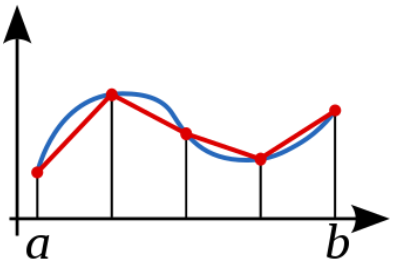
Método de Regla del Trapecio



La técnica de integración por trapecios es una forma de aproximar una integral definida utilizando cierto número de trapecios. El valor de la integral representa el área de la región delimitada por la gráfica comprendida entre los trapecios cuyos vértices sobre la función son equidistantes.

El error en el método de la regla del trapecio está relacionado inversamente con el número de trapecios: a más trapecios, más precisión, por lo que menos error.

Para probar este método, se utiliza la siguiente función que devuelve el cálculo del número PI:

$$\pi = \int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}}$$

Sin embargo, no se programa directamente la solución con el método de trapceios, sino que se codifica una versión modificada compatible con MPI, para poder analizar su rendimiento en función del número de procesadores disponibles. Dicho esto, el código resultante es el siguiente:

```
double a = -1.0, b = 1.0;
double h = fabs(b-a) / (double) (THROWS*numtasks), x, sum = 0.0;

for (int i=(THROWS*taskid)+1; i < (THROWS*(taskid+1)); i++) {
    x = a + i*h;
    sum += f(x);
}

if (taskid == MASTER) {
    double sumRecv;
    MPI_Status status;
    for (int i = 1; i < numtasks; i++) {
        MPI_Recv(&sumRecv, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 1,
MPI_COMM_WORLD, &status);
        sum += sumRecv;
    }

    double pi = (h/2.0) * (f(a+0.0001) + f(b-0.0001) + 2.0*sum);
} else {
    MPI_Send(&sum, 1, MPI_DOUBLE, MASTER, 1, MPI_COMM_WORLD);
}
```

En este caso, `THROWS` es una constante que define el número de números aleatorios a generar, `taskid` es el número de tarea (habiendo tantas tareas como procesadores), y `f(x)` evalúa para esta función de PI el resultado de x. Cabe destacar dos aspectos de este método:

- El primero "*throw*" ocurre en 1, y no en 0, ya que en este caso la función evaluada sobre el límite inferior tiende a infinito.
- Al final, tras recolectar todos los valores, dado que ambos límites de la función tienden a infinito, se aplica un *offset* para evitar este error de cálculo, a coste de incrementar ligeramente el error de PI.

Una vez obtenido el valor de PI, se calcula el error con respecto a la siguiente referencia:

$$3.14159265358979323846264338327950288419716939937510582097494 46$$

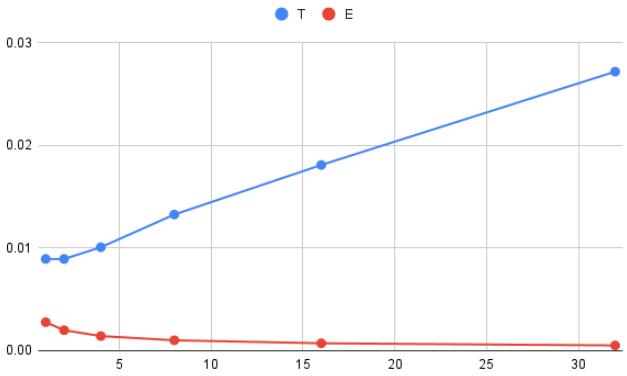
Y se define la calidad del resultado como (siendo T el tiempo de cómputo y E el error):

$$C = \frac{1}{T \cdot E}$$

Se procede entonces a realizar el cómputo con 1, 2, 4, 8, 16 y 32 procesadores. Para cada una de estas opciones, se calcula 5 veces para evitar posibles ruidos, y se toma el valor en la mediana (no se usa la media para así descartar los valores muy desviados debido al ruido).

El resultado se puede ver en la siguiente tabla:

# Cores	T	E	C
1	0.008944	0.002779	40228.62882
2	0.008939	0.001996	56059.65949
4	0.010086	0.001427	69496.45925
8	0.013270	0.001017	74094.77142
16	0.018103	0.000724	76324.34070
32	0.027203	0.000515	71446.78964



Ya a primera vista, se puede ver como hay mejor calidad que en el método de Montecarlo, y además se puede ver como la tendencia de tiempo y error es algo más constante y sin desviaciones extrañas. En cuanto al tiempo, tal como es de esperar, a mayor número de procesadores mayor es el tiempo debido al coste de las comunicaciones requeridas. Y en cuanto al error, cuantos más procesadores menor error hay, pero a cambio de un mayor tiempo de cómputo. Se puede ver como el valor óptimo en cuanto a la relación tiempo-error es el de 16 procesadores, ya que el incremento de tiempo para el siguiente valor de procesadores no compensa para el error reducido.