Métodos numéricos - Tarea 8
Método de la Jacobi
Cómputo paralelo

Salim Vargas Hernández

7 de octubre del 2018

1. Introducción

El método de Jacobi para el cálculo de valores y vectores propios es un método iterativo que permite aproximar la diagonalización de la matriz A simétrica, es decir, aproximar la forma $P^T AP = \Lambda$ donde Λ es una matriz diagonal que contiene en la diagonal a los eigenvalores de A y P es una matriz ortogonal con los vectores propios de A.

La paralelización es un procedimiento que permite aprovechar los cores de una computadora multicore, de forma que una operación o cálculo que se realiza normalmente en serie, se puede realizar a trozos en cada uno de los cores de la máquina, para depués colapsar el resultado en uno solo. La ventaja de la paralelización es que, para problemas muy grandes, permite realizar estas operaciones en un menor tiempo que el que tomaría hacerlas en serie.

2. Desarrollo

2.1. Método de Jacobi

El método de Jacobi consiste en aplicar rotaciones estratégicas en ciertos puntos de la matriz simétrica A para diagonalizarla. El algoritmo 1 muestra el pseudocódigo del método de Jacobi.

3. Resultados

3.1. Método de Jacobi

Se probó el método de Jacobi con las matrices `mat5a.txt`, `mat5b.txt` y `mat10.txt`. Para lograr la convergencia se utilizaron los siguientes parámetros:

Parámetro	`mat5a.txt`	`mat5b.txt`	`mat10.txt`
tolerancia	0.0001	0.0001	0.001
maxIter	1,000,000	1,000,000	1,000,000

Para cada matriz se imprimían los eigenvalores obtenidos. Los resultados se muestran en la figura 1.

Algoritmo 1 Método de Jacobi para valores y vectores propios

Entrada:
Matriz simétrica A
tolerancia tol
máximo de iteraciones $maxIter$

Salida: Matriz diagonal C con los valores propios
Matriz ortogonal P con los vectores propios

$C = A$
$P = I_N$
Mientras haya valores arriba de la tolerancia fuera de la diagonal y las iteraciones sean menores que $maxIter$ **hacer**
  max = máximo en valor absoluto de C
  $[k, l]$ = posición de max en C
  Si $|max| > tol$ **entonces**
    Si $C_{kk} == C_{ll}$ **entonces**
      $\theta = \frac{\pi}{4}$
    Si no
      $\theta = arctan\left(\frac{2C_{kl}}{C_{kk} - C_{ll}}\right)$
    Fin Si
    $P^{(i)} = I_N$
    $P^{(i)}_{kk} = cos(\theta)$
    $P^{(i)}_{ll} = cos(\theta)$
    $P^{(i)}_{kl} = -sin(\theta)$
    $P^{(i)}_{lk} = sin(\theta)$
    $C = P^{(i)T} C P^{(i)}$
    $P = P P^{(i)}$
  Fin Si
Fin Mientras
Devolver C, P

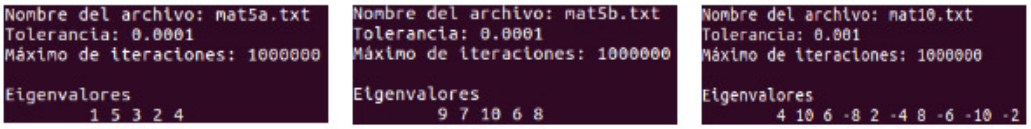

Figura 1: Resultados del método de Jacobi

3.2. Cómputo paralelo

Se hizo uso de `OpenMP` para paralelizar algunas operaciones con vectores y matrices, como fueron:

1. Suma de vectores
2. Producto de vectores elemento a elemento
3. Producto punto entre dos vectores
4. Producto matriz-vector
5. Producto matriz-matriz

Se tomó en cuenta el tiempo de ejecución de las funciones paralelas y el método en serie, así como el *speed-up* y la *eficiencia*

Se generaban vectores y matrices aleatorias de tamaño especificado por el usuario; asimismo se pedía al usuario el número de *threads* a utilizar en la paralelización.
Los resultados se muestran en las figuras 2 a la 6.

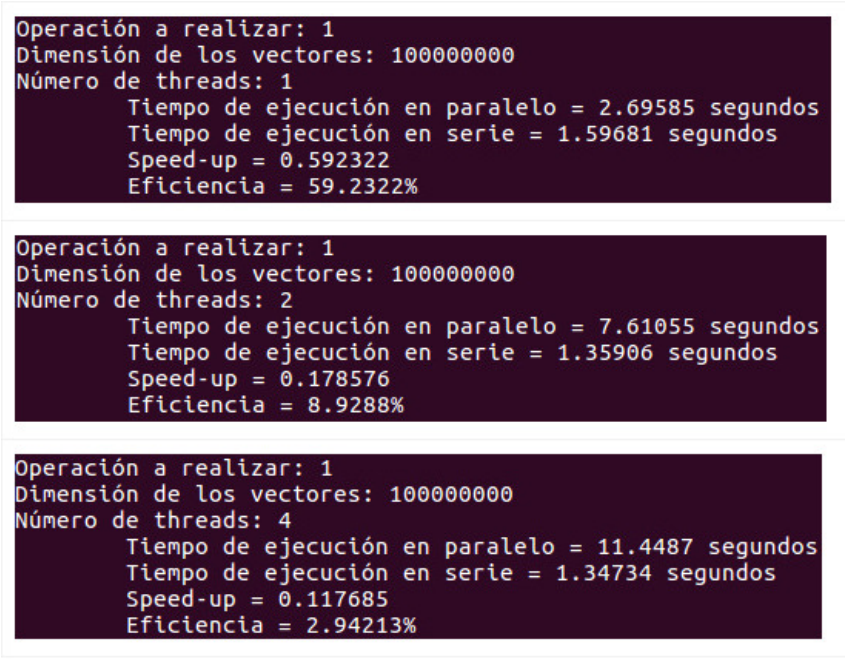

Figura 2: Resultados de la operación 1

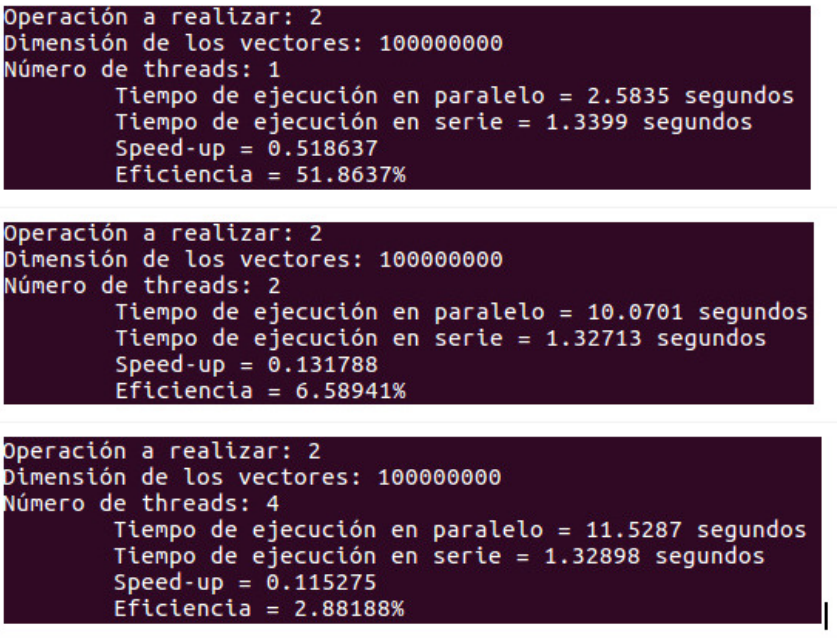

Figura 3: Resultados de la operación 2

```
Operación a realizar: 3
Dimensión de los vectores: 100000000
Número de threads: 1
        Tiempo de ejecución en paralelo = 1.98251 segundos
        Tiempo de ejecución en serie = 0.655635 segundos
        Speed-up = 0.33071
        Eficiencia = 33.071%
```

```
Operación a realizar: 3
Dimensión de los vectores: 100000000
Número de threads: 2
        Tiempo de ejecución en paralelo = 8.25783 segundos
        Tiempo de ejecución en serie = 0.666994 segundos
        Speed-up = 0.0807711
        Eficiencia = 4.03855%
```

```
Operación a realizar: 3
Dimensión de los vectores: 100000000
Número de threads: 4
        Tiempo de ejecución en paralelo = 12.3339 segundos
        Tiempo de ejecución en serie = 0.666639 segundos
        Speed-up = 0.0540495
        Eficiencia = 1.35124%
```

Figura 4: Resultados de la operación 3

```
Operación a realizar: 4
Dimensión 1 de la matriz: 10000
Dimensión 2 de la matriz: 10000
Número de threads: 1
        Tiempo de ejecución en paralelo = 2.03109 segundos
        Tiempo de ejecución en serie = 0.645681 segundos
        Speed-up = 0.317899
        Eficiencia = 31.7899%
```

```
Operación a realizar: 4
Dimensión 1 de la matriz: 10000
Dimensión 2 de la matriz: 10000
Número de threads: 2
        Tiempo de ejecución en paralelo = 2.25068 segundos
        Tiempo de ejecución en serie = 0.680713 segundos
        Speed-up = 0.302448
        Eficiencia = 15.1224%
```

```
Operación a realizar: 4
Dimensión 1 de la matriz: 10000
Dimensión 2 de la matriz: 10000
Número de threads: 4
        Tiempo de ejecución en paralelo = 2.7608 segundos
        Tiempo de ejecución en serie = 0.649015 segundos
        Speed-up = 0.235082
        Eficiencia = 5.87706%
```

Figura 5: Resultados de la operación 4

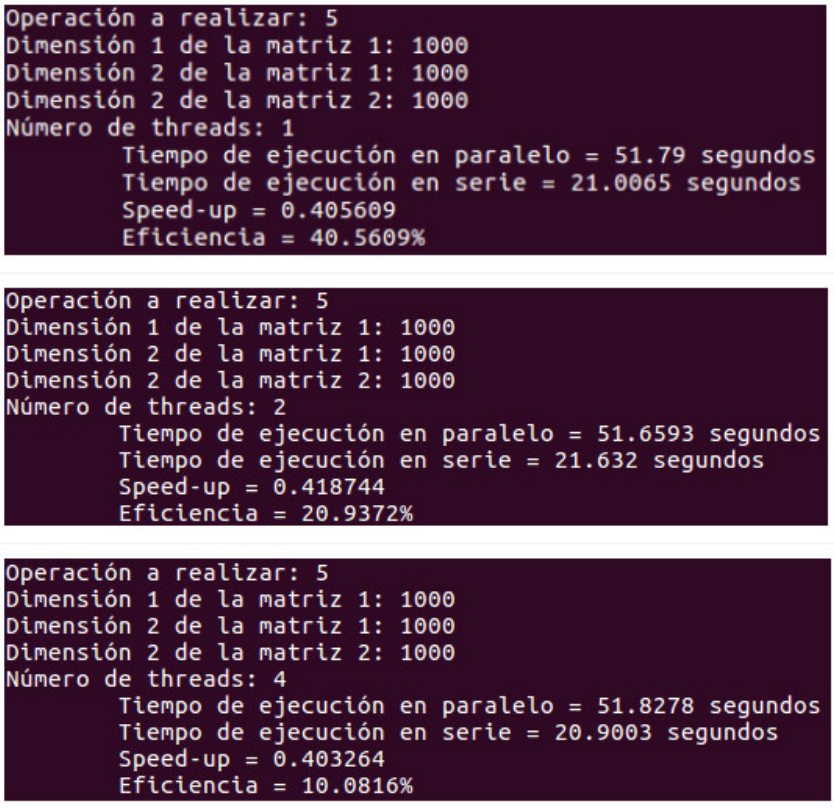

Figura 6: Resultados de la operación 5