

Computação Paralela

Paralelização em MPI de métodos iterativos para resolução de equações diferenciais a duas dimensões

Alexandre Rodrigues, 92993, Gustavo Morais, 92978

15 de julho de 2022



1 Introdução

O presente trabalho tem como objetivo paralelizar, através de MPI, métodos iterativos para resolução de equações diferenciais a duas dimensões. Para este efeito considera-se a equação de Poisson a duas dimensões:

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

onde $f(x, y) = 7 \sin(3\pi x) \cos(3\pi x) \sin(2\pi y) \cos(3\pi y)$. É possível desenvolver programas paralelizados com MPI que recorrem ao método de Jacobi para encontrar soluções numéricas deste tipo de equação no domínio $-1 \leq x \leq 1$ e $-1 \leq y \leq 1$. Estes programas discretizam o domínio em ambas as direções, x e y , formando uma espécie de grelha de pontos onde a função $V(x, y)$ é calculada iterativamente. Para tal, as segunda derivadas são aproximadas por diferenças finitas em cada ponto da grelha utilizando os valores da função nos seus pontos vizinhos:

$$\begin{aligned} \frac{\partial^2 V(x, y)}{\partial x^2} &\approx \frac{V(x-h, y) - 2V(x, y) + V(x+h, y)}{h^2}, \\ \frac{\partial^2 V(x, y)}{\partial y^2} &\approx \frac{V(x, y-h) - 2V(x, y) + V(x, y+h)}{h^2} \end{aligned}$$

onde h é o espaçamento uniforme entre pontos vizinhos na grelha.

As coordenadas dos pontos da grelha são dados por $x = -1 + jh$ e $y = 1 - ih$ através dos índices $j = 0, 1, \dots, n_x - 1$ e $i = 0, 1, \dots, n_y - 1$ com $n_x = n_y = \frac{2}{h} + 1$ neste caso. A equação que resulta da substituição das segundas derivadas pelas suas aproximações, e das coordenadas x e y pelos índices j e i , respetivamente, é expressa na forma iterativa do método de Jacobi por

$$V_{i,j}^{(k)} = \frac{1}{4} [V_{i-1,j}^{(k-1)} + V_{i+1,j}^{(k-1)} + V_{i,j-1}^{(k-1)} + V_{i,j+1}^{(k-1)} - h^2 f_{i,j}]$$

onde k representa a iteração. A tolerância ϵ é definida à partida, e considera-se que o método convergiu quando

$$\frac{\sqrt{\sum_{i,j} [V_{i,j}^{(k)} - V_{i,j}^{(k-1)}]^2}}{\sqrt{\sum_{i,j} [V_{i,j}^{(k)}]^2}} < \epsilon$$

Para atingir os objetivos do presente trabalho, os programas desenvolvidos devem usar uma decomposição do domínio bidimensional com os subdomínios distribuídos por duas colunas.

2 Resolução dos exercícios

Os resultados apresentados são referentes a uma tolerância de $\epsilon = 10^{-6}$, a um número de processos igual a 4 e $n_x = n_y = 100$. Para verificar os resultados obtidos no programa paralelizado tais como os gráficos e a *mean square error (MSE)* obtida, foi desenvolvido um programa sequencial em MATLAB.

Exercício a)

Nesta alínea foi pedido para adaptar o programa de forma a incorporar as seguintes condições fronteiras:

- $V(-1, y) = \frac{1+y}{4}$

- $V(1, y) = \frac{3+y}{4}$
- $V(x, -1) = \frac{1+x}{4}$
- $V(x, 1) = \frac{3+x}{4}$

O código adicionado nesta alínea está presente no Código 1, onde no primeiro *if* adicionamos as condições fronteira para $x = -L$ e $x = L$ e no segundo para $y = -L$ e $y = L$.

```
// Condições fronteira em x=-L e x=L
if (newid % 2 == 0) {
    for (int i = 1; i < myrows + 1; i++) {
        Vnew[i][0] = (1 + (-L + h*(firstrow+i-1))) / 4;
        Vold[i][0] = Vnew[i][0] ;
    }
} else {
    for (int i = 1; i < myrows + 1; i++) {
        Vnew[i][mycols+1] = (3 + (-L + h*(firstrow+i-1))) / 4;
        Vold[i][mycols+1] = Vnew[i][mycols+1];
    }
}

// Condições fronteira em y=-L e y=L
if (newid == manager_rank || newid == 1) {
    for (int j = 0; j < mycols+2; j++) {
        Vnew[0][j] = (1 + (-L + h*(firstcol+j-1))) / 4;
        Vold[0][j] = Vnew[0][j];
    }
}

if (newid == nprocs - 2 || newid == nprocs - 1) {
    for (int j = 0; j < mycols + 2; j++) {
        Vnew[myrows+1][j] = (3 + (-L + h*(firstcol+j-1))) / 4;
        Vold[myrows+1][j] = Vnew[myrows+1][j];
    }
}
```

Código 1

Os resultados obtidos estão presentes na Figura 1 e na Tabela 1. É de notar o elevado número de iterações e erro (MSE) da ordem de $h^2 = (2.0 \times \frac{L}{n_x-1})^2 = 4.0812 \times 10^{-4}$. Obteve-se o gráfico esperado pois os resultados são extremamente afetados pelas condições fronteira, não permitindo obter resultados semelhantes às seguintes alíneas.

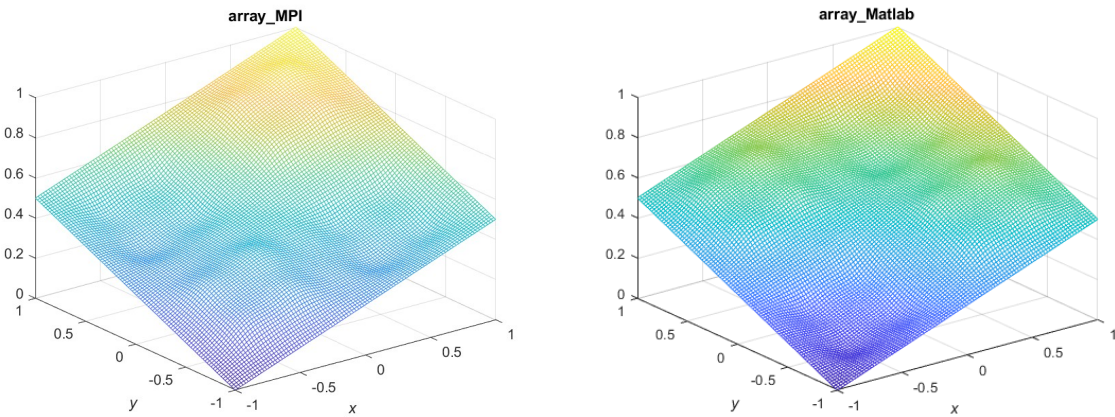


Figura 1: Representações gráficas dos resultados obtidos para a implementação paralela (*array_MPI*) e sequencial (*array_Matlab*) na alínea a).

Implementação	Nº de iterações	T. Cálculo (s)	T. Escrita (s)	MSE
Paralela	11804	0.6181	0.0023	7.8138×10^{-3}
Sequencial	11759	17.1762	N/A	N/A

Tabela 1: Resultados obtidos para a implementação paralela e sequencial na alínea a).

Exercício b)

Na alínea b) foi pedido para adaptar novamente o programa para, desta vez, utilizar condições fronteira periódicas em ambas as direções, x e y . Primeiramente, e de modo a poderem ser implementadas condições fronteira periódicas, a variável *periodic* foi alterada. Foi também necessário alterar o domínio de cálculo de maneira a que se incluía todos os pontos do domínio global, que não são pontos fantasma. Como as condições são periódicas, o passo h foi alterado de $h = 2.0 \times \frac{L}{n_x - 1}$ para $h = 2.0 \times \frac{L}{n_x}$ porque existe uma interação adicional.

Como agora estamos a considerar todos os pontos do domínio global, foi necessário criar um novo *datatype* que define uma nova *view* para a escrita do ficheiro. As alterações feitas ao código estão presentes em Código 2.

```
int periodic[2] = {1,1};

(...)

// Alterado para incluir as linhas de fronteira
for (int i = 0; i < nprocs_col; i++)
{
    listfirstrow[2*i] = i * (nrows);
    listmyrows[2*i] = nrows+1;
    listfirstrow[2*i+1] = i * (nrows);
    listmyrows[2*i+1] = nrows+1;
}
// Altera o numero de linhas do penultimo e do ultimo
listmyrows[nprocs-2] = ny - (nprocs_col - 1) * nrows;
listmyrows[nprocs-1] = ny - (nprocs_col - 1) * nrows;

// Colunas
// Alterado para incluir as colunas de fronteira
int ncols_temp = (int)((nx-2)/2);
for (int i = 0; i < nprocs_col; i++)
{
    listfirstcol[2*i] = 0;
    listmycols[2*i] = ncols_temp + 1;
    listfirstcol[2*i+1] = ncols_temp + 1;
    listmycols[2*i+1] = nx - 1 - ncols_temp;
}

(...)
int gsizes[2] = {ny, nx};
int lsizes[2] = {myrows, mycols};
int start_ind[2] = {firstrow, firstcol};
```

Código 2

Os resultados obtidos estão presentes na Figura 2 e na Tabela 2. O número de iterações reduziu consideravelmente tal como o tempo de cálculo. O tempo de escrita dos resultados manteve-se pois o número de processos não foi alterado. O erro (MSE) é menor mas pode-se ainda considerar da ordem de $h^2 = (2.0 \times \frac{L}{n_x})^2 = 4.0000 \times 10^{-4}$. Pode-se também entender que estes resultados já são mais realistas dado que não estão limitados às condições fronteiras.

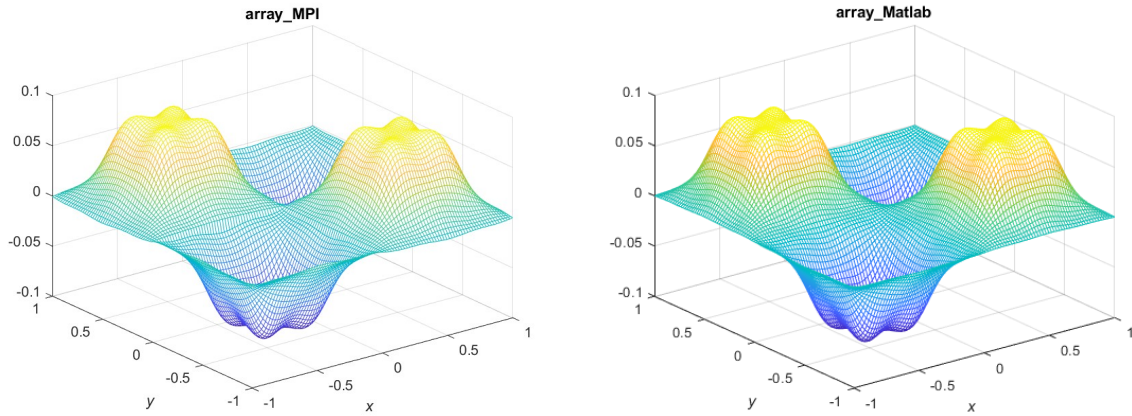


Figura 2: Representações gráficas dos resultados obtidos para a implementação paralela (*array_MPI*) e sequencial (*array_Matlab*) na alínea b).

Implementação	Nº de iterações	T. Cálculo (s)	T. Escrita (s)	MSE
Paralela	3877	0.2242	0.0023	1.9857×10^{-5}
Sequencial	3840	7.1621	N/A	N/A

Tabela 2: Resultados obtidos para a implementação paralela e sequencial na alínea b).

Exercício c)

Na alínea anterior, a aproximação por diferenças finitas utilizada para as segunda derivadas corresponde a um estêncil de 5 pontos que introduz um erro local de ordem de h^2 . Na presente alínea foi pedido para introduzir um estêncil de 9 pontos de forma a diminuir o erro mencionado para h^4 .

Como nesta alínea existem 4 novos pontos que vão ser utilizados para o cálculo, é necessário alocar espaço para mais 4 linhas fantasma. Como estas não fazem parte do domínio utilizado para o cálculo, é necessário adequar os índices dos parâmetros passados à função *myf* e também na implementação do método de Jacobi. De forma semelhante à alínea anterior, foi necessário alterar a escrita do ficheiro tendo em conta a introdução de mais pontos.

Nesta alínea vão ser criadas mais comunicações em comparação com a anterior. Primeiramente, na direção vertical é enviada a informação da 1ª linha para a penúltima linha fantasma e a última linha do domínio para a primeira linha fantasma. Para os pontos restantes, é enviada a informação da 2ª linha para a última linha fantasma e a penúltima linha do domínio para a segunda linha fantasma. O mesmo raciocínio é utilizado para as colunas na direção horizontal.

De forma a incluir todos os pontos do domínio global, foi feito um ajuste nos índices utilizados para o valor de *Vold*.

```
// ALterado para usar mais 2 colunas e linhas fantasma, necessário para o maior estêncil
double (*Vold)[mycols+4], (*Vnew)[mycols+4], (*myf)[mycols+4];
Vold = calloc(myrows + 4, sizeof(*Vold));
Vnew = calloc(myrows + 4, sizeof(*Vnew));
myf = calloc(myrows + 4, sizeof(*myf));

(...)

// dominio principal agora nao inclui as 2 primeiras e 2 ultimas linhas e colunas
for (int j = 2; j < mycols + 2; j++)
{
    for (int i = 2; i < myrows + 2; i++)
```

```

    {
        myf[i][j] = f(-L + (firstcol + j - 2) * h, -L + (firstrow + i - 2) * h);
    }
}

(...)

// Alterado para + 4
MPI_Datatype column;
MPI_Type_vector(myrows + 4, 1, mycols + 4, MPI_DOUBLE, &column);
MPI_Type_commit(&column);

(...)

// Dominio principal agora nao inclui as 2 primeiras e 2 ultimas linhas e colunas
// Alterado para usar o novo estencil e nova equação iterativa
for (int j = 2; j < mycols + 2 ; j++)
{
    for (int i = 2; i < myrows + 2; i++)
    {
        Vnew[i][j] = (W/60)*(16*Vold[i-1][j] + 16*Vold[i+1][j] + 16*Vold[i][j-1]+ 16*Vold[i][j+1]
        - Vold[i-2][j] - Vold[i+2][j] - Vold[i][j-2] - Vold[i][j+2] -12*h*h*myf[i][j]) + (1-W)*Vold[i][j];
        sums[0] += (Vnew[i][j] - Vold[i][j]) * (Vnew[i][j] - Vold[i][j]);
        sums[1] += Vnew[i][j] * Vnew[i][j];
    }
}

(...)

// Alterado para + 4
int memsizes[2] = {myrows+4, mycols+4};

(...)

// comunicações sentido ascendente
// Alterado para +4
MPI_Sendrecv(Vnew[myrows], mycols+4, MPI_DOUBLE, nbrtop, 0,
    Vnew[0] , mycols+4, MPI_DOUBLE, nbrbottom, 0, comm2D, MPI_STATUS_IGNORE);

// comunicações sentido descendente
// Alterado para usar 3a e antepenultima coluna (e +4)
MPI_Sendrecv(Vnew[2], mycols+4, MPI_DOUBLE, nbrbottom, 1,
    Vnew[myrows+2] , mycols+4, MPI_DOUBLE, nbrtop, 1, comm2D, MPI_STATUS_IGNORE);

// comunicações sentido para direita
MPI_Sendrecv(&(Vnew[0][mycols]), 1, column, nbrright, 2,
    &(Vnew[0][0]), 1, column, nbrleft, 2, comm2D, MPI_STATUS_IGNORE);

// comunicações sentido para esquerda
// Alterado para usar 3a e antepenultima coluna
MPI_Sendrecv(&(Vnew[0][2]), 1, column, nbrleft, 3,
    &(Vnew[0][mycols+2]), 1, column, nbrright, 3, comm2D, MPI_STATUS_IGNORE);

// Outras comunicações
// Comunicar 2a e penultima linhas
MPI_Sendrecv(Vnew[myrows+1], mycols+4, MPI_DOUBLE, nbrtop, 6,
    Vnew[1], mycols+4, MPI_DOUBLE, nbrbottom, 6, comm2D, MPI_STATUS_IGNORE);

// Comunicar 4a linha
MPI_Sendrecv(Vnew[3], mycols+4, MPI_DOUBLE, nbrbottom, 5,

```

```

Vnew[myrows+3], mycols+4, MPI_DOUBLE, nbrtop, 5, comm2D, MPI_STATUS_IGNORE);

// Comunicar 4a coluna
MPI_Sendrecv(&(Vnew[0][3]), 1, column, nbrleft, 7,
             &(Vnew[0][mycols+3]), 1, column, nbrright, 7, comm2D, MPI_STATUS_IGNORE);

// Comunicar 2a e penultima colunas
MPI_Sendrecv(&(Vnew[0][mycols+1]), 1, column, nbrright, 8,
             &(Vnew[0][1]), 1, column, nbrleft, 8, comm2D, MPI_STATUS_IGNORE);

// Alterado para +4
for (int i = 0; i < myrows + 4; i++)
{
    for (int j = 0; j < mycols + 4; j++)
    {
        Vold[i][j] = Vnew[i][j];
    }
}

```

Código 3

Os resultados obtidos estão presentes na Figura 3 e na Tabela 3. O número de iterações aumentou possivelmente devido à maior precisão dado o estêncil maior, consequentemente o tempo de cálculo também aumentou. O tempo de escrita dos resultados manteve-se pois o número de processos não foi alterado. O erro (MSE) é maior, não correspondendo ao esperado da ordem de $h^4 = (2.0 \times \frac{L}{n_x})^4 = 1.6000 \times 10^{-7}$.

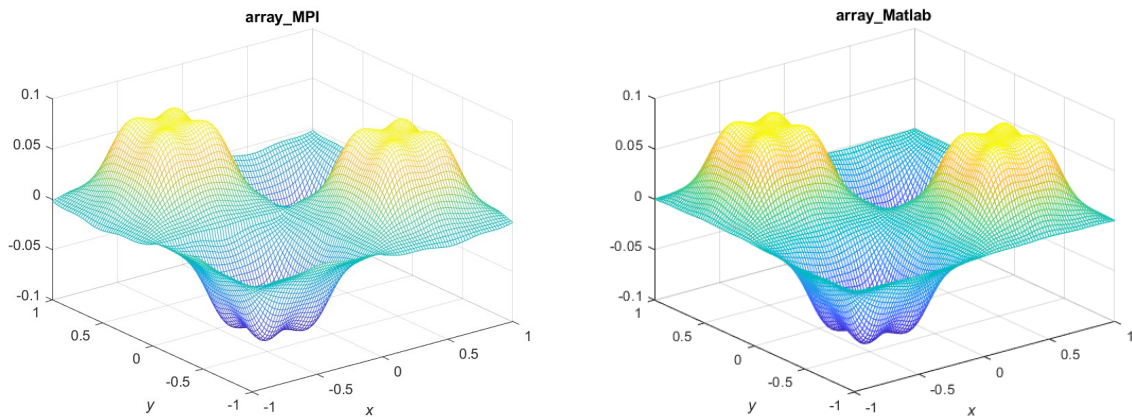


Figura 3: Representações gráficas dos resultados obtidos para a implementação paralela (*array_MPI*) e sequencial (*array_Matlab*) na alínea c).

Implementação	Nº de iterações	T. Cálculo (s)	T. Escrita (s)	MSE
Paralela	4972	0.3987	0.0023	7.2776×10^{-5}
Sequencial	4925	12.2727	N/A	N/A

Tabela 3: Resultados obtidos para a implementação paralela e sequencial na alínea c).

Exercício d)

Através da implementação da alínea b), é pedido que se utilize o método de Gauss-Seidel em alternativa ao método de Jacobi pois o método de Gauss-Seidel converge mais rapidamente para a solução. Para implementação vai ser utilizado um sistema do tipo par-ímpar visto que para calcular o ponto da iteração atual são utilizados os pontos da mesma iteração.

As condições fronteira mantêm-se periódicas e são consideradas de novo apenas mais duas linhas e colunas fantasma, de forma semelhante à alínea b). Primeiramente, calcula-se, no domínio local, e envia-se os pontos pares procedendo-se ao cálculo dos pontos ímpares. No fim, a informação é enviada aos vizinhos de maneira semelhante ao envio da alínea b).

```
// Calculos pares (i+j é par)
for (int i = 1; i < myrows + 1; i++)
{
    for (int j = 1; j < mycols + 1 ; j++)
    {
        if (((firstcol + j - 1) + (firstrow + i - 1)) % 2 == 0) // verifica que é par
        {
            Vnew[i][j] = (Vnew[i+1][j] + Vnew[i-1][j] + Vnew[i][j+1] + Vnew[i][j-1] - h*h*myf[i][j]) / 4.0;
            sums[0] += (Vnew[i][j] - Vold[i][j]) * (Vnew[i][j] - Vold[i][j]);
            sums[1] += Vnew[i][j] * Vnew[i][j];
        }
    }
}

// Comunicar aos vizinhos (pares para ímpares)
MPI_Sendrecv(&Vnew[1][1], mycols, MPI_DOUBLE, nbrbottom, 4,
             &Vnew[myrows+1][1], mycols, MPI_DOUBLE, nbrtop, 4, comm2D, MPI_STATUS_IGNORE);

MPI_Sendrecv(&Vnew[myrows][1], mycols, MPI_DOUBLE, nbrtop, 5,
             &Vnew[0][1], mycols, MPI_DOUBLE, nbrbottom, 5, comm2D, MPI_STATUS_IGNORE);

MPI_Sendrecv(&Vnew[1][1], 1, column, nbrleft, 6,
             &Vnew[1][mycols+1], 1, column, nbrright, 6, comm2D, MPI_STATUS_IGNORE);

MPI_Sendrecv(&Vnew[1][mycols], 1, column, nbrright, 7,
             &Vnew[1][0], 1, column, nbrleft, 7, comm2D, MPI_STATUS_IGNORE);

// Calcular ímpares (i+j é ímpar)
for (int i = 1; i < myrows + 1; i++)
{
    for (int j = 1; j < mycols + 1 ; j++)
    {
        if (((firstcol + j - 1) + (firstrow + i - 1)) % 2 == 1) // verifica que é ímpar
        {
            Vnew[i][j] = (Vnew[i-1][j] + Vnew[i][j-1] + Vnew[i][j+1] + Vnew[i+1][j] - h*h*myf[i][j]) / 4.0;
            sums[0] += (Vnew[i][j] - Vold[i][j]) * (Vnew[i][j] - Vold[i][j]);
            sums[1] += Vnew[i][j] * Vnew[i][j];
        }
    }
}

// Comunicar aos vizinhos (ímpares para pares)
MPI_Sendrecv(&Vnew[1][1], mycols, MPI_DOUBLE, nbrbottom, 8,
             &Vnew[myrows+1][1], mycols, MPI_DOUBLE, nbrtop, 8, comm2D, MPI_STATUS_IGNORE);

MPI_Sendrecv(&Vnew[myrows][1], mycols, MPI_DOUBLE, nbrtop, 9,
             &Vnew[0][1], mycols, MPI_DOUBLE, nbrbottom, 9, comm2D, MPI_STATUS_IGNORE);

MPI_Sendrecv(&Vnew[1][1], 1, column, nbrleft, 10,
             &Vnew[1][mycols+1], 1, column, nbrright, 10, comm2D, MPI_STATUS_IGNORE);

MPI_Sendrecv(&Vnew[1][mycols], 1, column, nbrright, 11,
             &Vnew[1][0], 1, column, nbrleft, 11, comm2D, MPI_STATUS_IGNORE);
```

Código 4

Os resultados obtidos estão presentes na Figura 4 e na Tabela 4. O número de iterações reduziu significativamente devido ao uso dos valores mais recentes em cada iteração, consequentemente o tempo de calculo também diminuiu. O tempo de escrita dos resultados manteve-se pois o número de processos não foi alterado. O erro (MSE) é menor, e corresponde aproximadamente ao esperado da ordem de $h^2 = (2.0 \times \frac{L}{n_x})^2 = 4.0000 \times 10^{-4}$.

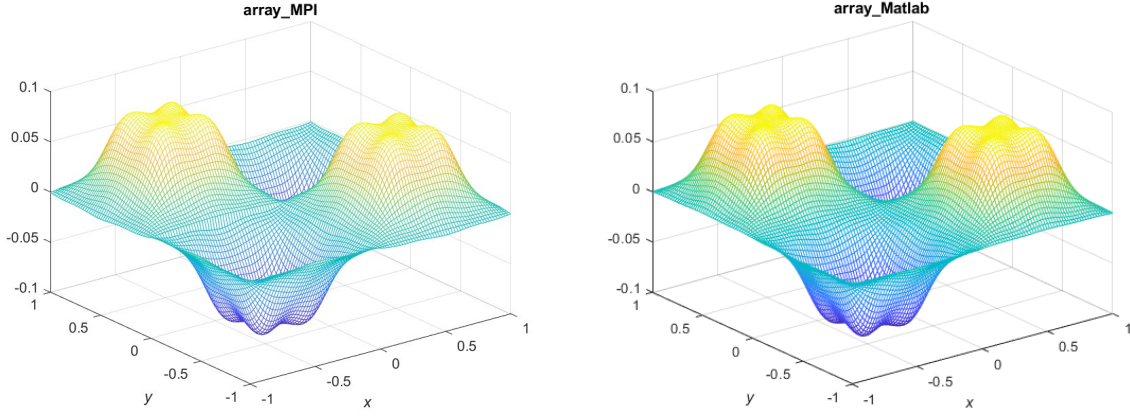


Figura 4: Representações gráficas dos resultados obtidos para a implementação paralela (*array_MPI*) e sequencial (*array_Matlab*) na alínea d).

Implementação	Nº de iterações	T. Cálculo (s)	T. Escrita (s)	MSE
Paralela	2118	0.1811	0.0023	2.2775×10^{-5}
Sequencial	2101	4.9398	N/A	N/A

Tabela 4: Resultados obtidos para a implementação paralela e sequencial na alínea d).

3 Conclusão

A clara diferença entre a primeira alínea e as restantes demonstra o efeito do uso de condições fronteira, que neste caso, reduzem muito os resultados do método e consequentemente o efeito representado na figura 1. A ordem de grandeza dos erros (MSE) foi de acordo com o esperado em todas as alíneas exceto a c), algo que pode ser explicado pelo erro base do método e do erro que advém do hardware usado e a sua representação dos valores. As representações dos resultados das figuras 2, 3 e 4, são bastante semelhantes, as maiores diferenças em relação aos resultados da implementação sequencial em Matlab são na zona da divisão de processos (linhas $x = 0$ e $y = 0$).