Análisis Comparativo y Paralelización de Métodos de Diferencias y Elementos Finitos para la Solución de la Ecuación de Poisson

Kevin Cruz
Universidad Distrital

Junio 2025

Resumen

Este documento presenta una evaluación comparativa de dos métodos numéricos fundamentales, el Método de Diferencias Finitas (FDM) y el Método de Elementos Finitos (FEM), aplicados a la solución de la ecuación de Poisson en dos dimensiones. Se desarrollan e implementan múltiples versiones de cada método en C++, explorando diversas estrategias de paralelización con OpenMP en arquitecturas de memoria compartida. El flujo de trabajo completo, desde la ejecución de benchmarks hasta el análisis de resultados, se automatiza mediante un entorno unificado en Python y Jupyter Notebooks. El objetivo es realizar un análisis riguroso de la eficiencia, escalabilidad y correctitud de cada método, proveyendo una comparativa clara de sus fortalezas y debilidades en un contexto de computación de alto rendimiento.

Índice

1. El Problema Físico: Potenciales en Estado Estacionario

El problema central que se resuelve es la **ecuación de Poisson**, una de las ecuaciones más fundamentales en física e ingeniería. Modela sistemas en **estado estacionario**, es decir, sistemas que han alcanzado un equilibrio y cuyas propiedades ya no varían en el tiempo.

La forma general de la ecuación de Poisson en un dominio bidimensional $\Omega \subset \mathbb{R}^2$ es:

$$-\nabla^2 u(x, y) = f(x, y) \quad \text{en } \Omega \tag{1}$$

donde:

- $u(x, y)$ es el **campo de potencial** que se desea encontrar. Dependiendo del contexto, puede representar la temperatura en una placa, el potencial eléctrico en una región, o la altura de una membrana elástica bajo carga.
- $f(x, y)$ es el **término fuente**, que representa la densidad de "fuentes.º "sumideros"que generan el potencial. Si u es la temperatura, f es una fuente de calor interna. Si u es el potencial eléctrico, f es la densidad de carga.
- ∇^2 es el **operador Laplaciano**, $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$. Físicamente, el Laplaciano de un campo en un punto mide cuánto difiere el valor en ese punto del **promedio** de sus alrededores. Por lo tanto, la ecuación de Poisson establece un balance fundamental: la curvatura o "tensión"del campo potencial en un punto es directamente proporcional a la fuente en ese mismo punto.

Cuando $f(x, y) = 0$, la ecuación se convierte en la **ecuación de Laplace**, que describe sistemas en equilibrio sin fuentes internas. Su solución representa un estado de "máxima suavidad.º mínima energía. Para que la solución sea única, se imponen **condiciones de frontera de tipo Dirichlet**, que fijan el valor del potencial en los bordes del dominio:

$$u(x, y) = g(x, y) \quad \text{en } \partial\Omega \tag{2}$$

2. Solución mediante el Método de Diferencias Finitas (FDM)

2.1. Principio Físico y Discretización

El FDM aborda el problema de una manera directa e intuitiva. El dominio continuo se reemplaza por una **rejilla de puntos discretos**. La filosofía del método es que el valor del potencial en cada punto de la rejilla está determinado por el valor de sus vecinos inmediatos, una manifestación directa de la propiedad de promedio del Laplaciano.

Al discretizar las derivadas con diferencias centradas, la ecuación de Poisson se transforma en una ecuación algebraica para cada nodo interior (i, j), que puede ser reordenada de la siguiente forma para el método de Jacobi:

$$u_{i,j}^{(\text{nuevo})} = \frac{(u_{i+1,j}^{(\text{antiguo})} + u_{i-1,j}^{(\text{antiguo})})k^2 + (u_{i,j+1}^{(\text{antiguo})} + u_{i,j-1}^{(\text{antiguo})})h^2 - f_{i,j}h^2k^2}{2(h^2 + k^2)} \tag{3}$$

donde h y k son los espaciados de la rejilla.

2.2. Implementación y Automatización (FDM)

El código FDM se implementa en C++ utilizando un ‘std::vector<std::vector<double‘ para representar la grilla de potencial. La solución se obtiene mediante el **método de relajación de Jacobi**. Físicamente, este proceso simula un sistema que evoluciona hacia el equilibrio:

1. Se fijan los valores en las fronteras.

2. En cada **iteración**, todos los puntos interiores ajustan su valor simultáneamente para acercarse al "promedio"dictado por el estado de sus vecinos en la iteración anterior.

3. El proceso se repite hasta que los cambios entre iteraciones son insignificantes (menores a una ‘TOLERANCE‘), momento en el cual el sistema ha alcanzado su estado de equilibrio.

La paralelización con OpenMP se aplica a este bucle de actualización, permitiendo que todos los puntos de la rejilla actualicen su valor simultáneamente.

2.2.1. Generalización y Automatización

Para facilitar un análisis sistemático, el código fue refactorizado para ser no interactivo. Un Jupyter Notebook en Python orquesta la ejecución, pasando parámetros (malla, hilos, caso) por línea de comandos y controlando el modo de operación (‘benchmark‘ para tiempo, ‘solution‘ para datos de visualización). Los resultados son centralizados en un archivo ‘resultados$_fdm.csv$‘.

3. Solución mediante el Método de Elementos Finitos (FEM)

3.1. Principio Físico y Discretización

El FEM se fundamenta en un principio físico diferente: la **minimización de la energía del sistema**. La solución a la ecuación de Poisson es aquella que minimiza una "funcional de energía"global. Imagínese una membrana de goma estirada sobre un marco con una forma determinada ($g(x, y)$) y con cargas ($f(x, y)$) aplicadas sobre ella. La forma final que adopta la membrana es aquella que minimiza su energía potencial elástica.

El método procede así:

1. **Discretización del Dominio:** El dominio se descompone en una malla de **elementos triangulares**. Esta flexibilidad es una ventaja clave de FEM.

2. **Aproximación Local:** Dentro de cada triángulo, se asume que la solución es una función muy simple (un plano).

3. **Balance de Energía Local:** Para cada elemento, se deriva una ecuación que describe su contribución a la energía total. Esto se encapsula en una pequeña **matriz de rigidez local** (K_e).

4. **Ensamblaje Global:** Se combinan las contribuciones de todos los elementos en una gran **matriz de rigidez global** (**K**), que representa las interacciones de toda la malla.

El resultado es un sistema de ecuaciones lineales $\mathbf{KU} = \mathbf{F}$. A diferencia de FDM, que "se relaja"hacia la solución, FEM construye un modelo matemático del equilibrio global y lo resuelve de una sola vez.

3.2. Implementación y Automatización (FEM)

La implementación de FEM utiliza la librería **Eigen** para álgebra lineal. El uso de ‘Eigen::SparseMatrix‘ es crucial, ya que aprovecha que la matriz **K** es **dispersa**. Esto reduce drásticamente el consumo de memoria y permite el uso de solucionadores directos altamente optimizados.

3.2.1. Generalización y Automatización

El flujo de trabajo automatizado es idéntico al de FDM, con un Jupyter Notebook dedicado que controla la ejecución y la recolección de datos, asegurando un entorno de prueba consistente para ambos métodos. La paralelización se centra en el bucle de **ensamblaje de la matriz**.

4. Análisis Comparativo de Métodos y Paralelización

Para evaluar y comparar el rendimiento de ambos métodos y sus variantes, se realizaron benchmarks sistemáticos variando el tamaño de la malla y el número de hilos de ejecución.

4.1. Estrategias de Paralelización y su Comparabilidad

Aunque el trabajo computacional dentro de los bucles de FDM (actualización de puntos) y FEM (ensamblaje de elementos) es diferente, las **estrategias de paralelización** aplicadas con OpenMP son conceptualmente análogas. Esto permite una comparación directa y significativa de cómo cada paradigma numérico se beneficia por una estrategia de paralelización específica. La Tabla 1 resume esta analogía.

Cuadro 1: Analogía de las estrategias de paralelización OpenMP aplicadas a FDM y FEM.

Pragma / Versión	**Aplicación en FDM (Jacobi)**	**Aplicación en FEM (Ensamblaje)**
`poisson_serial`	Bucle único que recorre la grilla secuencialmente.	Bucle único que ensambla elementos secuencialmente.
`poisson_parallel`	Bucle ‘for‘ paralelizado que distribuye filas de la grilla entre los hilos.	Bucle ‘for‘ paralelizado que distribuye un conjunto de elementos entre los hilos.
`collapse(2)`	Paraleliza explícitamente los bucles anidados ‘for i, j‘, distribuyendo bloques 2D de la grilla.	Paraleliza los bucles anidados que recorren los cuadriláteros para ensamblar sus elementos.
`schedule`	Prueba cómo la asignación `static` vs. `dynamic` de bloques de la grilla afecta el rendimiento.	Prueba cómo la asignación `static` vs. `dynamic` de elementos afecta el rendimiento.
`sections`	Utiliza paralelismo de tareas para ejecutar la inicialización y el cálculo de la fuente al mismo tiempo.	Menos común, pero podría paralelizar tareas distintas como el ensamblaje de la matriz K y el vector F.
`sincronizacion`	Usa directivas de sincronización fina como `atomic` o `critical` dentro del bucle.	Usa las mismas directivas para proteger el acceso a las estructuras de datos globales durante el ensamblaje.

4.2. Discusión de Resultados

(Aquí es donde interpretas tus gráficas. Ejemplo:)

Figura 1: Comparación de rendimiento entre las versiones seriales de FEM y FDM.

Las pruebas seriales (Figura 1) indican que para mallas pequeñas, el método FDM es más rápido debido a su menor complejidad de inicialización. Sin embargo, a medida que la malla crece, el rendimiento de FEM, apoyado en los solucionadores directos de Eigen, demuestra ser superior.

Figura 2: Análisis de escalabilidad (Speedup) para FEM y FDM en la malla más grande.

En el análisis de escalabilidad (Figura 2), se observa que el método FDM escala casi linealmente, ya que el método de Jacobi es un problema .embarazosamente paralelo". Por otro lado, FEM muestra una buena escalabilidad hasta los 8 hilos, pero su rendimiento disminuye con 16 hilos. Esto se debe a la sección crítica requerida para ensamblar la matriz global, que se convierte en un cuello de botella.

4.3. Análisis de Escalabilidad y Eficiencia

El análisis de Speedup (Aceleración) y Eficiencia Paralela, realizado sobre una malla de 50x50 (ver Figura 3), revela limitaciones de escalabilidad significativas para ambos métodos, FDM y FEM, bajo estas condiciones de prueba. Estos resultados, aunque a primera vista decepcionantes, ofrecen una visión profunda sobre los desafíos de la paralelización.

Figura 3: Análisis comparativo de Speedup (arriba) y Eficiencia (abajo) para las versiones paralelas de FDM y FEM en una malla de 50x50.

4.3.1. Observación 1: El Impacto Crítico del Tamaño del Problema

La observación más importante es que ninguno de los métodos se acerca al Speedup ideal. La razón principal es que una malla de 50x50 es un problema computacionalmente pequeño. En este escenario, el tiempo real de cálculo es muy bajo. Como resultado, el **overhead** —el costo de crear, gestionar y sincronizar los hilos de OpenMP— se convierte en una porción dominante del tiempo total de ejecución. El trabajo útil es tan poco que no justifica el esfuerzo de dividirlo entre muchos hilos.

4.3.2. Observación 2: Contención de Sincronización en FEM

Las implementaciones de FEM (poisson-parallel y poisson-collapse) muestran un comportamiento característico: alcanzan un Speedup máximo de aproximadamente 2x con 2 o 4 hilos, pero luego su rendimiento se estanca e incluso degrada al usar 8 o 16 hilos. Esto es un síntoma clásico de un cuello de botella por contención. En nuestro código FEM, la sección pragma omp critical utilizada para ensamblar la matriz global obliga a que solo un hilo pueda ejecutar esa porción de código a la vez. Con pocos hilos, la probabilidad de que dos hilos lleguen a la sección crítica al mismo tiempo es baja. Sin embargo, con 8 o 16 hilos, la mayoría de ellos pasan más tiempo esperando en "fila"para acceder a la sección crítica que realizando trabajo útil. Esto explica por qué añadir más trabajadores (hilos) no solo no ayuda, sino que empeora el rendimiento.

4.3.3. Observación 3: Dominancia del Overhead en FDM

El método FDM, aunque teóricamente muy escalable (el método de Jacobi es .embarazosamente paralelo"), muestra un Speedup muy bajo, que apenas supera 1x. Su eficiencia, como se ve en la

gráfica inferior, cae drásticamente. Esto confirma que para una malla de 50x50, el costo de lanzar la región paralela, distribuir los bucles con collapse(2) y realizar la reducción para la variable delta es comparable o incluso mayor que el tiempo que se ahorraría en el cálculo. El trabajo por hilo es tan trivialmente pequeño que el overhead de la paralelización consume todos los beneficios potenciales.

5. Conclusiones

Este estudio ha demostrado la viabilidad de automatizar benchmarks de rendimiento para códigos de computación científica complejos utilizando un flujo de trabajo basado en Python y Jupyter Notebooks. Se implementaron y validaron con éxito dos familias de solucionadores para la ecuación de Poisson, basados en FDM y FEM.

Las principales conclusiones son:

- **Eficiencia del Método:** Para problemas de gran escala, el Método de Elementos Finitos, implementado con librerías de álgebra lineal dispersa de alto rendimiento, es superior en tiempo de ejecución al método iterativo de Diferencias Finitas.
- **Escalabilidad Paralela:** El algoritmo de Jacobi en FDM es inherentemente más escalable que la estrategia de ensamblaje con sección crítica de FEM. Para mejorar la escalabilidad de FEM, sería necesario explorar técnicas más avanzadas como el coloreado de malla.
- **Metodología:** La refactorización de los códigos C++ para ser controlados por línea de comandos y la orquestación mediante Jupyter Notebooks demostraron ser una metodología extremadamente poderosa y flexible para la investigación computacional reproducible.
- **Validación Crucial:** La capacidad de generar visualizaciones de la solución física para cada implementación fue un paso indispensable para verificar la correctitud de los resultados numéricos, confirmando que todas las variantes válidas convergen a la misma solución física.

Como trabajo futuro, se podría extender este framework para incluir solucionadores iterativos más avanzados como el método del Gradiente Conjugado, así como explorar técnicas de paralelización distribuida con MPI.

- **La paralelización no es una solución universal:** Para problemas de tamaño insuficiente, el overhead de la gestión de hilos puede superar los beneficios del cálculo en paralelo, resultando en un rendimiento pobre o incluso una ralentización.
- **FEM sufre de contención:** La estrategia de ensamblaje con una única sección crítica es un claro factor limitante para la escalabilidad de FEM. Aunque es más eficiente que FDM a bajos números de hilos en esta prueba, no puede aprovechar un gran número de núcleos.
- **FDM necesita "trabajo pesado":** El potencial de escalabilidad de FDM solo se manifestaría en mallas significativamente más grandes, donde el tiempo de cálculo por punto sea lo suficientemente grande como para hacer que el overhead de OpenMP sea despreciable en comparación.
- **Recomendación para futuros benchmarks:** Para una evaluación justa y representativa del potencial de escalabilidad de estos códigos, es imperativo realizar los mismos análisis en mallas mucho más grandes (e.g., 400x400, 800x800 o superior). Solo en esos escenarios se podrá observar el verdadero comportamiento asintótico del rendimiento paralelo.