# Projeto 2 - Paralelização em MPI de métodos iterativos para resolução de equações diferenciais a duas dimensões

Computação Paralela 2020-2021

Ana Carlota Cação Moreira, nº88973

Departamento de Física
Universidade de Aveiro
10-07-2021

# Índice

**Exercício a)** **1**

**Exercício b)** **2**

**Exercício c)** **3**

**Exercício d)** **5**

## Lista de figuras


1 Resultados obtidos no exercício a) para $n_x = n_y = 100$. . . . . 1
2 Resultados obtidos no exercício b) para $n_x = n_y = 100$. . . . . 3
3 Resultados obtidos no exercício c) para $n_x = n_y = 100$. . . . . 5
4 Resultados obtidos no exercício d) para $n_x = n_y = 100$. . . . . 7


## Lista de tabelas


1 Resultados obtidos no exercício a). . . . . 2
2 Resultados obtidos no exercício b. . . . . 3
3 Resultados obtidos no exercício c). . . . . 5
4 Resultados obtidos no exercício d). . . . . 7

# Exercício a)

Este exercício tem como objetivo utilizar o método de Jacobi e alterar as condições fronteira dadas no script fornecido. Como é possível ver pelo código abaixo, no primeiro ciclo *if* define-se a condição fronteira em $x = -L$ e $x = L$ e no segundo em $y = L$ e $y = -L$.

```
// Condicoes fronteira em x=-L e x=L:
    if (newid % 2 == 0) {
        for (i=0; i<myrows+2; i++) {
            myVold[i][0] = (1+(L - h*(mytoprow+i-1)))/4;
            myVnew[i][0] = (1+(L - h*(mytoprow+i-1)))/4;
        }
    } else {
        for (i=0; i<myrows+2; i++) {
            myVold[i][mycols+1] = (3+(L - h*(mytoprow+i-1)))/4;
            myVnew[i][mycols+1] = (3+(L - h*(mytoprow+i-1)))/4;
        }
    }

    // Condicoes fronteira em y=L e y=-L:
    if (newid == 0 || newid == 1 ) {
        for (j=0; j<mycols+2; j++) {
            myVold[0][j] = (3+(-L + h*(myleftcol+j-1)))/4;
            myVnew[0][j] = (3+(-L + h*(myleftcol+j-1)))/4;
        }
    }
    if (newid == numprocs-2 || newid == numprocs-1 ) {
        for (j=0; j<mycols+2; j++) {
            myVold[myrows+1][j] = (1+(-L + h*(myleftcol+j-1)))/4;
            myVnew[myrows+1][j] = (1+(-L + h*(myleftcol+j-1)))/4;
        }
    }
```

Os resultados obtidos encontram-se apresentados na figura 1 e na tabela 1.

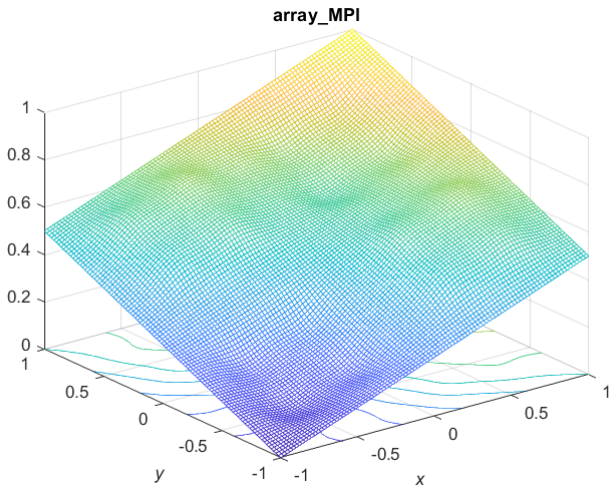


Figura 1: Resultados obtidos no exercício a) para $n_x = n_y = 100$.

| Nº de iterações | Tempo de cálculo (s) | Tempo de escrita (s) | Nº de processos | $n_x = n_y$ |
|---|---|---|---|---|
| 11804 | 0.580153 | 0.002339 | 4 | 100 |

Tabela 1: Resultados obtidos no exercício a).

# Exercício b)

Neste exercício pretende-se resolver o mesmo problema que no anterior mas considerando condições fronteira periódicas.

Em primeiro lugar foi alterada a variável periodic de modo a poderem ser implementadas condições fronteira periódicas de uma forma mais simplificada.

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

Em relação ao exercício a) foi também alterada a definição do domínio de calculo. Como estamos a considerar condições fronteiras periódicas temos que incluir todos os pontos do domínio global que não são pontos fantasma no domínio utilizado para efetuar os cálculos. Foi também alterado o passo espacial h: $h = 2.0 * L/(nx)$.

```
int numrows = (int)(((double)2*(nx-2)/numprocs) + 0.5);
int remaining = nx;
int listtoprows[numprocs], listnumrows[numprocs];
for (i=0; i<numprocs-2; i+=2) {
   listnumrows[i] = numrows + 1;
   listnumrows[i+1] = numrows + 1;
   listtoprows[i] = nx - remaining;
   listtoprows[i+1] = nx - remaining;
   remaining += -numrows;
   }

listnumrows[numprocs-2] = remaining-1;
listnumrows[numprocs-1] = remaining-1;
listtoprows[numprocs-2] = nx - remaining + 1;
listtoprows[numprocs-1] = nx - remaining + 1;

int listleftcols[numprocs], listnumcols[numprocs];
for (i=0; i<numprocs; i+=2) {
   listleftcols[i] = 0;
   listnumcols[i] = (ny-2)/2 + 1;
   listleftcols[i+1] = (ny-2)/2 + 1;
   listnumcols[i+1] = ny - listnumcols[i];
   }
```

Na escrita do ficheiro também foi necessário modificar o código. Como agora estamos a considerar todos os pontos do domínio global (à exceção dos pontos fantasma) foi necessário criar um datatype que define a nova view para a escrita do ficheiro. Deste modo, cada processo fica encarregue de escrever uma parte da matriz global no ficheiro de saída.

```
int globalsize[] = {nx, ny};
int localsize[] = {myrows, mycols};
int start_inds[] = {mytoprow, myleftcol};
```

Quanto às comunicações entre os processos, na direção "vertical", a primeira linha do domínio

é enviada para a última linha fantasma e a última é enviada para a primeira. O mesmo raciocínio acontece na direção "horizontal": a primeira coluna do domínio (à esquerda) é enviada para linha fantasma e a última (à direita) é enviada para a primeira .

```
MPI_Sendrecv(myVnew[1], mycols+2, MPI_DOUBLE, nbrtop, 0, myVnew[myrows+1], mycols+2,
    MPI_DOUBLE, nbrbottom, 0, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(myVnew[myrows], mycols+2, MPI_DOUBLE, nbrbottom, 1, myVnew[0], mycols+2,
    MPI_DOUBLE, nbrtop, 1, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(myVnew[0][1]), 1, column, nbrleft, 2, &(myVnew[0][mycols+1]), 1,
    column, nbrright, 2, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(myVnew[0][mycols]), 1, column, nbrright, 3, &(myVnew[0][0]), 1,
    column, nbrleft, 3, comm2d, MPI_STATUS_IGNORE);
```

Os resultados obtidos encontram-se apresentados na figura 2 e na tabela 2.

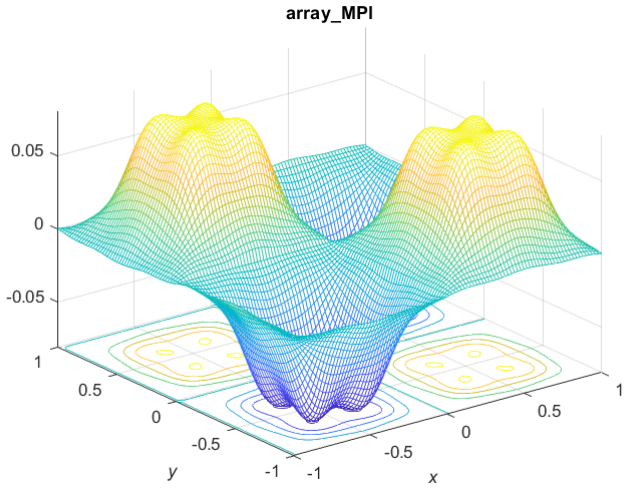


Figura 2: Resultados obtidos no exercício b) para $n_x = n_y = 100$.

| Nº de iterações | Tempo de cálculo (s) | Tempo de escrita (s) | Nº de processos | $n_x = n_y$ |
|---|---|---|---|---|
| 3839 | 0.0215815 | 0.002406 | 4 | 100 |

Tabela 2: Resultados obtidos no exercício b.

# Exercício c)

Este exercício tem como objetivo resolver o mesmo problema dos anteriores continuado a considerar condições fronteira periódicas mas usando um estêncil de 9 pontos.

Devido a existirem mais 4 pontos a serem utilizados para o cálculo, vão ser necessárias 4 pontos fantasma e não 2 como nos exercício anteriores e por isso é necessário alocar espaço para mais 4 linhas fantasma em vez de duas.

```
// Alocar matrizes na memoria
double (*myVold)[mycols+4], (*myVnew)[mycols+4], (*myf)[mycols+4];
myVold = calloc(myrows+4, sizeof(*myVold));
myVnew = calloc(myrows+4, sizeof(*myVnew));
myf = calloc(myrows+4, sizeof(*myf));
```

```
MPI_Type_vector(myrows+4, 1, mycols+4, MPI_DOUBLE, &column);
```

Como as linhas fantasma não fazem parte do domínio utilizado para o cálculo, é necessário ajustar os índices no cálculo da função myf e na implementação do método de Jacobi como se pode ver no código seguinte.

```
// Calcular myf:
for (i=2; i<myrows+2; i++){
   for (j=2; j<mycols+2; j++){
      myf[i][j] = f(-L + h*(myleftcol+j-2), L - h*(mytoprow+i-2));
   }
}
```

```
double sums[2];
sums[0] = 0.0;
sums[1] = 0.0;
for (i=2; i<myrows+2; i++) {
   for (j=2; j<mycols+2; j++) {
      myVnew[i][j] = (w/60)*(16*myVold[i-1][j] + 16*myVold[i+1][j] + 16*myVold[i][j-1]
          + 16*myVold[i][j+1] - myVold[i-2][j] - myVold[i+2][j] - myVold[i][j-2] -
          myVold[i][j+2] -12*h*h*myf[i][j]) + (1-w)*myVold[i][j];
      sums[0] += (myVnew[i][j]-myVold[i][j])*(myVnew[i][j]-myVold[i][j]);
      sums[1] += myVnew[i][j]*myVnew[i][j];
   }
}
```

Também na escrita do ficheiro é necessário ter em atenção os pontos que vão ser enviados e fazer os ajustes adequados.

```
int matrixsize[] = {myrows+4,mycols+4};
start_inds[0] = 1;
start_inds[1] = newid % 2;
if (newid == 0 || newid == 1) {
   start_inds[0]--;
}
```

Neste caso vamos ter que criar mais comunicações. Em primeiro lugar, na direção "vertical", é enviada a informação da 1ª linha para a penúltima linha fantasma e a última linha do domínio de cálculo para a primeira linha fantasma. O mesmo raciocínio acontece para as colunas na direção "horizontal". Depois é feito o mesmo mas para os pontos que faltam. Na direção "vertical", é enviada a informação da 2ª linha para a última linha fantasma e a penúltima linha do domínio de cálculo para a segunda linha fantasma. O mesmo raciocínio acontece para as colunas na direção "horizontal".

```
MPI_Sendrecv(myVnew[2], mycols+4, MPI_DOUBLE, nbrtop, 0, myVnew[myrows+2], mycols+4,
    MPI_DOUBLE, nbrbottom, 0, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(myVnew[myrows], mycols+4, MPI_DOUBLE, nbrbottom, 1, myVnew[0], mycols+4,
    MPI_DOUBLE, nbrtop, 1, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(myVnew[0][2]), 1, column, nbrleft, 2, &(myVnew[0][mycols+2]), 1,
    column, nbrright, 2, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(myVnew[0][mycols]), 1, column, nbrright, 3, &(myVnew[0][0]), 1,
    column, nbrleft, 3, comm2d, MPI_STATUS_IGNORE);

MPI_Sendrecv(myVnew[3], mycols+4, MPI_DOUBLE, nbrtop, 5, myVnew[myrows+3], mycols+4,
    MPI_DOUBLE, nbrbottom, 5, comm2d, MPI_STATUS_IGNORE);
```

```
MPI_Sendrecv(myVnew[myrows+1], mycols+4, MPI_DOUBLE, nbrbottom, 6, myVnew[1],
    mycols+4, MPI_DOUBLE, nbrtop, 6, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(myVnew[0][3]), 1, column, nbrleft, 7, &(myVnew[0][mycols+3]), 1,
    column, nbrright, 7, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(myVnew[0][mycols+1]), 1, column, nbrright, 8, &(myVnew[0][1]), 1,
    column, nbrleft, 8, comm2d, MPI_STATUS_IGNORE);
```

Por fim, é feito também um ajuste nos índices utilizados para atualizar o valor de Vold, de modo a incluir todos os pontos do domínio global.

```
// Vold = Vnew
for (i=0; i<myrows+4; i++) {
   for (j=0; j<mycols+4; j++) {
      myVold[i][j] = myVnew[i][j];
   }
}
```

Os resultados obtidos encontram-se apresentados na figura 3 e na tabela 3. É de notar que em comparação com o resultado da alínea b), este método utiliza mais iterações para convergir e encontrar a solução.

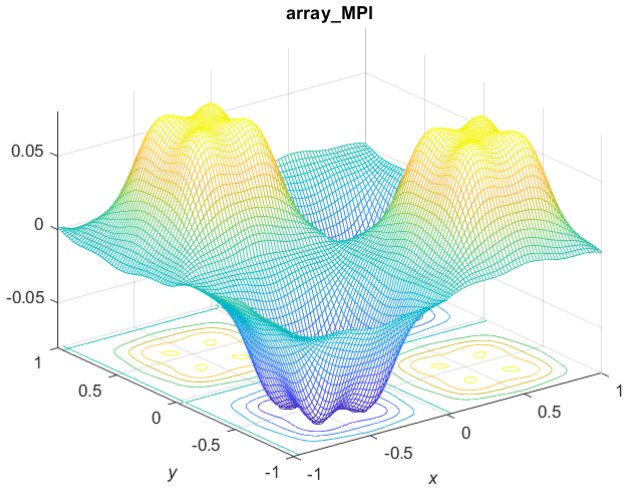


Figura 3: Resultados obtidos no exercício c) para $n_x = n_y = 100$.

| Nº de iterações | Tempo de cálculo (s) | Tempo de escrita (s) | Nº de processos | $n_x = n_y$ |
|---|---|---|---|---|
| 4924 | 0.387413 | 0.002258 | 4 | 100 |

Tabela 3: Resultados obtidos no exercício c).

# Exercício d)

Neste exercício é pedido que se resolva o mesmo problema das alíneas anteriores, mas usando o método de Gauss-Seidel em alternativa ao método de Jacobi. Para isso, vai ser utilizado um sistema de par/ímpar ou vermelho/preto, pois para calcular o ponto da iteração atual são usados pontos da mesma iteração.

Voltamos a considerar de novo apenas mais duas linhas e colunas fantasma e continuamos a

ter condições fronteira periódicas, como na alínea b). Dentro do domínio local em primeiro lugar são calculados os pontos pares, depois a informação é enviada para os vizinhos e são calculados os pontos ímpares. No final a informação é enviada aos vizinhos da mesma maneira que foi enviada na alínea b).

```
// Calcular pares
for (i=1; i<myrows+1; i++) {
    for (j=1; j<mycols+1; j+=2) {
            myVnew[i][j] = 0.25 * (myVnew[i-1][j] + myVnew[i][j-1] + myVnew[i][j+1] +
                myVnew[i+1][j] - h * h * myf[i][j]);
            sums[0] += (myVnew[i][j]-myVold[i][j])*(myVnew[i][j]-myVold[i][j]);
            sums[1] += myVnew[i][j]*myVnew[i][j];
    }
}

// Comunicar aos vizinhos
MPI_Sendrecv(&myVnew[1][1], mycols, MPI_DOUBLE, nbrtop, 0, &myVnew[myrows+1][1],
    mycols, MPI_DOUBLE, nbrbottom, 0, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&myVnew[myrows][1], mycols, MPI_DOUBLE, nbrbottom, 1, &myVnew[0][1],
    mycols, MPI_DOUBLE, nbrtop, 1, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&myVnew[1][1], 1, column, nbrleft, 2, &myVnew[1][mycols+1], 1, column,
    nbrright, 2, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&myVnew[1][mycols], 1, column, nbrright, 3, &myVnew[1][0], 1, column,
    nbrleft, 3, comm2d, MPI_STATUS_IGNORE);

// Calcular impares
for (i=1; i<myrows+1; i++) {
    for (j=2; j<mycols+1; j+=2) {
            myVnew[i][j] = 0.25 * (myVnew[i-1][j] + myVnew[i][j-1] + myVnew[i][j+1] +
                myVnew[i+1][j] - h * h * myf[i][j]);
            sums[0] += (myVnew[i][j]-myVold[i][j])*(myVnew[i][j]-myVold[i][j]);
            sums[1] += myVnew[i][j]*myVnew[i][j];
    }
}

// Comunicar aos vizinhos
MPI_Sendrecv(&myVnew[1][1], mycols, MPI_DOUBLE, nbrtop, 0, &myVnew[myrows+1][1],
    mycols, MPI_DOUBLE, nbrbottom, 0, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&myVnew[myrows][1], mycols, MPI_DOUBLE, nbrbottom, 1, &myVnew[0][1],
    mycols, MPI_DOUBLE, nbrtop, 1, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&myVnew[1][1], 1, column, nbrleft, 2, &myVnew[1][mycols+1], 1, column,
    nbrright, 2, comm2d, MPI_STATUS_IGNORE);
MPI_Sendrecv(&myVnew[1][mycols], 1, column, nbrright, 3, &myVnew[1][0], 1, column,
    nbrleft, 3, comm2d, MPI_STATUS_IGNORE);
```

Os resultados obtidos encontram-se apresentados na figura 4 e na tabela 4. É de notar que, em comparação com o resultado da alínea b) são utilizadas muito menos iterações, o que nos leva a concluir que o método de Gauss-Seidel converge mais rápido que o método de Jacobi, como era de esperar.

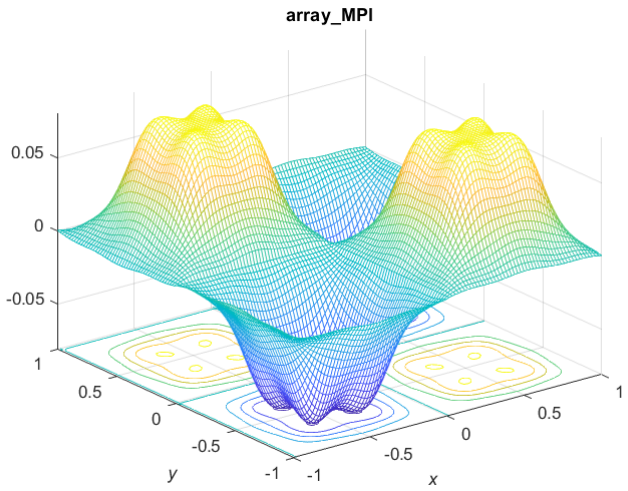


Figura 4: Resultados obtidos no exercício d) para $n_x = n_y = 100$.

| Nº de iterações | Tempo de cálculo (s) | Tempo de escrita (s) | Nº de processos | $n_x = n_y$ |
|---|---|---|---|---|
| 2098 | 0.146734 | 0.002337 | 4 | 100 |

Tabela 4: Resultados obtidos no exercício d).