells, respectively. Algorithm 3 calculates red cells using Eq. (19) or (23), and Algorithm 4 implements Parallel Modified TOR for black cell calculation, with accelerated parameters.

Algorithm 3: Kernel function to calculate Red cells

__global__
function CALCULATEREDCELLS (X, Y, N, ω_r)
  $i \leftarrow$ blockIdx.x×blockDim.x + threadIdx.x
  $j \leftarrow$ blockIdx.y×blockDim.y + threadIdx.y
  if $1 \le i, j \le N - 2$ **and** cellColor(i, j) is red **then** ▷ Border cells are skipped in the calculation
    $Y_{i,j} \leftarrow \frac{\omega_r}{4}[X_A+X_B+X_C+X_D-T]+(1+\omega_r)X_{i,j}$
  end if
end function

The conditions in Line 5 ensure that the indices are within the allowed range, and the function cellColor is used to identify cell colors. Note that border cells are excluded from the iteration procedure. Line 6 performs double-precision floating-point calculations.

Algorithm 4: Kernel function to calculate Black cells

__global__
function CALCULATEBLACKCELLS (X, Y, N, ω_b, α, β)
  $i \leftarrow$ blockIdx.x×blockDim.x + threadIdx.x
  $j \leftarrow$ blockIdx.y×blockDim.y + threadIdx.y
  if $1 \le i, j \le N - 2$ **and** cellColor(i, j) is black **then** ▷ Border cells are skipped in the calculation
    $Y_{i,j} \leftarrow \frac{\omega_b}{4}[X_A+X_B+X_C+X_D-T]+(1+\omega_r)X_{i,j}+\frac{\alpha}{4}[Y_A-X_A+Y_B-X_B]+\frac{\beta}{4}[Y_C-X_C+Y_D-X_D]$
  end if
end function

In half-sweep iterations, only half of the available GPU memory is used for alternating red and black cell calculations. Future research will explore more efficient utilization of CUDA threads by avoiding conditional statements.

2.4 Harmonic Path Planning

Achieving autonomous behavior in robotics remains a significant goal, necessitating progress in key areas such as localization, mapping, and planning. Among these, path planning has garnered substantial attention in recent years, particularly in the context of mobile robots [18,19]. This critical aspect of robotics, discussed in this section and formulated using Eq. (1), involves determining a collision-free trajectory for a robot to efficiently reach its intended target. The ability to generate such paths quickly is especially vital in time-sensitive scenarios, such as life-saving operations during natural disasters, where delays can have severe consequences.

The artificial potential field approach [20,21] is a widely used algorithm in the field of path planning due to its simplicity and efficiency in generating a direct route from the starting point to the goal. Its speed makes it particularly suitable for real-time applications; however, the approach encounters challenges in scenarios where deadlocks occur. These deadlock situations, where the robot becomes trapped in a local minimum, can be effectively resolved using harmonic functions, as detailed in [22].

Harmonic functions, which are solutions to Laplace's equation, offer unique advantages for path planning, as elaborated in [23]. Their mathematical properties, such as adherence to the min-max principle, ensure that deadlocks are avoided under specific conditions, thereby improving the robustness of the planning process. Moreover, these functions provide a framework for generating smooth and obstacle-free trajectories. By incorporating harmonic functions into path planning, it is possible to achieve safe navigation around obstacles, ensuring that the robot can reach its target with enhanced reliability and efficiency. This combination of speed, smoothness, and safety makes the strategy highly advantageous for diverse applications.

A harmonic function on a two-dimensional domain is a function that satisfies Laplace's equation:

$$\nabla^2 = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0 \tag{27}$$

In path planning, obstacles, walls, and goals define the boundary. This work computes harmonic functions over a grid representing the robot's environment. Obstacles have a high constant potential, while goal regions have a low potential, using Dirichlet boundary conditions. Sequential and parallel methods proposed in Sections 2.1 and 2.2 are examined to solve the path planning problem and their

performances are analyzed. The resulting potentials guide the robot away from obstacles by satisfying $\nabla^2 U = 0$ in free space. The path tracing algorithm, utilizing gradient descent search, traces a path from the start point to the goal point, proceeding to the next lower potential among the eight neighbouring points. Algorithm 5 outlines the workflow of this path planning strategy, employing Laplace's equation, Eq. (27) to compute harmonic potentials for generating paths.

Algorithm 5: Path Planning strategy

```
procedure PERFORMPATHPLANNING (P, L, G)          ▷ P is the chosen method
    (U, W ) ← convertMapToMatrix(L, G)           ▷ L stores the map, G is a set of goal points
    V ← calculatePotentials(P, U, W )            ▷ W indicates the occupied cells
    Q ← gradientDescentSearch(V, S, W )          ▷ S is a set of start points
    return Q                                     ▷ Q is the set of generated paths
end procedure
```

In the PerformPathPlanning procedure, the convertMapToMatrix function converts the map (L) and set of goal points (G) into a matrix variable (W), where each pixel is represented by an integer indicating free space (0), occupied cells (1), or goal regions (2). The initial potentials are stored in the matrix variable (U), assigning high potential (1.0) to occupied cells, low potential (0.0) to goal points, and intermediate values (0.0-1.0) to free cells. The function calculatePotentials wraps the iterative method (P) to compute harmonic potentials, returning the updated potentials (V) in the matrix variable. Sequential computations were performed in previous studies [15,16,22,23]. Finally, the gradientDescentSearch function utilizes the obtained harmonic potentials (V) to generate paths (Q) from start points (S) to specified goal points indicated in (W).

2.5 Poisson Image Blending

An image processing application that combines several images into a single composite image is known as image blending. Image blending is used in image editing, panoramic stitching, and image morphing, among other things. Color and lighting variations in pictures are noticeable to human eyes. The goal of image blending is to produce seamless transitions between image segments that come from various sources. Image blending is a well-researched problem. To create smooth composite images, various algorithms have been proposed. Recent works on blending methods in gradient domain can be found in [14,24-26].

Let G represent the image domain, and Ω be a subset of with boundary Ω. Let f^* be a known scalar function defined over G minus the interior of Ω, and f and be an unknown scalar function defined over the interior of Ω. Finally, let v be a gradient vector defined over Ω. The composition f of f^* over Ω satisfies the minimization problem and is shown in Eq. (28).

$$\min_f \iint_\Omega |\nabla f - v|^2 \text{ with} f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{28}$$

whose solution is the unique solution of Poisson's Eq. (29) with Dirichlet boundary conditions,

$$\Delta f = divv\text{over}\Omega f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{29}$$

Where, $divv = \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}$ is the divergence of $v = (u, v)$. Since domain, Eq. (29) is reduced to a discrete form with Dirichlet boundary conditions as in Eq. (30),

$$\min_{f|_\Omega} \sum_{(p,q)\cap\Omega} \left(f_p - f_q - v_{pq}\right)^2 \tag{30}$$

such that $f_p = f_p^*$ for all $p \in \partial\Omega$,where f_p is the intensity of one of the four neighbours of the current pixel p, $v_{pq}$is the first-order derivative of the source image, and f_p^* is the intensity of the pixel on the boundary. In the preceding study [27], Eq. (30) is minimized by making the Laplacian operator Δf equals to the Laplacian operator of the source image ∇v as in Eq. (31).

$$\Delta f_p = \nabla v_p \text{for all} p \in \Omega \tag{31}$$

Hence, the intensity of each pixel in Ω for the generated image can be obtained using the following Eq. (32).

$$\left|N_p\right| f_p - \sum_{q\in N_p} f_q = \sum_{q\in N_p} v_{pq} \tag{32}$$

Where, N_p is the set of the four neighbours of the current pixel p. Solving Eq. (32) generates a sparse linear system that has a size $\times N$, where N is the number of pixels. The solution to Eq. (32) can be obtained using the established iterative methods described in Sections 2.1 and 2.2 To perform image blending process in RGB color space, three equations of the form Eq. (32) are solved independently in the three (red, green, and blue) color channels. Further details on Poisson image editing are given in past studies [27,28].

2.5.1 Image similarity measurements

Several image quality measurement techniques are available to compare the similarity between the final images produced by the tested methods. Based on the statistical method ANOVA [29], three metrics were applied namely Mean Square Error (MSE), Structural Similarity Index (SSIM) and Structural Content (SC) as mentioned in prior works [30,31].

The MSE measurement value is given as in Eq. (33).

$$MSE = \frac{1}{mn} \sum_{i=j}^{m} \sum_{j=1}^{n} \left(A_{ij} - B_{ij}\right)^2 \tag{33}$$

Where, A and B represent the pixel values of reference and target images, and $(m \times n)$ is the size of the image. MSE is used to measure the difference between pixel values in A and B, in which a smaller value means higher similarity. The ideal MSE value of 0 is obtained when the two images A and B are identical.

Another similarity test that can be used to compare the two images is SSIM as follows in Eq. (34).

$$SSIM(x, y) = \frac{(2\phi_x\phi_y + C_1)(2\tau_{xy} + C_2)}{(\phi_x^2 + \phi_y^2 + C1)(\tau_x^2 + \tau_y^2 + C_2)} \tag{34}$$

Where, ϕ_x,ϕ_y and τ_x,τ_y denote the mean intensity and standard deviation set of image block and image block x, respectively, while τ_{xy} denote their cross-correlation. C_1 and C_2 are small constants value to avoid instability problems when the denominator is too close to zero. If the obtained SSIM is equal to 1, it means the two images are identical.

Similarly, if SC value is equal to 1, it indicates that the two images are identical. The SC measurement value is written as in Eq. (35).

$$SC = \frac{\sum_{i=j}^{m}\sum_{j=1}^{n}(A_{ij})^2}{\sum_{i=j}^{m}\sum_{j=1}^{n}(B_{ij})^2} \tag{35}$$

These three metrics are applied to assess the similarity of the images produced by the proposed methods.

3. Results

This section presents the simulation results of applying the considered iterative methods to obtain the solutions of Laplace's and Poisson's equations for solving path planning and image blending problems, respectively. All iterative methods were executed on the same machine running Xubuntu 20.04 on Intel i5 3570K CPU running at 3.40GHz with 12GB of RAM. The parallel implementations were simulated on Nvidia GeForce RTX 3060 GPU.

3.1 Experiment on Path Planning

The experiment involved various sizes of static environments, with different goal points, starting positions, and wall setups (e.g., N_1 = 330×270, N_2 = 660×540, N_3 = 990×810, N_4 = 1320×1080). Walls and obstacles had a high potential of 1, while the goal area had a very low potential of 0. Free spaces had a constant potential between 0 and 1. The path planning simulator was implemented in Java and is provided in [32]. The iterative methods in Sections 2.1 and 2.2 were applied to numerically compute harmonic potential values, stopping when convergence was achieved. To prevent flat areas hindering goal reachability, a maximum precision solution for Eq. (27) was necessary. Table 1 displays the iteration count and computational time (in seconds) for computing harmonic potentials. AOR variants were slightly faster than SOR, and TOR variants outperformed both.

Table 1
Simulation results in computing the harmonic potentials in terms of number of iterations and execution time (in seconds). Map size: N_1 = 330×270, N_2 = 660×540, N_3 = 990×810, N_4 = 1320×1080; Relaxation parameters: ω_b = 1.80, ω_r = 1.82; Accelerated parameters: α = 1.84, β = 1.86

Method	Iterations				Time			
	N_1	N_2	N_3	N_4	N_1	N_2	N_3	N_4
FSSOR	6212	23988	53256	92438	2.799	53.473	404.615	1352.573
FSAOR	5004	19369	43033	74768	2.487	45.011	333.312	1073.432
FSTOR	4700	18209	40490	70368	2.314	42.830	307.926	1002.658
HSSOR	3178	12321	27404	47765	1.133	22.346	175.710	549.628
HSAOR	2547	9934	22150	38595	1.042	18.710	159.187	476.524
HSTOR	2384	9331	20813	36315	0.992	16.442	141.212	441.592
FSMSOR	5886	22702	50461	87624	2.493	46.529	395.447	1067.953
FSMAOR	4663	18077	40224	69937	2.299	40.921	312.588	962.097
FSMTOR	4356	16919	37660	65461	2.116	37.048	290.222	939.386
HSMSOR	3005	11660	25968	45227	1.056	24.325	160.588	524.213

HSMAOR	2372	9261	20693	36064	0.957	18.084	132.668	439.635
HSMTOR	2207	8666	19359	33745	0.902	17.207	125.094	417.201
P-FSMSOR	5875	22659	50309	87343	0.490	3.363	14.226	39.107
P-FSMAOR	4653	18017	40050	69619	0.487	3.204	13.738	38.523
P-FSMTOR	4348	16861	37523	65232	0.436	3.067	13.515	37.158
P-HSMSOR	3004	11644	25928	45081	0.320	1.688	6.844	19.904
P-HSMAOR	2364	9241	20620	35912	0.288	1.671	6.765	18.944
P-HSMTOR	2202	8639	19309	33620	0.291	1.606	6.652	18.666

Figure 3 illustrates the generated paths for an environment of size 330×270, in which several different start (green) and goal (red) positions were tested. Since identical outputs were obtained, the paths generated for other sizes of environment are not shown. Full-Sweep Modified and Half-Sweep Modified variants were faster than their standard counterparts. Half-Sweep methods were superior, reducing iterations by half and significantly speeding up computation (2x faster). For CPU implementations, Half-Sweep Modified variants (HSMSOR, HSMAOR and HSMTOR) had the lowest execution time, with HSMTOR being the best. Parallel rotating versions (P-HSMSOR, P-HSMAOR and P-HSMTOR) outperformed regular versions (P-FSMSOR, P-FSMAOR and P-FSMTOR), with P-HSMTOR being the most efficient. Computation increased exponentially with larger environment sizes for all iterative methods. The sequential Full-Sweep Modified variants outperformed their Standard counterparts by 5-7% in terms of iteration reduction and 6-9% in time reduction. Half-Sweep Modified variants reduced iterations by 5-7% and execution time by 5-10% compared to Half-Sweep Standard variants. Half-Sweep approaches significantly outperformed Full-Sweep techniques, reducing iterations by 48-49% and execution time by 55-57%.

Full-Sweep Modified and Half-Sweep Modified variants were faster than their standard counterparts. Half-Sweep methods were superior, reducing iterations by half and significantly speeding up computation (2x faster). For CPU implementations, Half-Sweep Modified variants (HSMSOR, HSMAOR and HSMTOR) had the lowest execution time, with HSMTOR being the best. Parallel rotating versions (P-HSMSOR, P-HSMAOR and P-HSMTOR) outperformed regular versions (P-FSMSOR, P-FSMAOR and P-FSMTOR), with P-HSMTOR being the most efficient. Computation increased exponentially with larger environment sizes for all iterative methods. The sequential Full-Sweep Modified variants outperformed their Standard counterparts by 5-7% in terms of iteration reduction and 6-9% in time reduction. Half-Sweep Modified variants reduced iterations by 5-7% and execution time by 5-10% compared to Half-Sweep Standard variants. Half-Sweep approaches significantly outperformed Full-Sweep techniques, reducing iterations by 48-49% and execution time by 55-57%.

Fig. 3. The generated paths for an environment of size 330×270 with several different starting (red) and goal positions (green)

Iteration counts showed no significant differences between parallel and sequential versions, attributed to variations in math library implementations. However, parallel versions were 20-28 times faster than sequential versions. Notably, there were no significant differences in iteration reduction between sequential and parallel versions. In this path planning simulation for computing harmonic potentials, rotating parallel versions achieved nearly 2x fewer iterations and execution time compared to regular parallel versions. Once the potential values were obtained using the examined iterative methods, the gradient descent search algorithm would utilize them to guide its exploration.

3.2 Experiment on Image Blending

In this experiment, two sets of images are used to assess the performance of the methods under consideration. Table 2 shows the number of pixels inside the image masks that were applied for the two image sets. The number of iterations, execution time, and image quality of the examined methods are all recorded and evaluated. Optimal relaxation factor and accelerated parameter values are required in all variants. Initial results from a trial-and-error method indicate that all parameter values should be between 1.5 and 1.9. Table 3 shows that values between 1.62 and 1.76 were chosen based on these findings.

Table 2
Number of pixels inside the image masks

Item	Sky and ballons	Lake and crocodile
Number of pixels	49,506	39,617

The similarity index is used to compare the picture quality, as defined in Section 2.4.1. The two images set that include source, target, mask, and initial images are shown in Figure 4. From Table 3, it can be observed that the Full-Sweep Standard (FSSOR, FSAOR and FSTOR) and Full-Sweep Modified (FSMSOR, FSMAOR and FSMTOR) approaches take at least 600 iterations to converge for sky and balloon images.

Table 3
Computational cost for tested images. The relaxation factors are ω_b = 1.62, ω_r = 1.68 and accelerated parameters are α = 1.72, β = 1.76. The two sets of images are: (A) Sky and ballons, and (B) Lake and crocodile

	Iterations		Time	
Method	A	B	A	B
FSSOR	862	1443	9.725	12.964
FSAOR	680	1144	8.051	10.957
FSTOR	641	1080	7.467	9.712
HSSOR	464	777	3.684	5.134
HSAOR	363	611	2.965	4.170
HSTOR	341	576	2.778	3.824
FSMSOR	793	1328	9.436	12.619
FSMAOR	606	1020	7.109	9.656
FSMTOR	565	953	6.634	8.926
HSMSOR	426	713	3.472	4.762
HSMAOR	322	543	2.662	3.710
HSMTOR	299	506	2.437	3.422
P-FSMSOR	793	1328	0.791	1.080
P-FSMAOR	606	1020	0.677	0.871
P-FSMTOR	565	953	0.589	0.790
P-HSMSOR	426	713	0.490	0.668
P-HSMAOR	322	543	0.441	0.549
P-HSMTOR	299	506	0.369	0.472

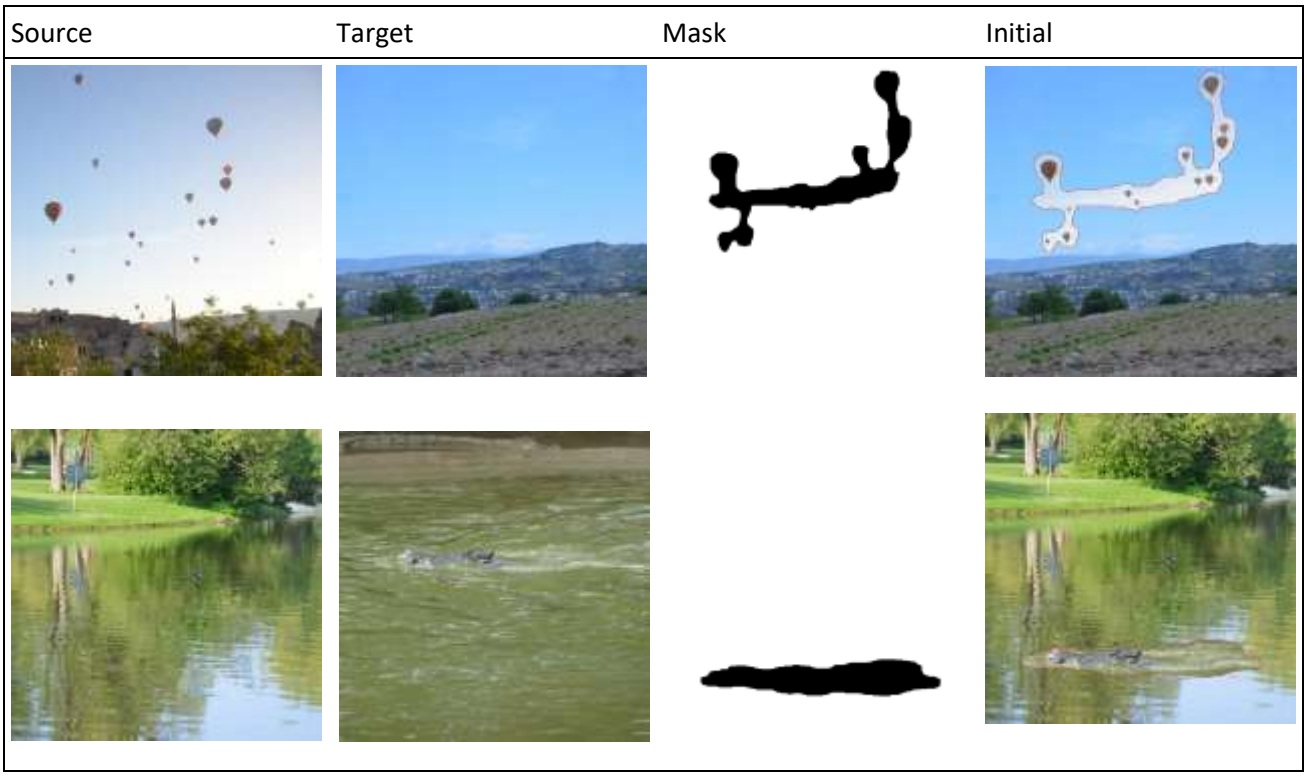


Fig. 4. The source, target, mask and initial images of sky and balloons scene (top) and lake and crocodile (bottom)

The Half-Sweep Standard (HSSOR, HSAOR and HSTOR) and the Half-Sweep Modified (HSMSOR, HSMAOR and HSMTOR) approaches cut the number of iterations necessary in half, in which the gradual improvement of the blending process using Half-Sweep iterations is depicted in Figure 5. The rotated Half-Sweep Modified variants (HSMSOR, HSMAOR and HSMTOR) outperform the regular Full-

Sweep Modified variants (FSMSOR, FSMAOR and FSMTOR) in terms of iteration and computational time. Among them, HSMTOR has the fewest iterations and fastest execution time. These results strongly support the superiority of HSMTOR. The Modified variants are well-suited for parallel processing due to their utilization of Red-Black ordering approaches.

The parallel implementations on regular (P-FSMSOR, P-FSMAOR and P-FSMTOR) and rotated (P-HSMSOR, P-HSMAOR and P-HSMTOR) grids require the same number of iterations as their respective sequential versions. However, these parallel implementations are significantly faster than the sequential ones, with the parallel Half-Sweep Modified variants delivering the shortest execution time. The parallel versions on regular grids take 500 milliseconds to 1 second for image blending, while the rotating parallel versions take less than 500 milliseconds for image A and 400 to 700 milliseconds for image B. The Full-Sweep Modified variants reduced iterations by 8 to 12% and were 3 to 12% faster compared to the Full-Sweep Standard variants. The Half-Sweep Modified variants outperformed the Half-Sweep Standard variants, reducing iterations and execution time by 8 to 12% and 6 to 11% respectively. The sequential rotating Standard and Modified variants were superior to their regular counterparts, with a 46 to 47% reduction in iterations and a 61 to 63% improvement in execution time. The parallel versions required the same iterations as their sequential counterparts but were 6x to 12x faster. Rotating parallel implementations were faster than regular parallel versions, reducing iterations and time by 46 to 47% and 38 to 39% correspondingly.

10^{th} iteration	100^{th} iteration	200^{th} iteration	300^{th} iteration
HSSOR			
			
HSAOR			
			

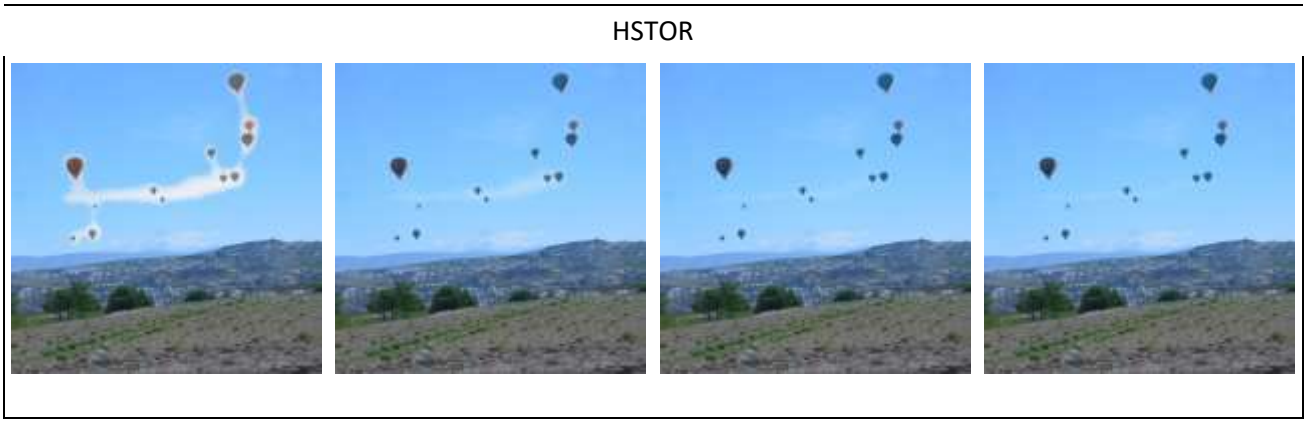


Fig. 5. Illustration of image blending process using the Half-Sweep approaches at different iterations

The resulting blended images of the lake and crocodile were virtually indistinguishable across all the tested methods, as evidenced by the similarity metrics presented in Table 4. The MSE values were remarkably close to 0, indicating minimal pixel-wise differences between the images generated by the different methods.

Additionally, the SSIM and SC values were consistently close to 1, further confirming the high degree of visual and structural similarity between the generated images (Figure 6). These metrics collectively validate the identical nature of the output images across the methods, highlighting the robustness and reliability of the blending techniques evaluated in this study.

Table 4
Similarity measurement for the generated images

Methods	MSE	SSIM	SC
FSSOR	0.06491	0.99995	1.00107
FSAOR	0.07592	0.99995	1.00115
FSTOR	0.07835	0.99995	1.00117
FSMSOR	0.06744	0.99995	1.00109
FSMAOR	0.07103	0.99995	1.00112
FSMTOR	0.07315	0.99995	1.00113
HSMSOR	0.07777	0.99993	1.00115
HSMAOR	0.08120	0.99993	1.00117
HSMTOR	0.08287	0.99993	1.00118
P-FSMSOR	0.06678	0.99995	1.00108
P-FSMAOR	0.07942	0.99995	1.00118
P-FSMTOR	0.08158	0.99995	1.00119
P-HSMSOR	0.07866	0.99993	1.00116
P-HSMAOR	0.08763	0.99993	1.00122
P-HSMTOR	0.08923	0.99993	1.00123

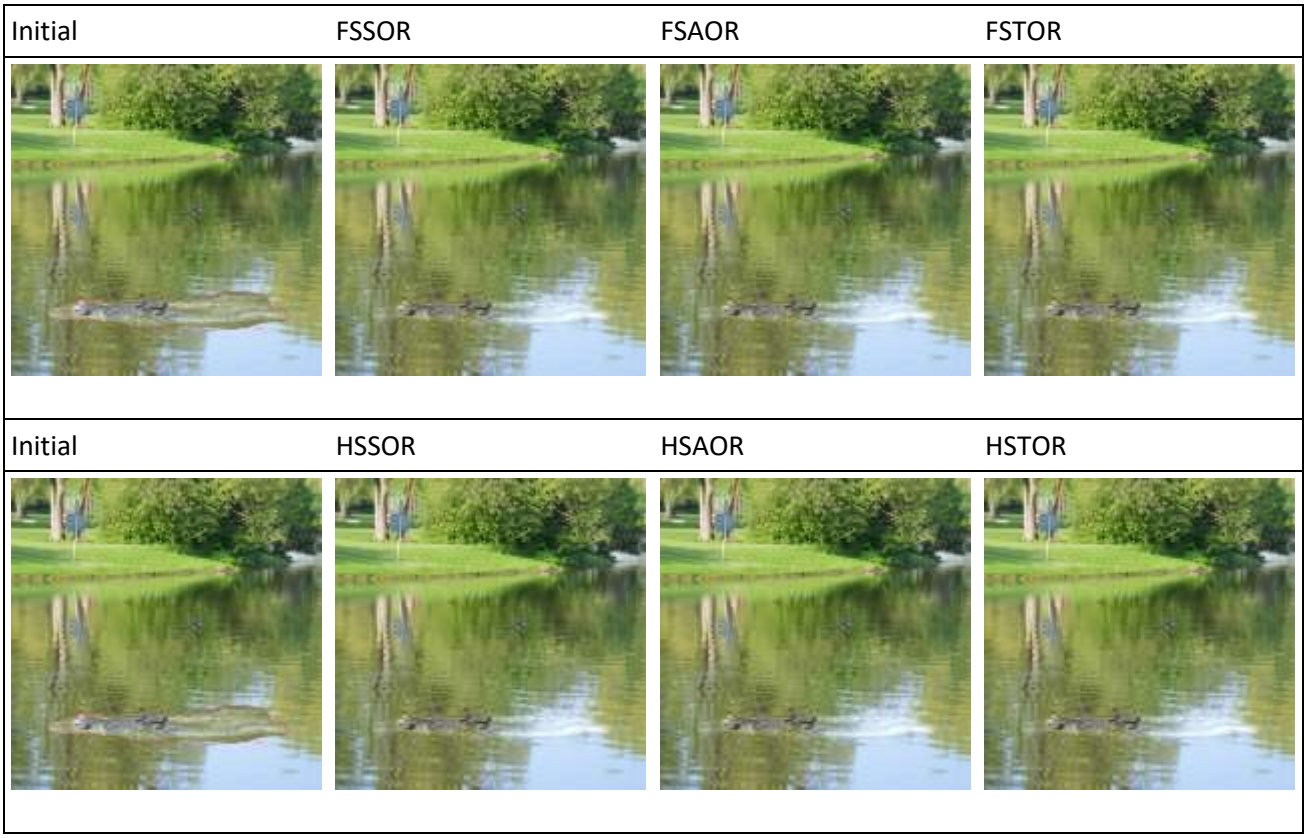


Fig. 6. The output images

4. Conclusion

The SOR, AOR and TOR iterative methods have been successfully used for path planning simulation and image blending. Red-Black ordering with MSOR, MAOR and MTOR schemes enables efficient parallel processing on platforms like CUDA. On CPU, HS Standard and HS Modified methods on a rotated grid were approximately 100% faster than their FS Standard and FS Modified counterparts on a regular grid. The parallel implementations P-FSMSOR, P-HSMAOR and P-HSMTOR on a regular grid outperformed CPU implementation, but P-HSMSOR, P-HSMAOR and P-HSMTOR on a rotated grid were the most efficient, with P-HSMTOR delivering the best performance. Fast parallel processing is crucial for quick responses in dynamic environments during path planning, and the Red-Black strategy of MTOR can also be applied to real-time video processing. Future work will explore more powerful iterative methods, such as quarter-sweep techniques and block iteration.

References

[1] Chow, Alex D., Benedict D. Rogers, Steven J. Lind, and Peter K. Stansby. "Incompressible SPH (ISPH) with fast Poisson solver on a GPU." *Computer Physics Communications* 226 (2018): 81-103. https://doi.org/10.1016/j.cpc.2018.01.005

[2] Chen, Jen-Hao, Ren-Chuen Chen, and Jinn-Liang Liu. "A GPU Poisson–Fermi solver for ion channel simulations." *Computer Physics Communications* 229 (2018): 99-105. https://doi.org/10.1016/j.cpc.2018.04.002

[3] Kim, Boram, Kwang Seok Yoon, and Hyung-Jun Kim. "Gpu-accelerated laplace equation model development based on cuda fortran." *Water* 13, no. 23 (2021): 3435. https://doi.org/10.3390/w13233435

[4] Young, David M. *Iterative solution of large linear systems*. Elsevier, 2014.

[5] Hadjidimos, Apostolos. "Accelerated overrelaxation method." *Mathematics of Computation* 32, no. 141 (1978): 149-157. https://doi.org/10.1090/S0025-5718-1978-0483340-6

[6] Kuang, Jiaoxun, and Jun Ji. "A survey of AOR and TOR methods." *Journal of computational and applied mathematics* 24, no. 1-2 (1988): 3-12. https://doi.org/10.1016/0377-0427(88)90340-8

[7] Kincaid, David R., and David M. Young. "The modified successive overrelaxation method with fixed parameters." *Mathematics of Computation* 26, no. 119 (1972): 705-717. https://doi.org/10.1090/S0025-5718-1972-0331746-2
[8] Hadjidimos, A., A. Psimarni, and A. K. Yeyios. "On the convergence of the modified accelerated overrelaxation (MAOR) method." *Applied numerical mathematics* 10, no. 2 (1992): 115-127. https://doi.org/10.1016/0168-9274(92)90034-B
[9] Musli, F. A., J. Sulaiman, and A. Saudi. "Numerical simulations of agent navigation via half-sweep modified two-parameter over-relaxation (HSMTOR)." In *Journal of Physics: Conference Series*, vol. 1988, no. 1, p. 012035. IOP Publishing, 2021. https://doi.org/10.1088/1742-6596/1988/1/012035
[10] Abdullah, Abdul Rahman. "The four point Explicit Decoupled Group (EDG) method: A fast Poisson solver." *International Journal of Computer Mathematics* 38, no. 1-2 (1991): 61-70. https://doi.org/10.1080/00207169108803958
[11] Ali, Norhashidah Hj Mohd, and Lee Siaw Chong. "Group accelerated overrelaxation methods on rotated grid." *Applied mathematics and computation* 191, no. 2 (2007): 533-542. https://doi.org/10.1016/j.amc.2007.02.131
[12] Saudi, Azali, and Jumat Sulaiman. "Robot path planning using Laplacian behaviour-based control (LBBC) via half-sweep SOR." In *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, pp. 424-429. IEEE, 2013. https://doi.org/10.1109/TAEECE.2013.6557312
[13] Muthuvalu, Mohana Sundaram, and Jumat Sulaiman. "Performance analysis of Half-Sweep AOR iterative method in solving second kind Linear Fredholm Integral Equations." In *2014 International Conference on Computational Science and Technology (ICCST)*, pp. 1-5. IEEE, 2014. https://doi.org/10.1109/ICCST.2014.7045197
[14] Saad, Nordin, and Azali Saudi. "Modified Poisson compositing technique on skewed grid." *AIMS Mathematics* 7, no. 2 (2022): 2176-2194. https://doi.org/10.3934/math.2022124
[15] Dahalan, A'qilah Ahmad, and Azali Saudi. "Rotated TOR-5P Laplacian iteration path navigation for obstacle avoidance in stationary indoor simulation." In *Advances in Robotics, Automation and Data Analytics: Selected Papers from iCITES 2020*, pp. 285-295. Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-70917-4_27
[16] Suparmin, Sumiati, and Azali Saudi. "Path planning in structured environment using harmonic potentials via half-sweep modified AOR method." *Advanced Science Letters* 24, no. 3 (2018): 1885-1891. https://doi.org/10.1166/asl.2018.11182
[17] Guide, Design. "CUDA C++ programming guide." *NVIDIA, July* (2020).
[18] Hamd, Mostafa Mohammed Massoud, Ahmed Abdellatif Hamed Ibrahim, and Mostafa Rostom Ahmed Atia. "Selecting Dynamic Path Planning Algorithm Based-Upon Ranking Approach for Omni-Wheeled Mobile Robot." *Journal of Advanced Research in Applied Sciences and Engineering Technology* 41, no. 2 (2024): 125-138. https://doi.org/10.37934/araset.41.2.125138
[19] Abu Ubaidah Shamsudin, Puteri Alisha Balqis Mohd Sharif, Zubair Adil Soomro, Ruzairi Abdul Rahim, Ahmad Athif Mohd Faudzi, Wan Nurshazwani Wan Zakaria, Mohamad Heerwan Peeie, Carl John Salaan. "Autonomous Navigation Robot using Slam and Path Planning Based on a Single RP-LIDAR". Journal of Advanced Research in Applied Sciences and Engineering Technology 53, no. 2 (2025): 161-169. https://doi.org/10.37934/araset.53.2.161169
[20] Khatib, Oussama. "Real-time obstacle avoidance for manipulators and mobile robots." In *Proceedings. 1985 IEEE international conference on robotics and automation*, vol. 2, pp. 500-505. IEEE, 1985. https://doi.org/10.1109/ROBOT.1985.1087247
[21] Rimon, Elon. *Exact robot navigation using artificial potential functions*. Yale University, 1990.
[22] Connolly, Christopher I., and Roderic A. Grupen. "The applications of harmonic functions to robotics." *Journal of robotic Systems* 10, no. 7 (1993): 931-946. https://doi.org/10.1002/rob.4620100704
[23] Garrido, Santiago, Luis Moreno, Dolores Blanco, and Fernando Martín Monar. "Robotic motion using harmonic functions and finite elements." *Journal of intelligent and Robotic Systems* 59 (2010): 57-73. https://doi.org/10.1007/s10846-009-9381-3
[24] Hu, Changmiao, Lian-Zhi Huo, Zheng Zhang, and Ping Tang. "Multi-temporal landsat data automatic cloud removal using poisson blending." *IEEE Access* 8 (2020): 46151-46161. https://doi.org/10.1109/ACCESS.2020.2979291
[25] Pan, Yang, Mingwu Jin, Shunrong Zhang, and Yue Deng. "TEC map completion using DCGAN and Poisson blending." *Space Weather* 18, no. 5 (2020): e2019SW002390. https://doi.org/10.1029/2019SW002390
[26] Tan, Jeremy, Benjamin Hou, Thomas Day, John Simpson, Daniel Rueckert, and Bernhard Kainz. "Detecting outliers with poisson image interpolation." In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2021: 24th International Conference, Strasbourg, France, September 27–October 1, 2021, Proceedings, Part V 24*, pp. 581-591. Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-87240-3_56

[27] Pérez, Patrick, Michel Gangnet, and Andrew Blake. "Poisson image editing." In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pp. 577-582. 2023. https://doi.org/10.1145/3596711.3596772

[28] Afifi, Mahmoud, and Khaled F. Hussain. "MPB: A modified poisson blending technique." *Computational Visual Media* 1 (2015): 331-341. https://doi.org/10.1007/s41095-015-0027-z

[29] Hashim, S. H. A., F. A. Hamid, J. J. Kiram, and J. Sulaiman. "The relationship investigation between factors affecting demand for broadband and the level of satisfaction among broadband customers in the South East Coast of Sabah, Malaysia." In *Journal of Physics: Conference Series*, vol. 890, no. 1, p. 012149. IOP Publishing, 2017. https://doi.org/10.1088/1742-6596/890/1/012149

[30] Wang, Zhou, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. "Image quality assessment: from error visibility to structural similarity." *IEEE transactions on image processing* 13, no. 4 (2004): 600-612. https://doi.org/10.1109/TIP.2003.819861

[31] Memon, Farida, Mukhtiar Ali Unar, and Sheeraz Memon. "Image quality assessment for performance evaluation of focus measure operators." *Mehran University Research Journal of Engineering & Technology* 34, no. 4 (2015): 379-386.

[32] "Path planning simulator," https://github.com/azalisaudi/planner. https://github.com/azalisaudi/planner