Universidade de Aveiro
Mestrado em Engenharia Computacional
Computação Paralela
40779

MPI

PROJETO 2: PARALELIZAÇÃO EM MPI DE MÉTODOS ITERATIVOS PARA RESOLUÇÃO DE EQUAÇÕES DIFERENCIAIS A DUAS DIMENSÕES

Vasco Costa - 97746

23 de junho de 2023

MPI vem do inglês *Message-Passing Interface* este foi desenvolvido para colmatar e unificar as diferentes interfaces que estavam a surgir nos anos de 1990 de computação paralela [3, 4]. Dado que a escrita de ficheiros é também indispensável uma realidade que se viu presente foi a de introduzir à interface uma forma de esta comunicar, *I/O* [2]. Uma das vantagens desta interface é a possibilidade de definir no momento de execução quantas unidades existem para processar, sendo assim mais fácil de adaptar a qualquer sistema que esteja presente[1, 4].

Com este projeto pretendem-se responder às questões propostas no enunciado para modificar o código desenvolvido nas aulas. Este foi desenvolvido para resolver a equação de Poisson a duas dimensões, equação (1), no domínio de $-1 \leq x \leq 1$ e $-1 \leq y \leq 1$. Em que este é dividido para que processadores diferentes analisem apenas uma porção mais pequena. Para resolver esta equação diferencial é feita a aproximação com diferenças finitas, $\frac{\partial^2 V(x,y)}{\partial x^2} \approx \frac{V(x-h,y)-2V(x,y)+V(x+h,y)}{h^2}$ e $\frac{\partial^2 V(x,y)}{\partial y^2} \approx \frac{V(x,y-h)-2V(x,y)+V(x,y+h)}{h^2}$.

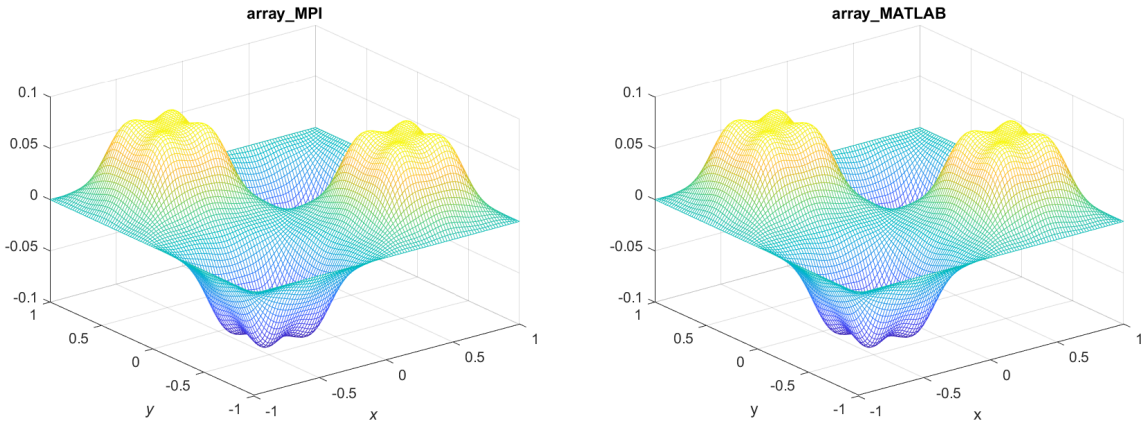
$$\begin{aligned} \frac{\partial^2 V(x,y)}{\partial x^2} + \frac{\partial^2 V(x,y)}{\partial y^2} &= f(x,y) \\ f(x,y) &= 7\sin(2\pi x)\cos(3\pi x)\sin(2\pi y)\cos(3\pi y) \end{aligned} \tag{1}$$

Utilizando assim o código desenvolvido nas aulas com condições fronteira a 0 e a função f de (1), `original.c`, e comparando com o método sequencial desenvolvido em *Matlab*, `original.m`, obteve-se a Figura 1 onde se podem visualizar as diferentes superfícies obtidas que seguem a mesma forma. Na Tabela 1 podem-se também visualizar o número de iterações necessárias para atingir a convergência assim como os tempos de execução. Incluindo-se algumas medidas de diferença entre as superfícies como MSE, $\frac{\sum_i \sum_j (Matriz_{Matlab} - Matriz_{MPI})^2}{n_x \times n_y}$, e a máxima diferença absoluta, MDF, $max(|Matriz_{Matlab} - Matriz_{MPI}|)$. Como se pode confirmar as superfícies obtidas são similares existindo apenas ligeiras diferenças como se verifica através da análise numérica.

Tanto neste caso como nos que seguintemente se apresentam foram utilizados 4 processadores, `mpiexec -n 4 ./file_name`, com $n_x = n_y = 100$ e uma tolerância de $\epsilon = 1e-6$. A compilação foi feita através da *Makefile* também presente na pasta entregue e com *run.sh* a demonstrar como este foi executado. Na pasta *matlab* encontram-se os códigos sequenciais para cada uma das alíneas em que também se efetua a leitura dos resultados `.bin` assim como a sua representação e a escrita em `.png`. A obtenção dos tempos foi realizada numa máquina virtual com a distribuição de Ubuntu 22.04.2 LTS com 6,6Gib de memória num processador AMD Ryzen 7 3700x 8-core com 6 processadores alocados.

Tabela 1 – Número de iterações e tempos de execução para a versão MPI com o erro entre este e o método sequencial em Matlab para o código desenvolvido nas aulas.

Número de iterações	Tempo de cálculo(s)	Tempo de escrita(s)	MSE	MDF
3772	0.130274	0.002206	0.0078062	0.1623



(a) MPI (b) Sequencial

Figura 1 – Comparativo das superfícies obtidas pelo método paralelizado em MPI e sequencial em *Matlab* para o código desenvolvido nas aulas.

Alínea **A)**

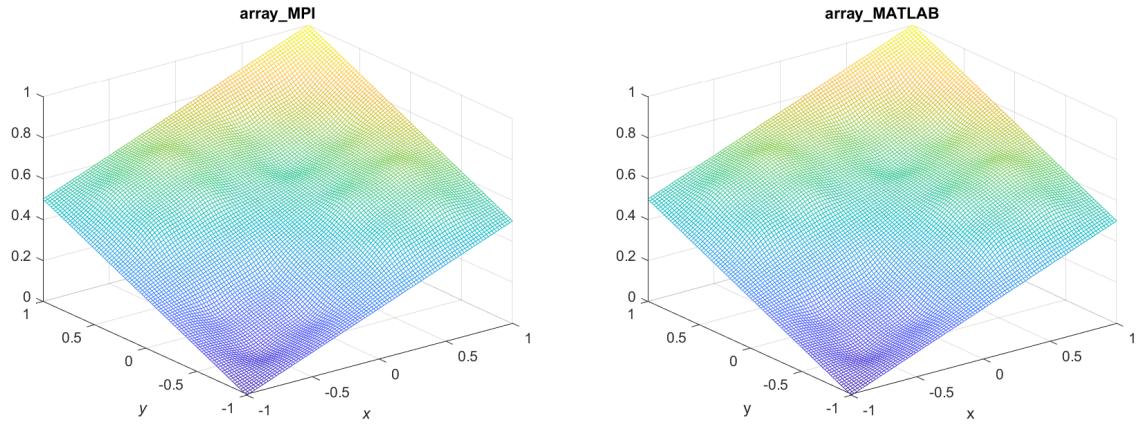
Para esta alínea pretende-se modificar as fronteiras anteriormente iguais a zero para que $V(-1,y) = \frac{1+y}{4}$, $V(x,-1) = \frac{1+x}{4}$, $V(x,1) = \frac{3+x}{4}$ e $V(1,y) = \frac{3+y}{4}$. Para isto tiveram que se efetuar mudanças no código como apresentado em Código 1. Aqui alteram-se as condições fronteira da matriz inicial sendo iguais para ambas `Vnew` e `Vold`.

Código 1 – Alterações ao código para a Alínea **A)** da alteração das condições fronteira.

```
if (newid == 0 || newid == 1){ // y = -L, V(x,-1)
   for (int j = 0; j < ncols+2; j++){
      Vnew[0][j] = (3 + (-L + h*(firstcol+j-1)))/4;
      Vold[0][j] = Vnew[0][j];
   }
}
if (newid == nprocs - 2 || newid == nprocs - 1){ // y = L, V(x,1)
   for (int j = 0; j < ncols + 2; j++){
      Vnew[nrows+1][j] = (1 + (-L + h*(firstcol+j-1)))/4;
      Vold[nrows+1][j] = Vnew[nrows+1][j];
   }
}
if (newid % 2 == 0){
   for (int i = 1; i < nrows + 1; i++){ // x = -L, V(-1,y)
      Vnew[i][0] = (1 + (L - h*(firstrow+i-1)) )/4;
      Vold[i][0] = Vnew[i][0] ;
   }
}
else{
   for (int i = 1; i < nrows + 1; i++){ // x = -L, V(1,y)
      Vnew[i][ncols+1] = (3 + (L - h*(firstrow+i-1)) )/4;
      Vold[i][ncols+1] = Vnew[i][ncols+1];
   }
}
```

Os resultados obtidos podem-se visualizar na Figura 2 onde se efetua a comparação entre a

figura do resultado obtido em MPI[1], 2a, e do sequencial[2], 2b, sendo que se verifica que estas são bastante similares. Esta semelhança é ainda mais evidente com a Tabela 2 onde se verifica que foram necessárias mais iterações que no caso das fronteiras iguais a 0. Isto resultou num tempo de computação superior com as diferenças entre as superfícies a serem maiores neste caso que no anterior.



(a) MPI (b) Sequencial

Figura 2 – Comparativo das superfícies obtidas pelo método paralelizado em MPI e sequencial em *Matlab* para a resolução da Alínea **A)**.

Tabela 2 – Número de iterações e tempos de execução para a versão MPI com o erro entre este e o método sequencial em Matlab para a Alínea **A)**.

Número de iterações	Tempo de cálculo(s)	Tempo de escrita(s)	MSE	MDF
11804	0.390259	0.003893	0.092831	0.50641

Alínea **B)**

Para esta alínea assumiram-se as condições fronteira periódicas e por isso a inicialização das matrizes é feita a 0 não se utilizando as condições da Alínea **A)**. Assim foi necessário efetuar alterações ao código como são demonstradas no Código 2. Primeiramente é necessário alterar o valor de `periodic` para `{1,1}`, `alteracao 1`, de forma a se implementarem as condições periódicas mais facilmente. De seguida, foi também necessário alterar o código da definição do domínio de cálculo, `alteracao 2`, para a inclusão dos pontos que não são os fantasmas. Quando ao momento da escrita também é necessário efetuar modificações para que fossem tidos em conta todos os pontos do domínio global, `alteracao 3`, em que já não é necessário efetuar os passos da alínea anterior para não se considerarem os elementos fantasma.

[1] Código em `alinea_a.c`
[2] Código em `alinea_a.m`

Código 2 – Alterações ao código para a Alínea **B)** para a consideração das fronteiras periódicas.

```
//alteracao 1
int periodic[2] = {1, 1};
...
//alteracao 2
for (int i = 0; i < dim0; i++){
   listfirstrow[2*i] = i * nrowsP;
   listnrows[2*i] = nrowsP+1;
   listfirstrow[2*i+1] = i * nrowsP;
   listnrows[2*i+1] = nrowsP+1;
}
listfirstrow[nprocs - 1] = 1 + (dim0 - 1) * nrowsP;
listnrows[nprocs - 1] = ny - (dim0 - 1) * nrowsP;
listfirstrow[nprocs - 2] = 1 + (dim0 - 1) * nrowsP;
listnrows[nprocs - 2] = ny - (dim0 - 1) * nrowsP;
int ncols_temp = (int)((nx-2)/2);
for (int i = 0; i < dim0; i++){
   listfirstcol[2*i] = 0;
   listncols[2*i] = ncols_temp + 1;
   listfirstcol[2*i+1] = ncols_temp + 1;
   listncols[2*i+1] = nx - 1 - ncols_temp;
}
...
//alteracao 3
int lsizes[2] = {nrows, ncols};
int start_ind[2] = {firstrow, firstcol};
```

Na Figura 3 são apresentadas as superfícies obtidas do método em MPI[3] e do sequencial[4], em que ambos consideram as fronteiras periódicas. Como se pode verificar estas são bastante similares. Analisando a Tabela 3 consegue-se também perceber que a diferença entre ambos os resultados obtidos é reduzida, tendo aqui o método convergido mais rapidamente, tanto temporalmente como em relação ao número de iterações face à alínea anterior, Alínea **A)**. Relativamente ao código original é esperado que se verifiquem ligeiras diferentes, em especial nas fronteiras, algo que acontece ligeiramente.

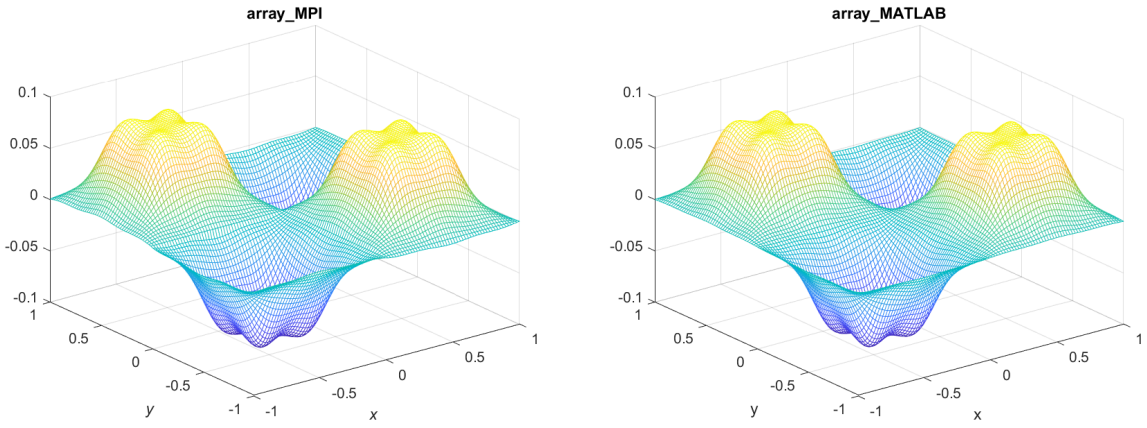
Tabela 3 – Número de iterações e tempos de execução para a versão MPI com o erro entre este e o método sequencial em Matlab para a Alínea **B)**.

Número de iterações	Tempo de cálculo(s)	Tempo de escrita(s)	MSE	MDF
3872	0.143638	0.003901	0.0078451	0.16187

Alínea **C)**

Com esta alínea pretende-se melhorar o método desenvolvido anteriormente para em vez de serem considerados apenas 4 vizinhos se utilizarem 8, passando a usarem-se 9 pontos para o

[3] Código em `alinea_b.c`
[4] Código em `alinea_b.m`



(a) MPI (b) Sequencial

Figura 3 – Comparativo das superfícies obtidas pelo método paralelizado em MPI e sequencial em *Matlab* para a resolução da Alínea **B)**.

cálculo, isto reduz o erro da ordem de h^2 para h^4. Para isto foi necessário efetuar alterações ao código como é apresentado em Código 3.

Inicialmente foi definido o valor de `W` que é necessário ao método de Jacobi ponderado. De seguida, já que são necessários mais 4 pontos para efetuar o cálculo é necessário alocar espaço para estas linhas e colunas fantasmas extra, `alteracao 4`. Dado que estas não fazem parte do domínio é necessário alterar os índices para o cálculo de f, `myf`, feito na `alteracao 5`. Na `alteracao 6` efetua-se a concretização do método como apresentado no enunciado.

Devido à inclusão de mais elementos fantasma que não são para escrever é também necessário efetuar alterações ao nível da escrita. Com a `alteracao 7` esta modificação é feita para se escreverem exclusivamente os pontos desejados. Inclusive, a cada iteração é necessário efetuar também a comunicação das novas linhas e colunas para as fantasma em que é necessário ter em especial atenção os índices, `alteracao 8`, para que a primeira linha atualizada passe a ser a penúltima e a segunda a última seguindo-se a mesma lógica para receber as atualizações e de forma idêntica para as colunas. No final de cada iteração é também necessário atualizar a matriz `Vold` para os novos valores `Vnew`, `alteracao 9`.

Código 3 – Alterações ao código para a Alínea **C)** para a consideração das fronteiras periódicas com o método de Jacobi ponderado.

```
#define W (15.0/16.0)
...
//alteracao 4
double (*Vnew)[ncols + 4];
double (*Vold)[ncols + 4];
double (*myf)[ncols + 4];
Vnew = calloc(nrows + 4, sizeof(*Vnew));
Vold = calloc(nrows + 4, sizeof(*Vold));
myf = calloc(nrows + 4, sizeof(*myf));

//alteracao 5
for (int j = 2; j <= ncols +2 ; j++){
```

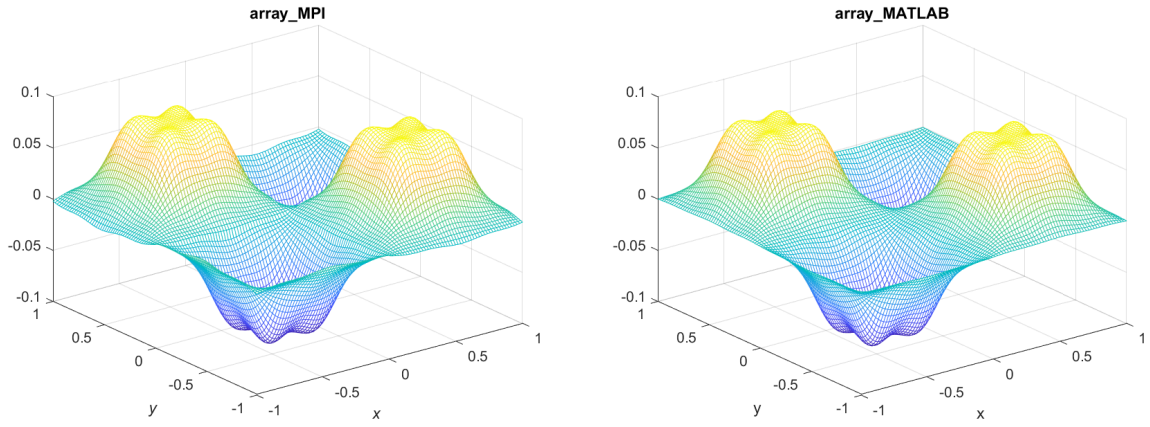
```
   for (int i = 2; i <= nrows +2; i++){
      myf[i][j] = f(-L + (firstcol + j - 2) * h, -L + (firstrow + i - 2) * h);
   }
}
MPI_Type_vector((nrows + 4), 1, (ncols + 4) , MPI_DOUBLE, &column);
...
//alteracao 6
for (int j = 2; j < ncols + 2 ; j++){
   for (int i = 2; i < nrows + 2; i++){
      Vnew[i][j] = (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];
   }
}
...
//alteracao 7
int memsizes[2] = {nrows+4, ncols+4};
start_ind[0] = 1;
start_ind[1] = newid % 2;
if (newid == 0 || newid == 1) {
   start_ind[0]--;
}
...
//alteracao 8
MPI_Sendrecv(Vnew[nrows], ncols+4, MPI_DOUBLE, nbrbottom, 0, Vnew[0] , ncols+4,
    MPI_DOUBLE, nbrtop, 0, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(Vnew[2], ncols+4, MPI_DOUBLE, nbrtop, 1, Vnew[nrows+2] , ncols+4,
    MPI_DOUBLE, nbrbottom, 1, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(Vnew[0][ncols]), 1, column, nbrright, 2, &(Vnew[0][0]), 1, column,
    nbrleft, 2, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(Vnew[0][2]), 1, column, nbrleft, 3, &(Vnew[0][ncols+2]), 1, column,
    nbrright, 3, comm2D, MPI_STATUS_IGNORE);


MPI_Sendrecv(Vnew[nrows+1], ncols+4, MPI_DOUBLE, nbrbottom, 4, Vnew[1] , ncols+4,
    MPI_DOUBLE, nbrtop, 4, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(Vnew[3], ncols+4, MPI_DOUBLE, nbrtop, 5, Vnew[nrows+3] , ncols+4,
    MPI_DOUBLE, nbrbottom, 5, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(Vnew[0][ncols+1]), 1, column, nbrright, 6, &(Vnew[0][1]), 1, column,
    nbrleft, 6, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&(Vnew[0][3]), 1, column, nbrleft, 7, &(Vnew[0][ncols+3]), 1, column,
    nbrright, 7, comm2D, MPI_STATUS_IGNORE);

//alteracao 9
for (int i = 0; i < nrows + 4; i++){
   for (int j = 0; j < ncols + 4; j++){
      Vold[i][j] = Vnew[i][j];
   }
}
```

Da Figura 4 podem-se visualizar as superfícies obtidas em cada um dos métodos, do sequen-

cial[5] com o método de Jacobi ponderado e do em MPI[6]. Constata-se que estas são similares mas com o reparo de que foi necessário alterar a rotação de *array_MPI*, 4a, para que esta superfície ficasse na mesma orientação de 4b, algo que se pode dever à troca de algum sinal na implementação da função. Verifica-se também que junto à fronteira, $x = -1$ e $y < 0$, em 4a este é mais subido que em 4b. Dos resultados numéricos, Tabela 4, comparativamente ao anterior foram necessárias mais iterações e por si um maior tempo computacional mas as diferenças entre as superfícies, os erros medidos, foram ligeiramente inferiores.



(a) MPI (b) Sequencial

Figura 4 – Comparativo das superfícies obtidas pelo método paralelizado em MPI e sequencial em *Matlab* para a resolução da Alínea **C)**.

Tabela 4 – Número de iterações e tempos de execução para a versão MPI com o erro entre este e o método sequencial em Matlab para a Alínea **C)**.

Número de iterações	Tempo de cálculo(s)	Tempo de escrita(s)	MSE	MDF
4966	0.259101	0.002288	0.007782	0.161

Alínea **D)**

Nesta secção o objetivo parte pela alteração do método, alterando de Jacobi para Gauss-Seidel. Este método tem a especificidade de que para a atual iteração já são utilizados os valores mais recentemente calculados, e por isso é necessário efetuar a comunicação dos valores mais recentemente calculados para que os outros processos também tenham acesso a estes [5].

Dado que é utilizado o estêncil de 5 pontos e não o de 9 como na Alínea **C)** são apenas necessárias duas linhas e duas colunas fantasmas considerando-se assim as condições fronteiras como na Alínea **B)**. Por isso, foi utilizado o código desta alínea e foram efetuadas as modificações apresentadas em Código 4. Aqui o método de Jacobi foi substituído pelo que se apresenta.

[5] Código em `alinea_c.m`
[6] Código em `alinea_c.c`

Primeiramente efetua-se o cálculo para os elementos par, sendo de seguida efetuada a comunicação aos vizinhos calculando-se os ímpares. Sendo no fim as linhas fantasmas processadas como na Alínea **B)** para a não escrita destas na matriz de saída.

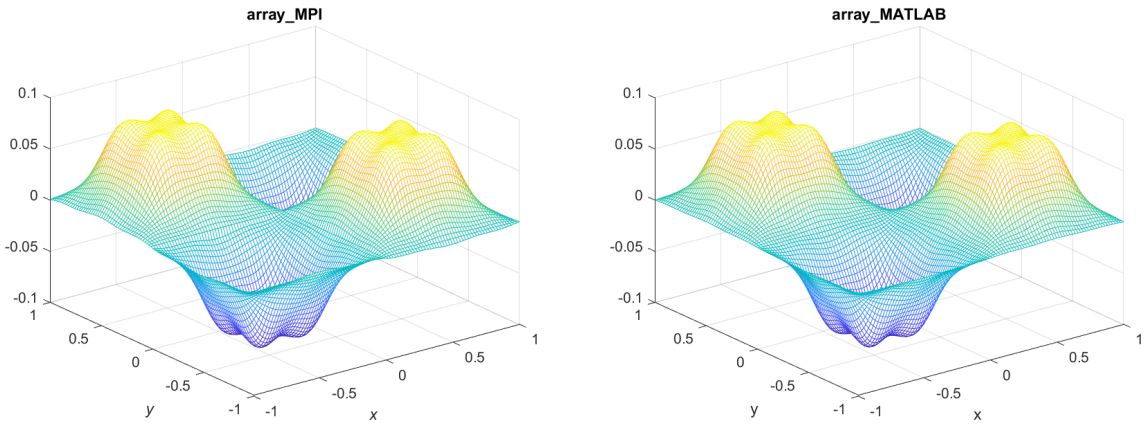
Código 4 – Alterações ao código para a Alínea **D)** para a consideração das fronteiras periódicas com o método de Gauss-Seidel.

```
for (int i = 1; i < nrows + 1; i++){
   for (int j = 1; j < ncols + 1 ; j++){
      if (((firstcol + j - 1) + (firstrow + i - 1)) %2 == 0){ // 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
MPI_Sendrecv(&Vnew[1][1], ncols, MPI_DOUBLE, nbrtop, 4, &Vnew[nrows+1][1], ncols,
    MPI_DOUBLE, nbrtop, 4, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&Vnew[nrows][1], ncols, MPI_DOUBLE, nbrbottom, 5, &Vnew[0][1], ncols,
    MPI_DOUBLE, nbrtop, 5, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&Vnew[1][1], 1, column, nbrleft, 6, &Vnew[1][ncols+1], 1, column,
    nbrright, 6, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&Vnew[1][ncols], 1, column, nbrright, 7, &Vnew[1][0], 1, column,
    nbrleft, 7, comm2D, MPI_STATUS_IGNORE);
// impar
for (int i = 1; i < nrows + 1; i++){
   for (int j = 1; j < ncols + 1 ; j++){
      if (((firstcol + j - 1) + (firstrow + i - 1)) %2 == 1){ // impar
         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
MPI_Sendrecv(&Vnew[1][1], ncols, MPI_DOUBLE, nbrtop, 8, &Vnew[nrows+1][1], ncols,
    MPI_DOUBLE, nbrbottom, 8, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&Vnew[nrows][1], ncols, MPI_DOUBLE, nbrbottom, 9, &Vnew[0][1], ncols,
    MPI_DOUBLE, nbrtop, 9, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&Vnew[1][1], 1, column, nbrleft, 10, &Vnew[1][ncols+1], 1, column,
    nbrright, 10, comm2D, MPI_STATUS_IGNORE);
MPI_Sendrecv(&Vnew[1][ncols], 1, column, nbrright, 11, &Vnew[1][0], 1, column,
    nbrleft, 11, comm2D, MPI_STATUS_IGNORE);
```

Na Figura 5 verificam-se as superfícies obtidas pelo método sequencial[7] também a tomar o método de Gauss-Seidel, e pelo em MPI[8]. Como se pode visualizar estas são muito semelhantes algo que também se confirma com as diferenças obtidas, Tabela 5. Nesta é também de notar que o número de iterações foi reduzido para quase metade em comparação à Alínea **B)**. O tempo

[7] Código em `alinea_d.m`
[8] Código em `alinea_d.c`

de cálculo foi também mais reduzido mas tendo em conta que são necessárias efetuar mais comunicações para ambientes de utilização em que estas sejam feitas mais lentamente, por exemplo em computadores ligados em rede, pode o mesmo nem sempre se verificar.



(a) MPI (b) Sequencial

Figura 5 – Comparativo das superfícies obtidas pelo método paralelizado em MPI e sequencial em *Matlab* para a resolução da Alínea **D)**.

Tabela 5 – Número de iterações e tempos de execução para a versão MPI com o erro entre este e o método sequencial em Matlab para a Alínea **D)**.

Número de iterações	Tempo de cálculo(s)	Tempo de escrita(s)	MSE	MDF
2113	0.118033	0.002271	0.0078492	0.16191

Conclusões

Assim com este trabalho considera-se que se atingiram os pontos pretendidos a desenvolver presentes no enunciado. Também foram testadas as condicionantes de um número de processadores ímpar e para um valor superior a 4, sendo que também se apresentou como funcional e com resultados muito semelhantes que não são apresentados devido à pouca informação que iria acrescentar. Os resultados obtidos foram sempre de encontro aos verificados nos métodos sequenciais também concretizados. Os valores de tempo medido são no entanto apenas referente a uma tentativa e por isso não são representativos o suficiente. Contudo, o número de iterações foi sempre o mesmo uma vez que este processo é determinístico, para os mesmos dados de entrada, obtém-se a mesma solução.

Um aspeto que deve também ser tido em conta com o desenvolvimento deste tipo de aplicações paralelizadas é o seu caso de uso. A concretização de um programa sequencial pode apresentar um desempenho inferior em alguns casos dependendo do tamanho do problema em questão mas se estes forem raros para o uso em produção o tempo de desenvolvimento é também

necessário de ter em conta. Adicionalmente, existem múltiplas formas de paralelizar uma resolução ao problema em causa e algumas são mais eficazes que outras mas também apresentam especificidades. A utilização de CUDA, por exemplo, para a resolução deste sistema poderia ter sido mais fácil uma vez que se pode fazer uso da memória global do dispositivo que é também bastante rápida. Contudo, para dimensões muito elevadas que requerem muita memória pode ser necessário fazer uso de diversos dispositivos, equipamentos em rede, sendo também mais economicamente fazível. A possibilidade de também poder especificar apenas no momento de execução o número de processos a atribuir a implementação em MPI pode ser vantajosa.

Bibliografia

[1] Jehoshua Bruck et al. “Efficient message passing interface (MPI) for parallel computing on clusters of workstations”. Em: *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*. 1995, pp. 64–73.

[2] Peter Corbett et al. “Overview of the MPI-IO Parallel I/O Interface”. Em: *Input/Output in Parallel and Distributed Computer Systems*. Ed. por Ravi Jain, John Werth e James C. Browne. Boston, MA: Springer US, 1996, pp. 127–146. ISBN: 978-1-4613-1401-1. DOI: 10.1007/978-1-4613-1401-1_5. URL: https://doi.org/10.1007/978-1-4613-1401-1_5.

[3] Rolf Hempel. “The MPI standard for message passing”. Em: *High-Performance Computing and Networking*. Ed. por Wolfgang Gentzsch e Uwe Harms. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 247–252. ISBN: 978-3-540-48408-0.

[4] CORPORATE The MPI Forum. “MPI: a message passing interface”. Em: *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*. 1993, pp. 878–883.

[5] Jianping Zhu. *Solving Partial Differential Equations on Parallel Computers*. WORLD SCIENTIFIC, 1994. DOI: 10.1142/2190. eprint: https://www.worldscientific.com/doi/pdf/10.1142/2190. URL: https://www.worldscientific.com/doi/abs/10.1142/2190.