Computação Paralela/Avançada (2016/2017)

Trabalhos para a avaliação final

A escolha do trabalho final tem ser comunicada até 19/12/2016 para `pedro.alberto@uc.pt` *para aprovação, com indicação de 3 trabalhos ordenados por ordem de preferência.*

Nota: os exemplos de código da referência [1] destinam-se *apenas* a ajudar na resolução dos problemas!

Os ficheiros e relatório tem de ser enviados para o inforestudante até 23/1/2017.

Trabalho 1

Considere a multiplicação de uma matriz A por uma matriz B

$$C = AB \ ,$$

onde A, B e C são matrizes quadradas $n \times n$.

1. Faça um programa série que calcule os elementos c_{ij} de C, dados os elementos da matrizes A e B (todos em dupla precisão) para uma certa dimensão n. Use uma rotina de sistema, para calcular o tempo que demora a calcular C para vários valores de n (considere valores de n até 8000).

2. Elabore agora um programa em MPI alocando cada n_p linhas de A, B e C a cada um dos p processos ($n_p = n/p$) para fazer o mesmo cálculo. O processo terá de envolver uma operação *gather*, uma vez que o elemento c_{ij} de C envolve o produto da linha i da matriz A com a coluna j da matriz B (ver secção 7.1 da ref. [1]). Obtenha o tempo de computação total usando a rotina `MPI_WTIME()`.

3. Considere o algoritmo de Fox para a multiplicação de matrizes (ver secção 7.2 da ref. [1]). O algoritmo consiste em alocar para cada processo p submatrizes de A e B, A_{ij} e B_{ij} respectivamente, de dimensão $n/\sqrt{p}$ sendo $i, j = 0, \ldots \sqrt{p} - 1$ (considere só valores de p que sejam quadrados perfeitos, até $p = 16$). A submatriz C_{ij} é dada por ($q = \sqrt{p}$)

   $$C_{ij} = A_{ii}B_{ij} + A_{i\,i+1}B_{i+1\,j} + \cdots + A_{i\,q-1}B_{q-1\,j} + \ A_{i\,0}B_{0\,j} + A_{i\,1}B_{1\,j} + \cdots + A_{i\,i-1}B_{i-1\,j}$$

   O algoritmo envolve, pois, a atribuição de um par de índices (i, j) para cada processo e consiste basicamente nos seguintes passos, para cada processo:

   - um "broadcast"das submatrizes A_{ij} com índice de linha i para todas os processos (i, j) com o mesmo índice i;
   - ciclo sobre o índice de coluna $\bar{k}$, em que $\bar{k} = (i+k) \bmod q$, $i = 0, \ldots, q-1$ em que se realiza a soma acima, um termo de cada vez;
   - envio ("send") da submatriz $B_{\bar{k}j}$ para o processo "de cima"(linha $i-1$) e recepção ("receive") da submatriz $B_{\bar{k}+1\,j}$ do processo "abaixo"(linha $i+1$).

   Implemente este algoritmo usando uma topologia virtual de duas dimensões e usando `MPI_CART_SUB` para criar comunicadores para sub-grelhas adequadas para os processos de "broadcast", "send"e "receive"descritos acima.

4. Compare, para matrizes iguais, os tempos de cálculo de cada algoritmo e comente o resultado. Faça também um estudo do "speed-up"para cada algoritmo para $p = 4, 9, 16$. Comente os resultados.

Trabalho 2

Considere o mesmo problema do trabalho 1, mas agora envolvendo o chamado produto "min-plus" entre matrizes. Para matrizes quadradas $n \times n$ A, B e C, o produto $C = A \star B$ é definido tal que o elemento ij de C é dado por

$$c_{ij} = \min_{1 \leq k \leq n} \{a_{ik} + b_{kj}\} \qquad i, j = 1, \ldots, n \ . \tag{1}$$

A expressão do algoritmo de Fox para as sub-matrizes C_{ij} toma agora a forma

$$C_{ij} = \min \{A_{ii} \star B_{ij} \, , A_{i\,i+1} \star B_{i+1\,j} \, , \cdots , A_{i\,q-1} \star B_{q-1\,j} \, , A_{i\,0} \star B_{0\,j} \, , A_{i\,1} \star B_{1\,j} \, , \cdots \, , A_{i\,i-1} \star B_{i-1\,j}\}$$

ou seja, as somas são substituídas por minimizações e as multiplicações das submatrizes por produtos do tipo (1).

O algoritmo descrito no trabalho 1 é executado exactamente da mesma forma, substituindo as somas descritas no ponto 3 desse trabalho por minimizações.

Realize as tarefas descritas nos pontos 1, 2, 3 e 4 do trabalho 1 tendo em conta as definições deste produto de matrizes.

Trabalho 3

Considere a resolução de um sistema de n equações lineares expresso pela equação matricial

$$A\boldsymbol{x} = \boldsymbol{b} \ , \tag{2}$$

onde A é a matriz quadrada $n \times n$ não singular dos coeficientes, $\boldsymbol{x}$ é o vector solução e $\boldsymbol{b}$ o vector não nulo dos elementos do 2º membro das equações. O método de Jacobi é um método iterativo em que o elemento i de $\boldsymbol{x}$, x_i, é calculado a partir da linha i da equação (2) usando o vector solução anterior (ver secção 10.2 da ref. [1]). Assim, para a iteração $k+1$ tem-se

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right) \ .$$

Há garantia de convergência se a matriz A for *diagonalmente dominante*, ou seja, se $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ para todos os $i = 1, \ldots, n$. O teste da convergência faz-se calculando a diferença

$$d(k+1) = \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| = \sqrt{\sum_{i=1}^{n} (x_i^{(k+1)} - x_i^{(k)})^2} \ .$$

O processo iterativo termina quando $d(k+1) < \texttt{tol}$, sendo `tol` uma tolerância pré-definida.

1. Faça um programa série que calcule as soluções x_i do sistema de equações lineares, dados os elementos da matriz A e do vector $\boldsymbol{b}$ (todos em dupla precisão) para uma certa dimensão n. Para testar o método, considere o caso particular de A ser diagonal $a_{ij} = \alpha_i \delta_{ij}$, $\alpha_i > 0$, para o qual as soluções são conhecidas (quais são?).

   Use uma rotina de sistema, para calcular o tempo que demora a calcular $\boldsymbol{x}$ para vários valores de n (considere valores de n até 8000) e para $\texttt{tol} = 10^{-6}$. Para estes cálculos considere matrizes A diagonalmente dominantes, não diagonais e não singulares. Defina um valor `nitmax` tal que, se não houver convergência ao fim de `nitmax` iterações, o sistema interrompa o cálculo e mande uma mensagem de erro ao utilizador. Considere $\boldsymbol{x}^{(0)} = \boldsymbol{b}$.

2. Elabore agora um programa em MPI para p processadores que implemente o método de Jacobi paralelo (ver secção 10.3 da ref. [1]). Considere dimensões n que sejam divisíveis por p. Cada processo calcula $n_p = n/p$ componentes de $\boldsymbol{x}$, usando a respectiva submatriz de A. Tenha atenção que todos os processos têm de ter disponível *todo* o vector $\boldsymbol{x}$ da iteração anterior, bem como o vector $\boldsymbol{b}$. Use o mesmo controle da convergência e do número máximo de iterações do programa série.

3. Compare, para matrizes iguais (com os mesmo elementos), os tempos de cálculo de algoritmo série e paralelo para $p = 2, 4, 8, 16$ e comente o resultado.

4. Indique, com algum detalhe, uma implementação eficiente do método de Jacobi paralelo para o caso geral em que n não é múltiplo de p.

Trabalho 4

Considere o condensador cúbico da figura. A aresta do condutor externo é 3 vezes a aresta do condutor interno (ver figura). Considere $V_1 = 5$ e $V_2 = 10$. Para determinar os valores do potencial no espaço entre os condutores cúbicos tem de se resolver numericamente a equação de Laplace tridimensional $\nabla^2 V = 0$. O método de Jacobi aplicado a este caso implica a definição de uma grelha tridimensional, sendo o valor do potencial no ponto da grelha com os índices i, j, k, para a iteração número $ni + 1$, dado por

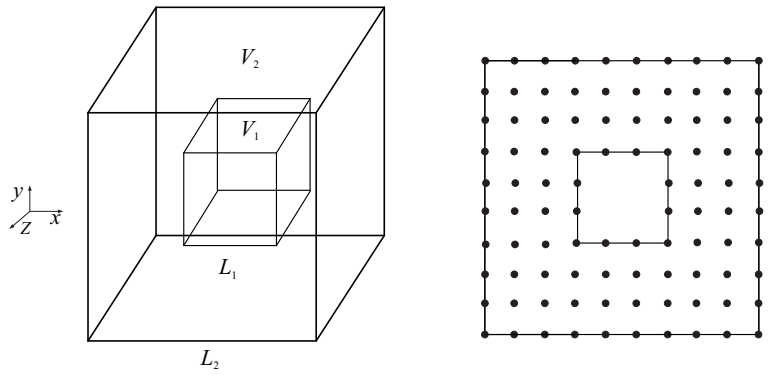
$$V^{(ni+1)}(i,j,k) = \frac{1}{6}[V^{(ni)}(i-1,j,k) + V^{(ni)}(i+1,j,k) + V^{(ni)}(i,j-1,k) + V^{(ni)}(i,j+1,k) + V^{(ni)}(i,j,k-1) + V^{(ni)}(i,j,k+1)]$$

(**Nota importante**: a grelha da figura é *apenas um exemplo*, não tem de ser a adoptada. Deve procurar definir-se a grelha mais edequada à divisão do domínio de integração que se escolha)

1. Escreva um programa MPI para resolver o problema acima, considerando que o valor inicial para o potencial é um valor constante V_0 tal que $V_1 < V_0 < V_2$. Defina uma topologia virtual tridimensional para implementar o método de Jacobi neste caso, definindo as subarrays tridimensionais `vlocal(i,j,k)` para cada processo apropriadamente, com a sua parte interior e as células fantasma ("ghost").

2. Como critério para a obtenção de uma solução adequada, no fim de cada iteração ni calcule a diferença
$$\texttt{dif} = \sum_{i,j,k} |V^{(ni)}(i,j,k) - V^{(ni-1)}(i,j,k)|/|V^{(ni-1)}(i,j,k)| \ ,$$
incluindo os valores de V para *toda* a grelha (veja [2], cap. 4), e verifique se ela é menor que um certo valor `tol`. Neste caso, considere $\texttt{tol} = 10^{-5}$. Estabeleça, à semelhança do trabalho 1, um número máximo de iterações ao fim do qual o programa termina.

3. Escreva o resultado final (coordenadas i, j, k e valores de V) num ficheiro. Opcionalmente, represente a solução num gráfico a 3 dimensões em que se represente a grelha e os valores do potencial através de cores com o GNUPLOT (ver, por exemplo, `http://objectmix.com/graphics/139988-4d-data-plotting-gnuplot.html`) .



Trabalho 5

Considere o sistema de condutores cilíndricos da figura. Os valores dos raios dos condutores interno e externo são, repectivaente, $R_1 = 1$ e $R_2 = 2$ (ver figura) e a altura é $L = 5$. O valor do potencial para $z = 0$ e $z = L$ (as bases do cilindro, que se supõem isoladas dos condutores laterais) é de 7. Considere $V_1 = 5$ e $V_2 = 10$. Para determinar os valores do potencial no espaço entre os condutores cilindricos tem de se resolver numericamente a equação de Laplace tridimensional $\nabla^2 V = 0$ nas coordenadas cilíndricas (r, φ, z). O método de Jacobi aplicado a este caso dá a fórmula

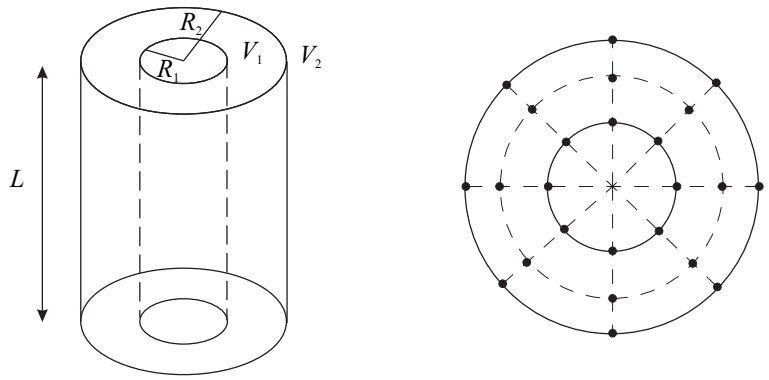
$$\begin{aligned} V^{(ni+1)}(i,j,k) &= \frac{1}{2}\frac{h^2\, r_i^2\, l^2\, m^2}{h^2\, r_i^2 l^2 + h^2\, m^2 + r_i^2\, l^2\, m^2} \times \\ &\left\{ V^{(ni)}(i-1,j,k)\left(\frac{1}{h^2} - \frac{1}{2hr_i}\right) + V^{(ni)}(i+1,j,k)\left(\frac{1}{h^2} + \frac{1}{2hr_i}\right) + \right. \\ &\left. [V^{(ni)}(i,j-1,k) + V^{(ni)}(i,j+1,k)]\frac{1}{l^2 r_i^2} + [V^{(ni)}(i,j,k-1) + V^{(ni)}(i,j,k+1)]\frac{1}{m^2} \right\} \end{aligned} \tag{3}$$

onde i, j, k designam os índices da grelha tridimensional tal que os valores discretos de (r, φ, z) são

$$\begin{aligned} r_i &= R_1 + i\, h \qquad & i &= 0, \ldots, n_r - 1 \qquad & h &= \frac{R_2 - R_1}{n_r - 1} \\ \varphi_j &= j\, l \qquad & j &= 0, \ldots, n_\varphi - 1 \qquad & l &= \frac{2\pi}{n_\varphi} \\ z_k &= k\, m \qquad & k &= 0, \ldots, n_z - 1 \qquad & m &= \frac{L}{n_z - 1} \end{aligned} \tag{4}$$

sendo n_r, n_φ, n_z respectivamente os números de pontos da grelha correspondentes às coordenadas (r, φ, z).

Resolva numericamente este problema seguindo os pontos 1, 2 e 3 do trabalho 4.



Referências

[1] Peter Pacheco, Parallel Programming with MPI, Morgan Kaufmann Publishers.

[2] W. Gropp, E. Lusk and A. Skjellum, Using MPI Portable Parallel Programming with the Message Passing Interface, 2nd Edition, MIT Press.

Computação Paralela/Avançada (2016/2017)

Final projects

The choice of the final project has be sent until 19/12/2016 to `pedro.alberto@uc.pt` *for approval, with the indication of 3 projects ordered by preference.*

Note: the code examples in reference [1] are meant *just* to help in problem resolution!

Files and reports are to be sent to inforestudante until 23/1/2017.

Project 1

Consider the multiplication of two matrices A and B

$$C = AB \ ,$$

where A, B and C are square matrices $n \times n$.

1. Write a serial program to compute the elements c_{ij} of C, given all elements of matrices A and B (all in double precision) for a given dimension n. Use a system call to obtain the time it takes to calculate C for several values of n (consider values of n up to 8000).

2. Write a MPI program to perform the same calculation by allocating n_p lines of A, B and C to each of the p processes ($n_p = n/p$). The calculation would include a *gather* operation, since the computation of element c_{ij} of C involves the product of line i of matrix A with column j of matrix B (see section 7.1 of ref. [1]). Obtain the total computation time using the function `MPI_WTIME()`.

3. Consider now the Fox algorithm for the multiplication of matrices (see section 7.2 of ref. [1]). It consists of in allocating for each process p submatrices of A and B, A_{ij} and B_{ij} respectively, of dimension $n/\sqrt{p}$ with $i, j = 0, \ldots \sqrt{p} - 1$ (consider only values of p which are perfect squares, up to $p = 16$). The submatrix C_{ij} is given by ($q = \sqrt{p}$)

   $$C_{ij} = A_{ii}B_{ij} + A_{i\,i+1}B_{i+1\,j} + \cdots + A_{i\,q-1}B_{q-1\,j} + \ A_{i\,0}B_{0\,j} + A_{i\,1}B_{1\,j} + \cdots + A_{i\,i-1}B_{i-1\,j}$$

   the algorithm involves the assignment of a pair of indices (i, j) to each process and consists basically in the following steps, for each process:

   - a *broadcast* of the submatrices A_{ij} with line index i for each process (i, j) with the same index i;
   - cycle over the column index $\bar{k}$, in which $\bar{k} = (i + k) \bmod q$, $i = 0, \ldots, q - 1$ in which the sum above is performed, term by term;
   - *send* submatrix $B_{\bar{k}j}$ to the process "above"(line $i - 1$) and *receive* submatrix $B_{\bar{k}+1\,j}$ from the process "below"(line $i + 1$).

   Implement this algorithm using a 2-dimensional virtual topology and using `MPI_CART_SUB` to create communicators for sub-grids appropriate for the *broadcast*, *send* and *receive* actions described above.

4. Compare, for equal matrices, the compute times for each algorithm and comment the result. Compute the speed-up for each algorithm for $p = 4, 9, 16$ and comment the results as well.

Project 2

Consider the problem as in project 1, but now involving the "min-plus" product between matrices. For square matrices $n \times n$ A, B e C, the product $C = A \star B$ is defined such that the element ij of C is given by

$$c_{ij} = \min_{1 \leq k \leq n} \{a_{ik} + b_{kj}\} \qquad i, j = 1, \ldots, n \ . \tag{5}$$

The Fox algorithm for submatrices C_{ij} is given by

$$C_{ij} = \min \{A_{ii} \star B_{ij} \, , A_{i\,i+1} \star B_{i+1\,j} \, , \cdots , A_{i\,q-1} \star B_{q-1\,j} \, , A_{i\,0} \star B_{0\,j} \, , A_{i\,1} \star B_{1\,j} \, , \cdots \, , A_{i\,i-1} \star B_{i-1\,j}\}$$

that is, the sums are replaced by minimizations and the multiplications of submatrices by products of the type (5).

The algorithm described in project 1 is executed exactly in the same way, replacing the sums described in point 3 of that project by minimizations.

Perform the tasks described in points 1, 2, 3 and 4 of project 1 taking into account the definition of this matrix product.

Project 3

Consider the problem of solving a system of n linear equations given by the matrix equation

$$A\boldsymbol{x} = \boldsymbol{b} \ , \tag{6}$$

where A is a square non-singular $n \times n$ matrix of the equations coeficients, $\boldsymbol{x}$ is the solution vector and $\boldsymbol{b}$ is the non-null vector containing the values of the right-hand side of each linear equation. The Jacobi method is an iterative method by which the element i of $\boldsymbol{x}$, x_i, is calculated from the ith line of equation (6) using the former solution vector (see section 10.2 of ref. [1]). Thus, for the iteration $k+1$ one has

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j^{(k)}\right) \ .$$

The convergence is guaranteed if the matrix A is *diagonally dominant*, that is, if $|a_{ii}| > \sum_{j\neq i}|a_{ij}|$ for all $i = 1, \ldots, n$. The convergence test is performed by calculating the difference

$$d(k+1) = \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| = \sqrt{\sum_{i=1}^{n}(x_i^{(k+1)} - x_i^{(k)})^2} \ .$$

The iterative process ends when $d(k+1) < \texttt{tol}$, where `tol` is a pre-defined tolerance.

1. Write a serial program which calculates the solutions x_i of the system of linear equations, given a matrix A and a vector $\boldsymbol{b}$ (all in double precision) of dimension n. For testing the method, consider the special case of A being diagonal $a_{ij} = \alpha_i\delta_{ij}$, $\alpha_i > 0$, for which the solutions are known (which are they?).

   Use a system routine to calculate the time it takes to calculate $\boldsymbol{x}$ for several values of n (consider values until 10000) and for $\texttt{tol} = 10^{-6}$. For these calculations consider diagonally dominant matrices A which are also non-diagonal and non-singular. Define a value `nitmax` such that, if there is no convergence after `nitmax` iteractions, the program stops and sends an error message to the user. Consider $\boldsymbol{x}^{(0)} = \boldsymbol{b}$.

2. Write a MPI program for p processors which implements the Jacobi parallel method (see section 10.3 of ref. [1]). Consider matrix dimensions n which are divisible by p. Each process calculates $n_p = n/p$ components of $\boldsymbol{x}$, using the respective submatrix of A. Note that every process need to have the *whole* vector $\boldsymbol{x}$ from the previous iteration, as well as the vector $\boldsymbol{b}$. Use the same convergence criterium and the maximum number of iterations of the serial program.

3. Compare, for the same matrices (i.e, with same elements), the computation time for the serial and parallel algorithm for $p = 2, 4, 8, 16$ as a function of p and comment the result.

4. Describe, with some detail, a efficient implementation of the Jacobi method for the general case for which n is not multiple of p.

Project 4

Consider the cubic capacitor of the figure. The size of the edge of external cubic conductor is 3 times the size of the edge of the internal conductor (see figure). The values of the constant potentials of the internal and external conductors are $V_1 = 5$ and $V_2 = 10$ respectively. To determine the value of the potential in the space between the the conductors one has to solve numerically the 3-dimensional Laplace equation $\nabla^2 V = 0$. The Jacobi method applied to this case requires the definition of a 3-dimensional grid, and the value of the potential at each grid point with indexes i, j, k, for the $ni + 1$ iteration is given by

$$V^{(ni+1)}(i,j,k) = \frac{1}{6}[V^{(ni)}(i-1,j,k) + V^{(ni)}(i+1,j,k) + V^{(ni)}(i,j-1,k) + V^{(ni)}(i,j+1,k) + V^{(ni)}(i,j,k-1) + V^{(ni)}(i,j,k+1)]$$

(**Important remark**: the grid in the figure is *just an example*, does not have to be the grid used. One should define the best grid to perform the division of the chosen integration domain)

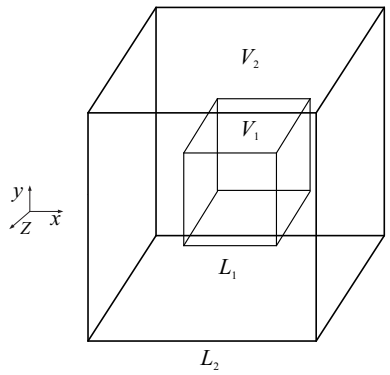
1. Write a MPI program to solve the problem above, considering that the initial value for the potential is a constant value V_0 such that $V_1 < V_0 < V_2$. Create a 3-dimensional cartesian virtual topology to implement the Jacobi method for this problem, defining the three-dimensional subarrays `vlocal(i,j,k)` for each process with its inner part and the ghost cells.

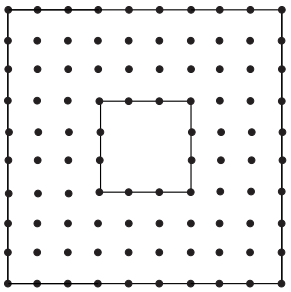
2. As a convergence criterium at the end of each iteration ni compute the difference

$$\texttt{dif} = \sum_{i,j,k} |V^{(ni)}(i,j,k) - V^{(ni-1)}(i,j,k)| \ ,$$

   including the values of V for *all* the grid (see [2], cap. 4), and check if this difference is less than a certain value `tol`. Set $\texttt{tol} = 10^{-5}$. Establish a maximum number of iterations such that the program stops if that number is reached.

3. Write the final result (coordinates i, j, k and values of V) in a file. As an option, make a 3D plot of the solution with the grid and the potential values through color with GNUPLOT (see, for example, `http://objectmix.com/graphics/139988-4d-data-plotting-gnuplot.html`) .





Project 5

Consider the system of cilindrical conductors in the figure. The value of the internal and external conductors are, respectively, $R_1 = 1$ and $R_2 = 2$ (see figure) and the height is $L = 5$. The value of the potential for $z = 0$ and $z = L$ (the cylinder bases, considered isolated from the lateral conductors) is 7. Consider $V_1 = 5$ and $V_2 = 10$. To determine the values of potential in the space between the cylindrical conductors on ehas to solve numerically the 3-dimensional Laplace equation $\nabla^2 V = 0$ in cylindrical coordinates (r, φ, z). The Jacobi method applied to this case gives the formula

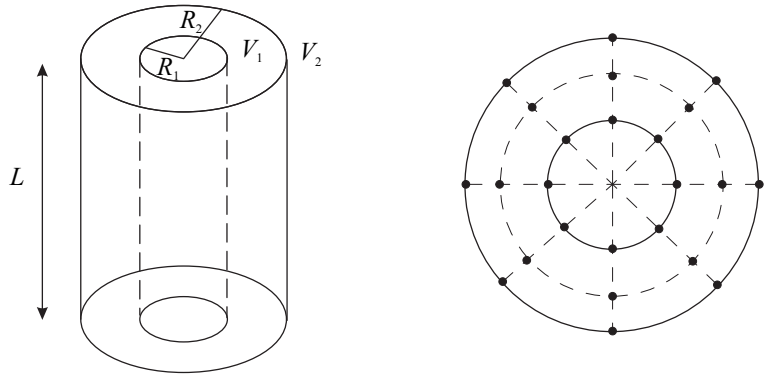
$$
\begin{aligned}
V^{(ni+1)}(i,j,k) \;=\; & \frac{1}{2}\frac{h^2\, r_i^2\, l^2\, m^2}{h^2\, r_i^2 l^2 + h^2\, m^2 + r_i^2\, l^2\, m^2} \times \\
& \left\{ V^{(ni)}(i-1,j,k)\left(\frac{1}{h^2} - \frac{1}{2hr_i}\right) + V^{(ni)}(i+1,j,k)\left(\frac{1}{h^2} + \frac{1}{2hr_i}\right) + \right. \\
& \left. [V^{(ni)}(i,j-1,k) + V^{(ni)}(i,j+1,k)]\frac{1}{l^2 r_i^2} + [V^{(ni)}(i,j,k-1) + V^{(ni)}(i,j,k+1)]\frac{1}{m^2} \right\}
\end{aligned}
\tag{7}
$$

where i, j, k denote the tridimensional grid points such that the corresponding coordinates (r, φ, z) are

$$
\begin{aligned}
& r_i = R_1 + i\, h \qquad && i = 0, \ldots, n_r - 1 \qquad && h = \frac{R_2 - R_1}{n_r - 1} \\
& \varphi_j = j\, l \qquad && j = 0, \ldots, n_\varphi - 1 \qquad && l = \frac{2\pi}{n_\varphi} \\
& z_k = k\, m \qquad && k = 0, \ldots, n_z - 1 \qquad && m = \frac{L}{n_z - 1}
\end{aligned}
\tag{8}
$$

and n_r, n_φ, n_z are the number of grid points along r, φ, z directions respectively.

Solve numerically this problem along the points 1, 2 and 3 of project 4.



Referências

[1] Peter Pacheco, Parallel Programming with MPI, Morgan Kaufmann Publishers.

[2] W. Gropp, E. Lusk and A. Skjellum, Using MPI Portable Parallel Programming with the Message Passing Interface, 2nd Edition, MIT Press.