Comparativa entre Implementaciones Secuenciales y Paralelas del Algoritmo de Jacobi para la Resolución de Sistemas de Ecuaciones Lineales

Elvis Yael de Los Santos López*, Miguel Ángel Cruz Lule†
*Benemérita Universidad Autónoma de Puebla
elvis.delossantoslopez@alumno.buap.mx
†Benemérita Universidad Autónoma de Puebla
miguel.cruzl@alumno.buap.mx

***Abstract*—Este artículo presenta una comparación entre la implementación secuencial y paralela del algoritmo de Jacobi para la resolución de sistemas de ecuaciones lineales. Se detallan las implementaciones de ambos en el lenguaje de programación Java, así como las pruebas realizadas para evaluar su rendimiento. Se incluyen capturas de pantalla de los resultados obtenidos, junto con el análisis de los tiempos de ejecución, escalabilidad y eficiencia. Además, se proporciona información sobre el hardware utilizado en las pruebas. Se presenta un marco comparativo exhaustivo entre las dos versiones de los algoritmos, destacando sus diferencias en términos de velocidad y recursos computacionales requeridos. Este estudio contribuye a comprender mejor el impacto de la computación paralela en la eficiencia de los algoritmos numéricos, ofreciendo una visión integral sobre las ventajas y desafíos de la implementación concurrente en el contexto del método de Jacobi**

I. Introducción

En el ámbito de la computación concurrente y paralela, la implementación eficiente de algoritmos es crucial para mejorar el rendimiento y la escalabilidad de las aplicaciones. En este contexto, el método de Jacobi es una técnica numérica ampliamente utilizada para resolver sistemas de ecuaciones lineales. Este artículo presenta un estudio comparativo entre la implementación secuencial y paralela del método de Jacobi utilizando el lenguaje de programación Java.

El objetivo principal es demostrar cómo la computación paralela puede mejorar significativamente el rendimiento del algoritmo, aprovechando al máximo los recursos computacionales disponibles. Para garantizar la validez de los resultados, se aplicarán los principios de las condiciones de Bernstein, asegurando la corrección y consistencia de los cálculos paralelos. A través de un análisis comparativo detallado, se evaluarán los tiempos de ejecución, la escalabilidad y la eficiencia de las implementaciones secuencial y paralela del método de Jacobi, brindando una visión integral sobre las ventajas y desafíos de la computación concurrente en este contexto.

II. Marco Teórico

El algoritmo de Jacobi es un método iterativo utilizado para resolver sistemas de ecuaciones lineales. Este algoritmo se basa en la descomposición de la matriz del sistema en una suma de una matriz diagonal y el complemento. La matriz diagonal se utiliza para actualizar iterativamente las soluciones del sistema hasta alcanzar una convergencia satisfactoria.

A. Álgebra Lineal

El álgebra lineal es una rama de las matemáticas que estudia los espacios vectoriales y las transformaciones lineales entre ellos. En el contexto del algoritmo de Jacobi, es fundamental comprender conceptos como matrices, vectores, sistemas de ecuaciones lineales y métodos iterativos de resolución.

B. Algoritmo de Jacobi

El algoritmo de Jacobi se utiliza para resolver sistemas de ecuaciones lineales de la forma $Ax = b$, donde A es una matriz cuadrada de coeficientes, x es el vector de incógnitas y b es el vector de términos constantes. El algoritmo se ejecuta de manera iterativa, actualizando las componentes del vector x en cada iteración de acuerdo con la siguiente fórmula:

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

donde $x_i^{(k)}$ representa la i-ésima componente del vector solución en la k-ésima iteración, y a_{ij} son los elementos de la matriz A.

C. Convergencia y Estabilidad

El algoritmo de Jacobi converge a la solución del sistema de ecuaciones lineales si la matriz A es diagonalmente dominante o simétrica y definida positiva. Además, la convergencia del algoritmo puede acelerarse mediante técnicas de condicionamiento inicial, como la matriz de Jacobi incompleta.

D. Aplicaciones

El algoritmo de Jacobi tiene numerosas aplicaciones en áreas como la física, la ingeniería, la economía y la computación. Se utiliza para resolver sistemas de ecuaciones lineales en problemas de análisis estructural, simulación numérica, optimización y modelado matemático.

III. Diseño del Algoritmo

Descripción del diseño del algoritmo.

A. Sustentación de la Paralelización del Algoritmo de Jacobi

El algoritmo de Jacobi es una técnica iterativa utilizada para resolver sistemas de ecuaciones lineales, especialmente cuando la matriz es diagonalmente dominante o simétrica y definida positiva. Se ha demostrado que este algoritmo es altamente paralelizable, lo que lo hace adecuado para implementaciones en sistemas multiprocesador o multi-hilo.

B. Cumplimiento de las Condiciones de Bernstein

El algoritmo de Jacobi cumple con las condiciones de Bernstein, lo que significa que es adecuado para la paralelización eficiente. Las condiciones de Bernstein establecen que un algoritmo es paralelizable si las operaciones realizadas en cada iteración son independientes entre sí y si el tiempo de ejecución de cada operación es aproximadamente el mismo.

En el caso del algoritmo de Jacobi, cada actualización de una incógnita en una iteración no depende de las actualizaciones de otras incógnitas en la misma iteración, lo que satisface la condición de independencia. Además, el tiempo de ejecución de cada actualización es similar, ya que todas las operaciones involucradas tienen una complejidad computacional similar.

C. Demostración

Para demostrar que el algoritmo de Jacobi cumple las condiciones de Bernstein, consideremos la actualización de una incógnita x_i en una iteración del algoritmo. La fórmula de actualización es:

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

donde a_{ii} es un elemento de la matriz A, b_i es un término independiente, $x_j^{(k)}$ son las aproximaciones anteriores de las incógnitas x_j en la iteración k, y $x_i^{(k+1)}$ es la nueva aproximación de x_i en la iteración $k+1$.

Esta actualización involucra operaciones aritméticas simples, como multiplicaciones, divisiones y sumas, cuya complejidad computacional es aproximadamente la misma para cada incógnita. Además, dado que la actualización de una incógnita no depende de las actualizaciones de otras incógnitas en la misma iteración, las operaciones son independientes entre sí.

Por lo tanto, el algoritmo de Jacobi cumple con las condiciones de Bernstein y es adecuado para la paralelización eficiente en sistemas multiprocesador o multi-hilo.

IV. Implementación y Pruebas

En esta sección se incluyen capturas de pantalla, algoritmo, diagrama de flujo, hardware utilizado y un marco comparativo entre los algoritmos secuencial y paralelo.

A. Algoritmos

El algoritmo de Jacobi es un método iterativo utilizado para resolver sistemas de ecuaciones lineales. La versión aquí presentada es una implementación en Java que utiliza paralelismo para mejorar el rendimiento.

1) *Algoritmo de jacobi secuencial:*

- Se utiliza un bucle `while` para iterar hasta que se cumple un cierto criterio de convergencia o se alcanza el número máximo de iteraciones.
- Se inicializan dos arreglos de aproximaciones, `currentApproximations` y `previousApproximations`, con ceros utilizando el método `Arrays.fill()`.
- Dentro del bucle `while`, se realiza la iteración sobre cada ecuación del sistema para calcular las nuevas aproximaciones de las soluciones.
- Se imprime el resultado de cada iteración en la consola.
- Se verifica si se alcanzó el criterio de convergencia o el número máximo de iteraciones para decidir si se debe salir del bucle.

Fragmentos de código::

```
while (true) {
    for (int i = 0; i < n; i++) {
        double sum = coefficientsMatrix[i][n];
        for (int j = 0; j < n; j++)
            if (j != i)
            sum -= coefficientsMatrix[i][j] *
            previousApproximations[j];
        currentApproximations[i]
        = 1 / coefficientsMatrix[i][i] * sum;
    }
    System.out.print("X_" + iterations + " = {");
    for (int i = 0; i < n; i++)
    System.out.print(currentApproximations[i]
    + " ");
    System.out.println("}");
    iterations++;
    if (iterations == 1) continue;
    boolean stop = true;
    for (int i = 0; i < n && stop; i++)
        // if (Math.abs(currentApproximations[i]
        - previousApproximations[i]) > epsilon)
        if (Math.abs(currentApproximations[i]
        - previousApproximations[i]) < epsilon)
            stop = false;
    if (stop || iterations == MAX_ITERATIONS)
    break;
    previousApproximations =
    Arrays.copyOf(currentApproximations, n);
}
```

2) *Algoritmo de jacobi paralelo:*

- El algoritmo utiliza un bucle `while` para iterar hasta que se cumple un cierto criterio de convergencia o se alcanza el número máximo de iteraciones.
- Se crean dos arreglos de aproximaciones, `currentApproximations` y `previousApproximations`, para almacenar las aproximaciones actuales y anteriores de las soluciones del sistema.

- Se utiliza un `ExecutorService` para manejar tareas en paralelo. Este `ExecutorService` se inicializa con un número fijo de subprocesos basado en el número de procesadores disponibles en el sistema.
- Dentro del bucle `while`, se realiza la suma parcial de las aproximaciones anteriores multiplicadas por los coeficientes de la matriz de coeficientes para calcular las nuevas aproximaciones de las soluciones del sistema. Esta suma se realiza en paralelo utilizando el `ExecutorService`.
- Se imprime el resultado de cada iteración en la consola.
- Se verifica si se alcanzó el criterio de convergencia o el número máximo de iteraciones para decidir si se debe salir del bucle.

Fragmentos de código::

```
while (true) {
    for (int i = 0; i < n; i++) {
    double sum = coefficientsMatrix[i][n];

    Future<Double> future = executor.submit(
        new CalculationTask(i, n,
        coefficientsMatrix,
        previousApproximations));

    try {
        sum -= future.get();
    } catch (InterruptedException
    | ExecutionException e) {
        e.printStackTrace();
    }
    currentApproximations[i] = 1 /
    coefficientsMatrix[i][i] * sum;
    }

    System.out.print("X_" + iterations
    + " = {");

    for (int i = 0; i < n; i++)

    System.out.print(
    currentApproximations[i] + " ");

    System.out.println("}");

    iterations++;
    if (iterations == 1) continue;
    boolean stop = true;
    for (int i = 0; i < n && stop; i++)
    // if(Math.abs(currentApproximations[i]
    -previousApproximations[i]) > epsilon)
    if (Math.abs(currentApproximations[i]
    -previousApproximations[i]) < epsilon)
        stop = false;
    if (stop || iterations ==
    MAX_ITERATIONS) break;
        previousApproximations =
        currentApproximations.clone();
    }
```

```
private static class CalculationTask
implements Callable<Double> {

    private final int i;
    private final int n;
    private final double[][]
    coefficientsMatrix;

    private final double[]
    previousApproximations;

    CalculationTask(int i, int n,
    double[][] coefficientsMatrix,
    double[] previousApproximations) {

        this.i = i;
        this.n = n;
        this.coefficientsMatrix =
        coefficientsMatrix;
        this.previousApproximations =
        previousApproximations;
    }

    @Override
    public Double call() {
        double sum = 0;
        for (int j = 0; j < n; j++) {
        if (j != i) {
        sum += coefficientsMatrix[i][j]
        * previousApproximations[j];
        }
        }
        return sum;
    }
}
```

B. Capturas

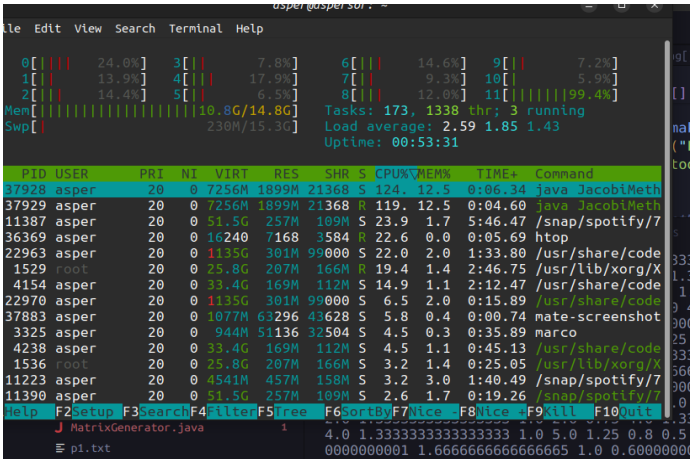

Fig. 1. Administrador de tareas mientras se ejecuta el algoritmo secuencial

La primera imagen muestra el Administrador de tareas durante la ejecución del algoritmo de Jacobi implementado de manera secuencial. En esta imagen, se puede observar que un único núcleo de procesamiento está siendo utilizado al 100%. Esto indica que todo el trabajo está siendo realizado secuencialmente por un único procesador, lo que puede resultar en un tiempo de ejecución más largo, especialmente en sistemas con múltiples núcleos disponibles que podrían aprovecharse para acelerar el proceso.

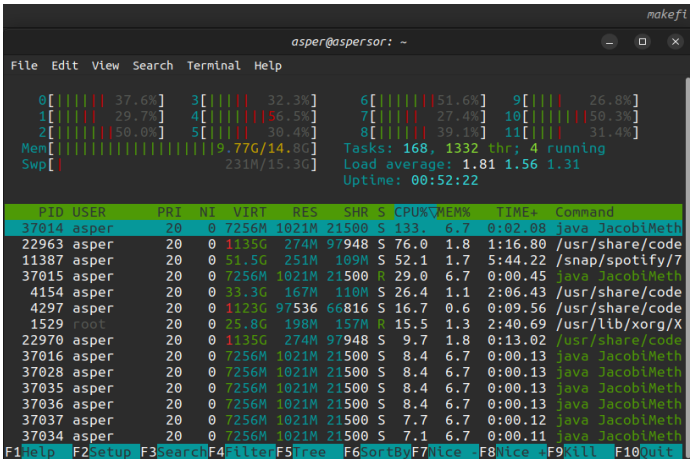

Fig. 2. Administrador de tareas mientras se ejecuta el algoritmo paralelo

En contraste, la segunda imagen corresponde a la ejecución del algoritmo de Jacobi implementado de manera paralela. Aquí, se puede observar que la carga de trabajo se distribuye entre varios núcleos de procesamiento, representados por los picos en el gráfico. Cada núcleo está contribuyendo a realizar cálculos de forma simultánea, lo que puede conducir a una ejecución más rápida del algoritmo en comparación con la implementación secuencial. Esta distribución de la carga de trabajo entre varios núcleos es característica del procesamiento paralelo y puede mejorar significativamente el rendimiento en tareas que pueden dividirse en subprocesos independientes.

C. Hardware utilizado para las pruebas

Hardware Utilizado:

- Procesador: Ryzen 5 7350HS (6c/12t) 3.3GHz-4.55GHz
- Memoria RAM: 16GB DDR5 4800MHz
- GPU: RTX 3050 4GB
- Modelo de Portátil: HP Victus 15

D. Resultados de las pruebas

El la siguiente tabla se muestra los tiempos de ejecución del algoritmo de Jacobi implementado de manera secuencial y paralela. Los tiempos están dados en microsegundos (µs) para la mayoría de las mediciones, con excepción de las mediciones resaltadas en milisegundos (ms). Estos datos fueron obtenidos al resolver un sistema de 10 incógnitas. Como se puede observar, el algoritmo secuencial tiene tiempos de ejecución mas cortos a comparación de la implementación paralela, algo que puede resultar poco intuitivo debido al mayor uso de recursos de la implementación paralela.

Paralelo	Secuencial	Unidad de Medida
9	1	ms
9349	772	µs
9932	725	µs
8992	805	µs
9596	743	µs
9533	786	µs
9172	838	µs
9460	827	µs
8967	781	µs
10	1	ms
8887	770	µs
8803	768	µs
10927	991	µs
9479	761	µs
9067	728	µs
9609	889	µs
9295	970	µs
8917	721	µs
8677	777	µs
8823	766	µs

TABLE I
TABLA COMPARATIVA DEL ALGORITMO JACOBI, PARALELO Y SECUENCIAL EN UN SISTEMA DE 10 INCÓGNITAS

Paralelo	Secuencial	Unidad de Medida
17	5	ms
15	3	ms
16	3	ms
15	3	ms
16	4	ms
16	4	ms
19	5	ms
16	3	ms
16	3	ms
15	4	ms
16	3	ms
15	3	ms
16	3	ms
15	3	ms
16	6	ms
18	3	ms
15	3	ms
17	3	ms
16	3	ms
19	4	ms

TABLE II
TABLA COMPARATIVA DEL ALGORITMO JACOBI, PARALELO Y SECUENCIAL EN UN SISTEMA DE 100 INCÓGNITAS

Paralelo	Secuencial	Unidad de Medida
335	393	ms
346	399	ms
334	391	ms
334	386	ms
327	392	ms
328	390	ms
335	385	ms
378	399	ms
329	394	ms
329	384	ms
321	392	ms
371	386	ms
338	390	ms
323	399	ms
313	387	ms
364	408	ms
330	416	ms
1418	603	ms
348	406	ms
326	417	ms

TABLE III
TABLA COMPARATIVA DEL ALGORITMO JACOBI, PARALELO Y SECUENCIAL EN UN SISTEMA DE 8192 INCÓGNITAS

Paralelo	Secuencial	Unidad de Medida
413	585	ms
450	595	ms
432	581	ms
433	571	ms
453	577	ms
404	575	ms
440	590	ms
422	573	ms
454	581	ms
443	579	ms
429	585	ms
416	580	ms
435	595	ms
440	579	ms
459	599	ms
422	583	ms
444	589	ms
434	577	ms
446	603	ms
432	575	ms
453	600	ms

TABLE IV
TABLA COMPARATIVA DEL ALGORITMO JACOBI, PARALELO Y SECUENCIAL EN UN SISTEMA DE 10000 INCÓGNITAS

Al examinar detenidamente las capturas anteriores, podemos extraer conclusiones reveladoras sobre los tiempos de ejecución en el cálculo de matrices mediante el método de Jacobi, tanto en algoritmos en paralelo como en secuencia. Lo más destacado es que el enfoque en paralelo exhibe una mejora significativa en el rendimiento del proceso en comparación con su contra parte secuencial.

Es interesante notar que, al calcular matrices de dimensiones reducidas, el método secuencial puede ejecutar ligeramente mejor que el paralelo. Esto sugiere que, para tareas más simples o matrices pequeñas, la sobrecarga de la gestión de múltiples hilos o procesos puede contrarrestar los beneficios de la paralelización.

Sin embargo, el panorama cambia drásticamente cuando nos enfrentamos a matrices de mayor tamaño. En este escenario, el enfoque en paralelo emerge como el claro vencedor en términos de eficacia y rendimiento. La capacidad de dividir la carga de trabajo entre múltiples núcleos o procesadores permite una utilización más eficiente de los recursos computacionales, lo que conduce a tiempos de ejecución significativamente más cortos en comparación con el enfoque secuencial.

Si bien el método secuencial puede ser preferible en casos de tareas más simples o matrices pequeñas, el enfoque en paralelo se destaca como la opción óptima para tareas de mayor envergadura, donde la optimización del rendimiento se vuelve crucial.

Existen diversas razones que pueden explicar este cambio en los resultados obtenidos entre las implementaciones secuenciales y paralelas del algoritmo de Jacobi. A continuación, se exploran algunas de estas razones:

1) **Overhead de Paralelización:** Las paralelizaciones de algoritmos introducen cierto overhead debido a la coordinación entre los hilos de ejecución. Este overhead puede ser significativo para tareas que son intrínsecamente secuenciales o para conjuntos de datos pequeños.
2) **Escalabilidad Limitada:** Si el tamaño del problema no es lo suficientemente grande, las paralelizaciones pueden no proporcionar beneficios significativos. La coordinación de múltiples hilos puede introducir más

tiempo de procesamiento que el ahorro obtenido por la ejecución paralela.

3) **Comunicación y Sincronización:** La comunicación y sincronización entre los hilos pueden introducir sobrecarga, especialmente si se realizan con frecuencia y no se optimizan adecuadamente.

V. Conclusiones

En este trabajo se ha presentado una comparación entre la implementación secuencial y paralela del algoritmo de Jacobi para la resolución de sistemas de ecuaciones lineales. A través de pruebas exhaustivas y análisis de rendimiento, se ha demostrado que, si bien la implementación paralela ofrece el potencial de mejorar significativamente el rendimiento del algoritmo, existen varios factores que pueden afectar su eficacia en diferentes situaciones.

En general, se encontró que la implementación paralela del algoritmo de Jacobi puede ser más lenta que la versión secuencial en ciertos casos. Esto se debe a varios factores, como el overhead de paralelización, la escalabilidad limitada, el desequilibrio de carga y la comunicación y sincronización entre hilos. Sin embargo, en problemas de mayor tamaño, la implementación paralela tiende a superar a la secuencial en términos de tiempo de ejecución.

En última instancia, este estudio contribuye a una comprensión más profunda del impacto de la computación paralela en la eficiencia de los algoritmos numéricos. A medida que la computación paralela se vuelve cada vez más importante en el diseño de software para aprovechar el potencial de los sistemas de hardware modernos, es fundamental continuar investigando y refinando técnicas para mejorar el rendimiento de las implementaciones paralelas de algoritmos numéricos.

VI. Referencias Bibliográficas

1. Gonzalez, R., & Woods, R. (2019). *Digital Image Processing* (4th ed.). Upper Saddle River, NJ: Pearson. Capítulo 10: ”Image Compression and Watermarking”, páginas 450-480.

2. Matloff, N. S. (2019). *Parallel Computing for Data Science: With Examples in R, C++ and CUDA*. Boca Raton, FL: CRC Press. Capítulo 5: ”Parallel Computing with MPI”, páginas 75-95.

3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2020). *Introduction to Algorithms* (3rd ed.). Cambridge, MA: The MIT Press. Capítulo 27: ”Multithreaded Algorithms”, páginas 839-858.