Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería en Computadores
Curso: CE-3102 Análisis Numérico para Ingeniería
Profesor: Juan Pablo Soto Quirós
Grupo: 01

Tarea 2
Parte 1 - Punto 3:
Implementación en Paralelo del Método de Jacobi en GNU Octave

Estudiantes:

Fiorella Delgado Leon - 2017121626

Cristian Marín Murillo - 2016134345

Randy Martínez Sandí - 2014047395

Karla Michelle Rivera Sánchez - 2016100425

Domingo 29 de noviembre, 2020
Cartago

Resumen

El presente documento consiste en el desglose y la explicación de cómo implementar el método de Jacobi para la solución de sistemas de ecuaciones (matriciales) en paralelo.

Implementación en Paralelo del Método de Jacobi

a. Fórmula del Método de Jacobi (Forma No Matricial)

El método de Jacobi es una forma de encontrar solución a sistemas de ecuaciones. Mediante la ecuación:

$$Ax = b$$

donde A representa la matriz que contiene los coeficientes, x las incógnitas a calcular y b los términos independientes que complementan la igualdad.

Para calcular el vector x, la fórmula de Jacobi tiene la siguiente forma:

$$x_i^{(k+1)} = \frac{1}{A_{i,i}} \left(b_i - \sum_{j=1\ ;\ j \neq i}^{m} A_{i,j} * x_j^{(k)}\right)$$

b. Paralelismo

A la hora de implementar la fórmula anterior, se deben hacer más ejecuciones debido a que la matriz puede tener m ecuaciones. Este proceso puede llegar a ser lento debido a que si la matriz A es muy grande, se van a realizar tantos cálculos como filas en A, uno tras otro.

Una forma de solucionar esta cuestión es realizar los cálculos utilizando paralelismo. Es decir, se desea evitar que el valor de de x sea obtenido utilizando todos los procesadores para el mismo cálculo en cada iteración consecutiva. En su

lugar, se busca utilizar todos los procesadores, al mismo tiempo en múltiples cálculos diferentes, para una ejecución total más veloz.

Una manera de ilustrar tal premisa es mediante la ***Figura 1***.

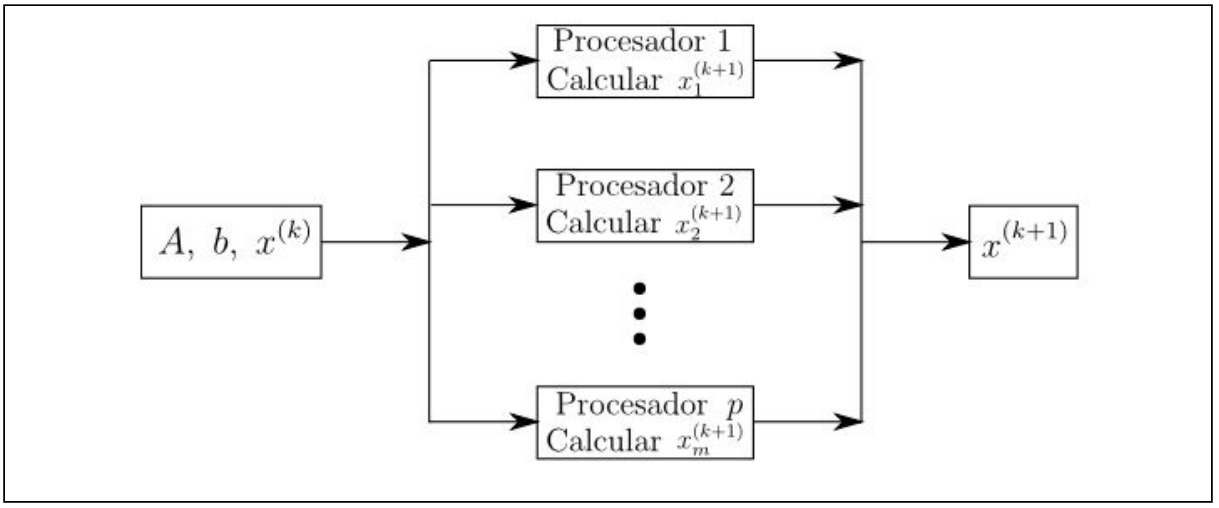

Figura 1. *Paralelismo aplicado a p procesadores para el cálculo de x.*

c. Implementación Propuesta por los Autores

Descripción

La solución creada por los autores para la ejecución del Método Jacobi en paralelo, se basó en la solución secuencial. Es decir, se reescribió el código y se sustituyeron las iteraciones que ofrecían un trabajo secuencial por herramientas ofrezcan un resultado en paralelo.

Para esto, se utilizó el paquete `Parallel,` el cual está disponible en el sitio *web:* https://octave.sourceforge.io/packages.php. El mismo ofrece las herramientas para poder ejecutar cualquier solución programada de manera paralela. El funcionamiento interno de esto es ajeno a los usuarios. No obstante, la premisa general se basa en utilizar los núcleos físicos y lógicos disponibles en cada computador y dividir los procesos (de cálculos) en los núcleos para una mayor rapidez en las soluciones.

De manera de código, los autores crearon una función llamada **parte1_p3.m.** La cual aplica la función `pararrayfun` del paquete de `Parallel,` que realiza una distribución de los cálculos en los núcleos disponibles.

Para poder aplicar la función previamente mencionada, se efectuaron algunos cambios en la sección iterativa del *while*, ya que, acá se realiza el cálculo de Jacobi, pero en esta nueva implementación se hacen ajustes al método. Estos cambios se construyeron con una nueva función llamada **jacobi.m,** la cual contiene la misma fórmula con ligeros ajustes.

Estos pasos, cambios y códigos se van a explicar en la siguiente subsección, ya que, se van a mostrar los pasos que se ejecutan. Posteriormente extractos del código que se utilizó.

Pseudocódigo

Pasos para la implementación de `pararrayfun()`:

I. Cargar el paquete `Parallel`.
II. Inicializar el vector x_k.
III. Asignar, a las variables m y n, el tamaño de filas y columnas de la matriz de entrada A, respectivamente.
IV. Inicializar el vector inicial x_0 de tamaño m.
V. Asignar el valor del vector x_0 al vector x_k.
VI. Definir la tolerancia de $tol = 10^{-5}$.
VII. Definir error $err = tol + 1$.
VIII. Definir la variable de iteraciones $iter = 0$.
IX. Entrada del ciclo definido por las condiciones de parada $tol < err \wedge iter < 1000$.
X. Definición de la función anónima f_jacobi como la función auxiliar que se envía a la función paralela.
XI. Cálculo del vector resultante x_k utilizando la función `pararrayfun()` del paquete `Parallel`.
XII. Cálculo del error $err = norm(A \cdot x_k - b)$ usando la norma 2.
XIII. Incrementar en 1 la cantidad de iteraciones para el ciclo.

```
    % carga del paquete parallel %
    pkg load parallel

    % declaración: vector resultante %
    xk = [];

    % calculo: dimensiones de la matriz A %
    [m, n] = size(A);

        % declaración: vector inicial %
        x0 = zeros(m, 1);

        % asignación: vector resultante %
        xk = x0;

        % declaración: tolerancia %
        tol = 10^-5;

        % declaración: error %
        err = tol + 1;

        % declaracion: número de iteraciones realizadas %
        iter = 0;

        % iteración: mientras el error sea mayor que la %
        % tolerancia y las iteraciones realizadas sean  %
        % menor que 1000, ejecuta la siguiente serie    %
        while(tol < err && iter < 1000)

            % declaracion: funcion auxiliar del metodo de jacobi    %
            f_jacobi = @(r) jacobi(A, b, xk, m, r);

            % calculo: del vector resultante utilizando las funciones
            % paralelas del paquete 'parallel'
            xk = pararrayfun(nproc, f_jacobi, 1 : m);

            % calculo: error mediante la norma 2 %
            err = norm(A * xk - b);

            % aumento del contador de iteraciones realizadas %
            ++iter;
        end
```

Pasos para la implementación de la función Jacobi:

I. Definición de la función $x_k = jacobi(A, b, x_k, m, r)$, con los parámetros de entrada:
   - A : matriz de coeficientes.
   - b : matriz columna de términos independientes.
   - x_k :vector inicial.
   - m : grado de la matriz.
   - r : vector de los índices de la matriz A.

II. Entrada del ciclo que recorre cada fila de la matriz A.

III. Declaración de la variable suma $suma = 0$.

IV. Entrada del ciclo que recorre cada columna de cada fila de la matriz A.

V. Condición para realizar el cálculo de $suma$ mientras $i \neq j$.

VI. Cálculo de la suma $suma = A_{i,j} \cdot x_{k_j}$.

VII. Salida de la condición y del ciclo de recorrido de filas y columnas.

VIII. Cálculo del vector resultante con los valores de la aproximación $x_k = \frac{1}{A_{r,r}} \cdot (b_r - suma)$.

```
function xk = jacobi (A, b, xk, m, r)

    % iteracion: recorrido por cada fila de la matriz A %
    for( i = 1 : m)

        % declaracion: valor resultante de la serie %
        suma = 0;

         % iteracion: recorrido por cada columna de la matriz A %
        for (j = 1 : m)

           % verificacion: si 'i' no es igual a 'j' realice la suma, si no, salto %
            if (i != j)

                % calculo: de la suma de (Aij * xki + Aij+1 * xk2)%
                suma = suma + A(i, j) * xk(j);
```

```
17              end
18          end
19
20      end
21          % calculo: calculando secuencialmente del valor final xk(i)
con la formula de la serie
22      xk(r) = 1/A(r, r) * (b(r) - suma);
23  end
```