Integración por el método de trapecio

Giovanni Ramírez*

Guatemala, 27 de febrero de 2020

Resumen

El Método de Trapecios se usa para calcular una integral definida. El método de trapecios es una aproximación al cálculo del área bajo la curva mediante el uso de un número dado de trapecios que cubren el área buscada. En este trabajo muestra la convergencia del valor aproximado del área en función del número de trapecios usados. También se muestra que el error de aproximación decrece con el cubo del número de trapecios usados en la aproximación. Además, se presentan varias soluciones, todas implementadas en FORTRAN, unas para entender el método en un algoritmo secuencial usando funciones y módulos y las otras usando algoritmos paralelos donde se muestra la mejora en el tiempo necesario para encontrar una solución con un número dado de trapecios.

1. Introducción

En el análisis numérico, el método del trapecio o regla del trapecio es una técnica de aproximación para integrales definidas de la forma

$$\int_a^b f(x)dx,$$

que representa el área bajo la curva $f(x)$ y limitada por el eje x y las rectas $y = a$ y $y = b$. El método aproxima el área colocando trapecios cuya área se obtiene con

$$A = h\left(\frac{base_1 + base_2}{2}\right),$$

es decir, la semisuma de sus bases, que son paralelas, multiplicadas por la altura.

*Escuela de Ciencias Físicas y Matemáticas, Universidad de San Carlos de Guatemala.

Este trabajo presenta la implementación del método de trapecios usando FORTRAN. Se realizaron varias implementaciones para aprovechar las propiedades de modularización de FORTRAN. Además, se hizo una implementación con programación en paralelo usando la biblioteca openMPI. Todos los códigos están disponibles como material suplementario y son compatibles con cualquier implementación de compilador FORTRAN que siga el estándar FORTRAN95.

Para todas las implementaciones se muestran comparaciones de rendimiento, asumiendo que la complejidad algorítmica está directamente relacionada con el tiempo de cómputo y la cantidad de memoria RAM usada.

Este trabajo está organizado de la siguiente manera, en la sección 1 se presenta el formalismo del método, en la sección 3 se discute los detalles de las diferentes formas de implementación del método y luego se muestran los resultados de las pruebas de rendimiento en la sección 4. Finalmente en la sección 5 se discuten las conclusiones motivadas por la definición del problema.

2. El método del trapecio

Según Cohen[1], el método puede deducirse como uno de los casos de las fórmulas de Newton-Cotes, donde una función puede aproximarse usando polinomios. De esta manera, la regla del trapecio representa el tercer caso después de la regla de la constante y de la regla del rectángulo. Por otro lado, Burden[2] el método también aparece como una aplicación del polinomio lineal de Lagrange.

Considerando una función f cualquiera, como la mostrada en el panel izquierdo de la figura 1, el método del trapecio usando un sólo trapecio aproximaría el área tomando como bases $f(a)$ y $f(b)$ y como altura $(b-a)$, de modo que

$$\int_a^b f(x)dx \approx (b-a)\left(\frac{f(a)+f(b)}{2}\right).$$

En el panel izquierdo de la figura 1 puede verse que el área sombreada del trapecio no cubre exactamente el área buscada. Una forma de mejorar la aproximación es usando más trapecios. Consideremos el conjunto $\{x_k\}$ como una partición uniforme del intervalo $[a,b]$ tal que

$$a = x_0 < x_1 < \cdots < x_{N-1} < x_N = b,$$

así que el k-ésimo intervalo tendrá una longitud $\Delta x_k = x_k - x_{k-1}$. Ahora podemos aproximar el área bajo la curva usando N trapecios cuyas bases

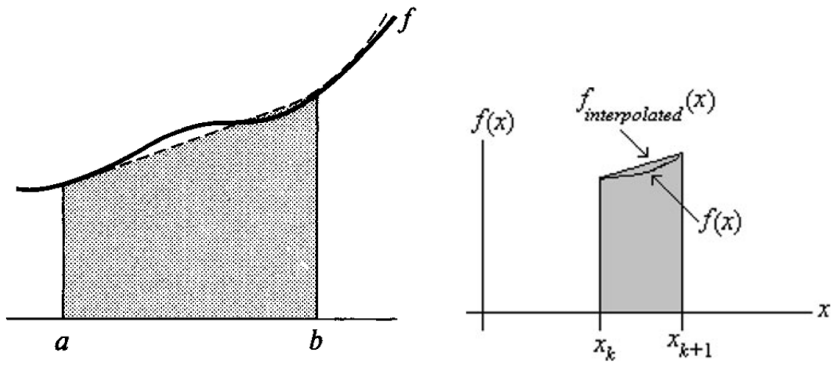

Figura 1: Método del trapecio para aproximar el área bajo la curva. A la izquierda mediante el área de un trapecio. A la derecha se muestra el detalle del $(k+1)$-ésimo trapecio cuando se usan N trapecios en la aproximación, aquí se muestra con énfasis el error de aproximación. Fuente: la figura de la izquierda aparece en Golub y Ortega [3], cap 5. La figura de la derecha aparece en Cohen [1], cap. 4.

se ubican en las posiciones $\{x_k\}$ y con una altura $\Delta x_k = \Delta x$ constante, entonces

$$\int_a^b f(x)dx \approx \Delta x \left[\frac{f(x_0)+f(x_1)}{2} + \frac{f(x_1)+f(x_2)}{2} + \cdots + \frac{f(x_{N-1})+f(x_N)}{2} \right],$$

o de una forma reducida, usando $\Delta x = (b-a)/N$

$$\int_a^b f(x)dx \approx \left(\frac{b-a}{N}\right) \sum_{k=1}^{N} \left[\frac{f(x_{k-1})+f(x_k)}{2} \right]. \tag{1}$$

Aunque usemos $N \gg 1$ trapecios, el $(k+1)$-ésimo trapecio tendrá un error de aproximación como el que se muestra en el panel derecho de la figura 1. Para analizar el error de aproximación resulta más sencillo seguir la deducción del método mediante el polinomio lineal de Lagrange usando sólo un trapecio

$$P_1(x) = \left(\frac{x-x_1}{x_0-x_1}\right) f(x_0) + \left(\frac{x-x_0}{x_1-x_0}\right) f(x_1),$$

donde se toma $x_0 = a$, $x_1 = b$. El área bajo la curva se puede calcular exactamente usando

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} dx \left[\left(\frac{x - x_1}{x_0 - x_1} \right) f(x_0) + \left(\frac{x - x_0}{x_1 - x_0} \right) f(x_1) \right] + \frac{1}{2} \int_{x_0}^{x_1} dx f''(\xi(x))(x - x_0)(x - x_1),$$

donde el segundo término representa el error de aproximación y la función $\xi(x) \in (x_0, x_1)$. El término de error se puede simplificar usando el Teorema del valor medio ponderado para integrales [2] ya que el producto $(x-x_0)(x-x_1)$ no cambia de signo en $[x_0, x_1]$

$$\begin{aligned} \int_{x_0}^{x_1} dx f''(\xi(x))(x - x_0)(x - x_1) &= f''(\xi) \int_{x_0}^{x_1} dx (x - x_0)(x - x_1) \\ &= f''(\xi) \left[\frac{x^3}{3} - \frac{(x_1 + x_0)}{2} x^2 + x_0 x_1 x \right]_{x_0}^{x_1} \\ &= -\frac{(x_1 - x_0)^3}{6} f''(\xi), \end{aligned}$$

donde si tomamos $h = (x_1 - x_0)$ tenemos el error de aproximación para el método del trapecio

$$-\frac{h^3}{12} f''(\xi), \tag{2}$$

que incluye la segunda derivada de $f(x)$ que en principio debemos conocer. El método es exacto, es decir con error cero, si aplicamos el método a una función $f(x)$ polinomial de grado uno o menor. Para cualquier otra función $f(x)$, sólo podemos establecer una cota para el error máximo de aproximación calculando el punto en el intervalo $[a, b]$ donde $f''(\xi)$ sea máxima.

Si se usan N trapecios, entonces

$$h = \frac{b - a}{N},$$

es decir, ya que el error decrece con el cubo del número de trapecios que se usen para la aproximación. Existen otros métodos cuyo error decrece de una forma más rápida, por ejemplo los métodos de Simpson que también pueden derivarse usando las fórmulas de Newton-Cotes[1] y el cálculo del error del método también se puede derivar usando los polinomios de Lagrange[2].

3. Implementación

Se realizaron implementaciones secuenciales para mostrar distintas características como el uso de funciones, subrutinas y módulos. También se hicieron dos implementaciones con programación en paralelo para mostrar distintos detalles del envío de mensajes entre los distintos procesos.

Las implementaciones en paralelo están diseñadas de modo que pueden usar cualquier número de procesos paralelos. El número de trapecios es igual al número de procesos paralelos que se asignan en tiempo de ejecución.

Todas las implementaciones leen los parámetros de un archivo de texto de nombre `parametros.conf` con la siguiente estructura:

- la primera línea debe ser un comentario, este se descartará;
- la segunda línea es para el límite de integración a, se considerará como un valor real;
- la tercera línea es para el límite de integración b, al igual que el caso anterior, se considerará como un valor real;
- la cuarta línea es para el número de trapecios con los que se hará la aproximación;
- la quinta línea es opcional, es para la implementación modular donde es posible usar la función de Bessel y este número representa el orden de la función, si esta línea no está presente, se asume el orden cero.

Todas las implementaciones escriben sus salidas en la salida estándar porque se diseñaron para completar su tarea rápidamente. Para guardar las salidas es mejor redirigir la salida estándar.

3.1. Implementación secuencial

En total se realizaron tres implementaciones secuenciales: (a) usando funciones en un código monolítico, (b) usando subrutinas en un código monolítico, y una forma más ordenada (c) usando módulos.

Los códigos monolíticos son `trapecioS.f90` y `trapecioF.f90` para probar el uso de una subrutina, y una función respectivamente, para calcular el valor de la función $f(x)$ cada vez que sea necesario. En el caso de la implementación con funciones el resultado de la función se regresa al ámbito local por asignación directa

$$S = f(x).$$

Sin embargo, en el caso de la implementación con subrutinas, la llamada a la subrutina se hace incluyendo una variable temporal que se define como `INTENT(OUT)` en el ámbito de la subrutina

$$\begin{aligned} &\text{CALL } f(x, y) \\ &S = y \end{aligned}$$

La implementación sigue el siguiente algoritmo

Algoritmo 1. Implementación secuencial monolítica. Entrada: la función $f(x)$; los límites de integración a, b y el número de trapecios n. Salida: el valor del área bajo la curva que se guardará en la variable S.

1. Inicio.
2. Definir las variables del problema y las variables auxiliares.
3. Leer los parámetros del archivo. Parar en caso de error.
4. Calcular $h = (b - a)/n$.
5. Calcular los valores en los extremos $f(a)$ y $f(b)$.
6. Establecer la variable $suma = f(a) + f(b)$.
7. Hacer para $1 \leq i \leq n - 1$.
   7.1 Agregar las sumas de las bases del i-ésimo trapecio a $suma$.
8. Calcular el área, $S = suma * h/2$.
9. Fin.

El código modular es `trapecioM.f90`, este código requiere del módulo *types* que está disponible en `types.f90` y del módulo *pool* que está disponible en `pool.f90`. El módulo *types* contiene definiciones para simplificar la modificación de precisión de las variables reales o la forma de representación de las variables enteras. El módulo *pool* contiene los prototipos para funciones polinomiales que están implementadas en `funciones.f90`, prototipos para la función factorial implementada en `factorial.f90` y el prototipo de la función de Bessel $J_n(x)$ que está implementada en `besselJ.f90`.

La implementación sigue el siguiente algoritmo

Algoritmo 2. Implementación secuencial modular. Entrada: la función $f(x)$; los límites de integración a, b y el número de trapecios n. Salida: el valor del área bajo la curva que se guardará en la variable S.

1. Inicio.
2. Incluir las definiciones de los módulos *types* y *pool*.
3. Definir las variables del problema y las variables auxiliares.
4. Leer los parámetros del archivo. Parar en caso de error.
5. Calcular $h = (b - a)/n$.
6. Calcular los valores en los extremos $f(a)$ y $f(b)$.
7. Establecer la variable $suma = f(a) + f(b)$.
8. Hacer para $1 \leq i \leq n - 1$.
   8.1 Agregar las sumas de las bases del i-ésimo trapecio a *suma*.
9. Calcular el área, $S = suma * h/2$.
10. Fin.

La secuencia para compilar todos los módulos es la siguiente

1. `gfortran -Wall -pedantic -std=f95 -c -o types.o types.f90`
2. `gfortran -Wall -pedantic -std=f95 -c -o funciones.o funciones.f90`
3. `gfortran -Wall -pedantic -std=f95 -c -o factorial.o factorial.f90`
4. `gfortran -Wall -pedantic -std=f95 -c -o besselJ.o besselJ.f90`
5. `gfortran -Wall -pedantic -std=f95 -c -o pool.o pool.f90`
6. `gfortran -Wall -pedantic -std=f95 -c -o trapecioM.o trapecioM.f90`

después de eso se enlazan todos los objetos en un archivo ejecutable usando
`gfortran -o trapecioM.x types.o pool.o funciones.o factorial.o besselJ.o trapecioM.o`

3.2. Implementación en paralelo

Las dos implementaciones en paralelo siguen el mismo algoritmo solo que ahora, se toma el número de procesos paralelos disponibles en tiempo de ejecución y se eligen tantos trapecios como proceso paralelos.

Existen dos implementaciones, una en la que se hace una implementación *single instruction stream multiple data stream*, SIMD según la taxonomía de Flynn, sin intercambio de mensajes entre los procesos paralelos. Cada proceso paralelo escribe el cálculo del subtotal de área que le corresponde en la salida estándar.

La segunda implementación es una mejora de la anterior, inicia con la implementación SIMD y después hace una implementación *multiple instruction stream multiple data stream*, MIMD según la taxonomía de Flynn, donde el proceso *rank 0* recibirá cada una de los subtotales calculados en la parte SIMD mientras que los otros procesos enviarán su subtotal al proceso *rank 0*. Cada proceso paralelo escribe el cálculo del subtotal de área que le corresponde y *rank 0* escribe el área total. Los mensajes se envían y reciben usando el esquema de comunicación básico con bloqueo de openMPI.

La implementación SIMD sigue algoritmo 3 que está en la página y como se explicó antes, no se calcula el área total. Esta implementación se hace únicamente para mostrar un ejemplo de un algoritmo paralelo SIMD. Es importante remarcar que en este esquema, todos los procesos harán la misma tarea, calcular el área de un trapecio, pero con diferentes conjuntos de datos, a cada proceso le corresponde un subintervalo distinto. La división de estos subintervalos se hace de forma que cada proceso hará el cálculo del límite inferior y del límite superior de su subintervalo

$$lim_{INF} = a + REAL(rank) * (b - a)/REAL(size),$$
$$lim_{SUP} = a + REAL(rank + 1) * (b - a)/REAL(size),$$

así que el altura del trapecio será $h = lim_{SUP} - lim_{INF}$ y las bases se calculan con $f(lim_{INF})$ y $f(lim_{SUP})$, de modo que el área de cada proceso será

$$A = h\left(\frac{f(lim_{INF}) + f(lim_{SUP})}{2}\right).$$

El área total no es calculada en esta implementación,

Algoritmo 3. Implementación paralela SIMD. Entrada: la función $f(x)$; los límites de integración a, b y el número de trapecios n. Salida: el valor de los subtotales de área calculados por cada proceso.

1. Inicio.
2. Incluir las definiciones de los módulos *types* y *pool*.
3. Definir las variables del problema, las variables para MPI y las variables auxiliares.
4. Iniciar el entorno MPI. Parar en caso de error.
5. Leer los datos de *rank* y *size*. Parar en caso de error.
6. Leer los parámetros del archivo. Parar en caso de error.
7. Calcular el límite inferior, lim_{INF}, y el superior, lim_{SUP}, de para los subintervalos.
8. Calcular el área del trapecio correspondiente.
9. Imprimir el área correspondiente o subtotal.
10. Terminar el entorno MPI.
11. Fin.

La segunda implementación paralela sigue el esquema SIMD-MIMD. Es importante notar que el esquema de SIMD únicamente separa los datos para que cada nodo haga la misma tarea y que el el esquema MIMD separa las tareas de envío y recepción. El *rank 0* se encargará de recibir todos los mensajes y sumarlos para presentar el área total calculada. Esta implementación se describe en el algoritmo 4.

Algoritmo 4. Implementación paralela SIMD+MIMD. Entrada: la función $f(x)$; los límites de integración a, b y el número de trapecios n. Salida: el valor de los subtotales de área calculados por cada proceso y el área total bajo la curva.

1. Inicio.
2. Incluir las definiciones de los módulos *types* y *pool*.
3. Definir las variables del problema, las variables para MPI y las variables auxiliares.
4. Iniciar el entorno MPI. Parar en caso de error.

5. Leer los datos de $rank$ y $size$. Parar en caso de error.
6. Leer los parámetros del archivo. Parar en caso de error.
7. Calcular el límite inferior, lim_{INF}, y el superior, lim_{SUP}, de para los subintervalos.
8. Calcular el área del trapecio correspondiente.
9. Imprimir el área correspondiente o subtotal.
10. Si $rank = 0$ hacer
    - 10.1 Establecer la variable $Total = Area$
    - 10.2 Para $1 < i < size - 1$ hacer
        - (a) Recibir el subtotal del $rank = i$
        - (b) Calcular $Total = Total + subtotal$

        Si no, enviar a $rank = 0$ el subtotal de área calculado
    - 10.3 Imprimir el área $Total$
11. Terminar el entorno MPI
12. Fin

---

4. Pruebas de rendimiento

Las pruebas de rendimiento para los códigos secuenciales se hacen sólo como muestra. Todas las pruebas fueron realizadas en el clúster Euclides de la ECFM, las pruebas secuenciales se corrieron en nodo maestro y las pruebas secuenciales usaron otros nodos según se detalla en cada una de las pruebas.

El clúster Euclides tiene un nodo maestro con un procesador Intel Pentium 4 CPU a 3.40GHz y 2GB de memoria RAM. También hay 4 nodos de cálculo con procesadores Pentium Dual-Core E5700 a 3.00GHz y 2GB de memoria RAM. Para hacer mediciones de tiempo, memoria usada y carga del procesador se usa el l

En el panel izquierdo de la figura 2 se muestran los recursos usados por el algoritmo secuencial usando subrutinas para el cálculo del área bajo la curva de una función polinomial en función del número de trapecios usados. En el

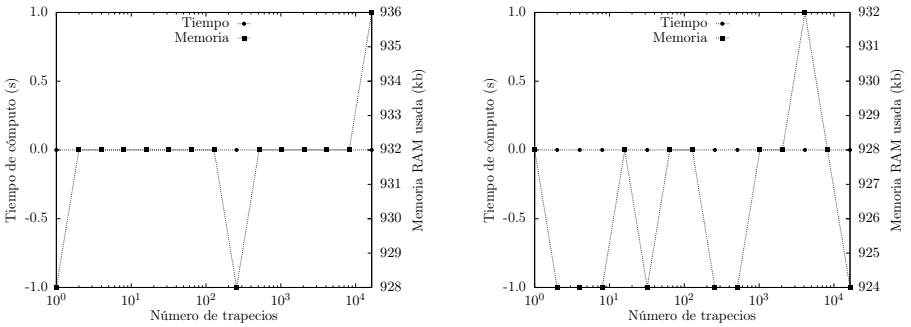

Figura 2: Recursos computacionales usados por el algoritmo secuencial. A la izquierda usando subrutinas y a la derecha usando funciones. En ambos casos la escala vertical de la derecha muestra la memoria RAM y la escala vertical de la izquierda muestra el tiempo usado para calcular el área bajo la curva en función del número de trapecios. Las líneas son guías visuales y no ajustes. Fuente: elaboración propia.

panel derecho de la figura 2 se muestran los recursos usados por el algoritmo secuencial usando subrutinas para el cálculo del área bajo la curva de una función polinomial en función del número de trapecios usados. En ambos casos, el tiempo de cómputo es más pequeño que la granularidad del sistema por lo que no hace falta hacer un ajuste. En cuanto a la memoria RAM usada, en ambos casos está limitada a un rango muy pequeño de tamaño 8kb y no es necesario hacer un ajuste.

En la figura 3 se muestra una comparación de los tiempos usados por la implementación secuencial y por la implementación paralela para calcular el área bajo la curva de la función de Bessel $J_n(x)$. La definición del problema pedía una implementación en paralelo donde cada proceso paralelo debía tomar uno y sólo un trapecio, entonces sólo se pudo obtener datos hasta 10 trapecios ya que Euclides sólo puede tomar 10 procesos paralelos. El tiempo que se mide en la implementación en paralelo incluye: el tiempo de la parte SIMD cuando se hace la separación de los intervalos que integrará cada proces; y el tiempo de la parte MIMD cuando cada proceso envía su subtotal al proceso de *rank 0*.

En el inserto de la figura 3 se muestra el consumo de la memoria RAM que hace el proceso secuencial, la cantidad de memoria que usa esta implementación no varía considerablemente de las opciones de implementación por funciones o por subrutinas presentadas anteriormente. No se están midiendo los recursos consumidos por cada hilo puesto que sólo se quieren comparar los

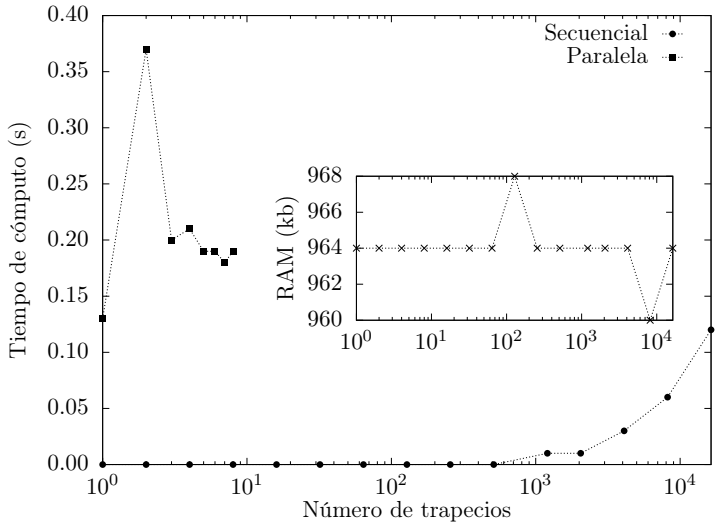

Figura 3: Comparación de los tiempos usados en la implementación secuencial y la implementación paralela. En el inserto se muestra el consumo de memoria RAM en la implementación secuencial. En cada caso, las líneas son guías visuales. Fuente: elaboración propia.

tiempos totales usados para el cálculo. Para medir estos tiempos habría que usar las funciones de openMPI para elaborar el perfil completo del programa implementado.

5. Conclusiones

Como parte de la definición del problema da la pregunta de cuál de las dos implementaciones, usando subrutinas o usando funciones, tiene menos error en la aproximación para una función polinomial. En la figura 4 se muestra el error relativo de la aproximación donde se puede ver que el error es independiente de los dos métodos usados. El error disminuye hasta llegar al orden de 10^3 donde presenta oscilaciones. Estas oscilaciones se deben a que las oscilaciones que pueda tener segunda derivada de la función que se está integrando empiezan a ser importantes y no se deben a la precisión usada para el cálculo.

La segunda pregunta de la definición del problema es verificar si se incrementa el error al enviar los mensajes de OpenMPI. En la figura 5 se muestran los valores del área bajo la curva obtenidos con los métodos de implementa-

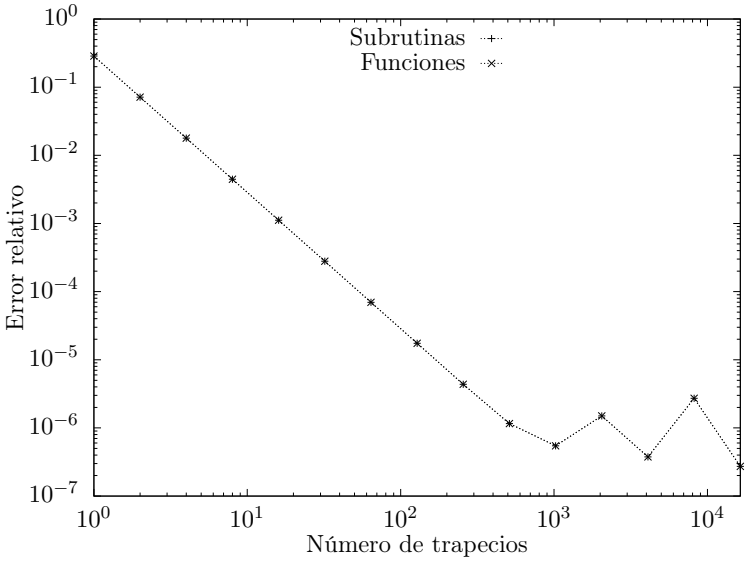

Figura 4: Error relativo de la aproximación en función del número de trapecios. Fuente: elaboración propia.

ción en función del número de trapecios. En el inserto muestra en detalle la diferencia entre el valor obtenido por la implementación secuencial menos el valor obtenido por la implementación paralela. La diferencia es menor que 10^{-3} por lo que habría que explorar si usando un esquema distinto para la implementación paralela puede usar más trapecios para comprobar este comportamiento.

La tercera pregunta de la definición es proponer un esquema distinto para la implementación en paralelo. Una modificación que permitiría incluir más trapecios para la aproximación es dividir el intervalo $[a, b]$ en tantos subintervalos como procesos paralelos haya. Después, cada subintervalo es asignado a cada proceso paralelo para encontrar la aproximación de su subtotal de área. Esta aproximación puede hacerse mediante un número de trapecios dado.

Cada proceso podría hacer el mismo número de trapecios, sin embargo si se conoce la forma de la función $f(x)$ que se quiere integrar es posible implementar un código adaptivo de modo que si $f'(x)$ no varía mucho en un subintervalo, entonces se pueden usar pocos trapecios. Ahora, si $f'(x)$ varía mucho en un subintervalo, entonces en ese subintervalo se debe incrementar el número de trapecios.

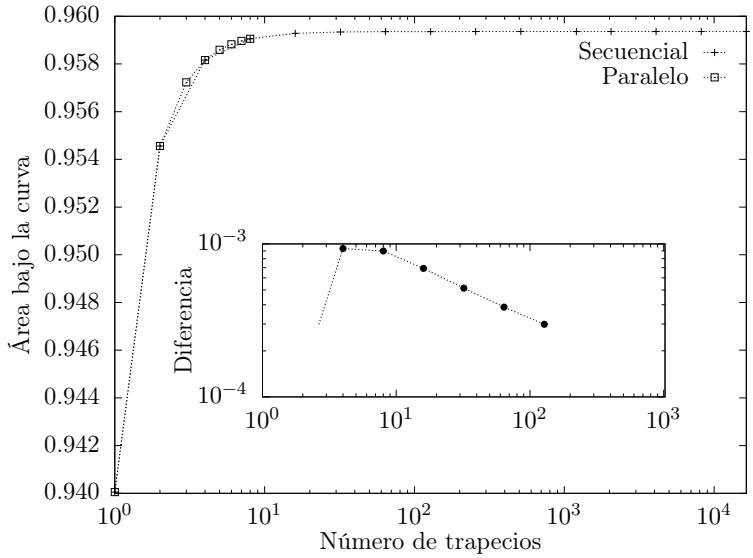

Figura 5: Área bajo la curva obtenido medidante las dos implementaciones, secuencial y paralelo en función del número de trapecios. En el inserto está la diferencia del valor obtenido en la implementación secuencial menos el valor obtenido en la implementación paralela. Fuente: elaboración propia.

Agradecimientos

Se agradece a la ECFM-USAC por el uso del clúster Euclides donde se realizaron las pruebas de rendimiento reportadas en este trabajo.

Referencias

[1] H. Cohen. *Numerical Approximation Methods:* $\Pi \approx 355/113$. Springer, 2011.

[2] R.L Burden, D.J. Faires, and A.M. Burden. *Numerical Analysis*. CENGAGE Learning, 2016.

[3] G.H. Golub and J.M. Ortega. *Scientific Computing: An Introduction with Parallel Computing*. Academic Press, Inc., 1993.