**Taller 4**

Se ha encomendado a los estudiantes de la asignatura de fundamentos de computación de alto desempeño realizar una mejora en el cálculo del algoritmo de PI, utilizando programación paralela

Pseudocódigo

|  |
| --- |
| Proceso ValorDePi      pi <- 0;      Escribir Sin Saltar "Ingresa el valor de n:";      Leer n;      Para i<-1 Hasta n Con Paso 1 Hacer          Escribir "PROCESO ", i;          pi <- pi+4.0/(i\*2-1);          Escribir "";      FinPara      Escribir "Valor de pi: ", pi;  FinProceso |

Para ello debe escribir la rutina en cpp para la solución de PI y registrar los siguientes casos:

1) Calcular el valor de PI para una precisión con el valor n = 1000, determinando el tiempo de procesamiento normal y luego compararlo con el procesamiento paralelo

2) Calcular el valor de PI para una precisión con el valor n =10000, determinando el tiempo de procesamiento normal y luego compararlo con el procesamiento paralelo

3) Calcular el valor de PI para una precisión con el valor n = 3, ¿es mejor el procesamiento paralelo?

Formato de Entrega: Word con Pantallazos de código, explicación de código y resultados, en conjunto con las conclusiones.

**Explicación del algoritmo**

El algoritmo de la serie de potencias de PI se refiere a una serie matemática convergente cuyo límite tiende a PI. Existen varias series de potencias utilizadas para calcular PI, pero una de las más conocidas y simples es la serie de Leibniz, que es una serie alternante que converge a PI/4. La serie de Leibniz se define de la siguiente manera:

π/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + ...

Esta serie converge lentamente a PI, lo que significa que se necesitan muchos términos para obtener una aproximación precisa de PI. Sin embargo, existen otras series que convergen más rápidamente, como la serie de Bailey-Borwein-Plouffe (BBP) y la fórmula de Chudnovsky. Estas series permiten calcular dígitos específicos de PI sin calcular los dígitos previos y se utilizan en cálculos de alta precisión de PI.

A continuación, se muestra un ejemplo de código en C++ para calcular PI utilizando la serie de Leibniz:

|  |
| --- |
| #include <iostream>  #include <cmath>  const int num\_terms = 1000000;  int main()  {  double pi = 0.0;  for (int i = 0; i < num\_terms; i++)  { double term = 1.0 / (2 \* i + 1); if (i % 2 == 0) { pi += term; } else { pi -= term; } }  pi \*= 4;  std::cout << "La aproximación de PI utilizando la serie de Leibniz con " << num\_terms << " términos es: " << pi << std::endl;  return 0;  } |

Este código calcula una aproximación de PI utilizando la serie de Leibniz con un número fijo de términos. La convergencia de la serie de Leibniz es lenta, y se necesitan un gran número de términos para obtener una aproximación precisa de PI. Para algoritmos más rápidos y precisos, considera investigar la serie BBP o la fórmula de Chudnovsky.

En este ejemplo, utilizaré OpenMP para paralelizar un bucle for que realiza la suma de los elementos de un vector. El código en C++ con OpenMP es el siguiente:

|  |
| --- |
| #include <iostream>  #include <vector>  #include <omp.h>  int main() {  // Definir el tamaño del vector y el número de hilos const int N = 1000;  const int num\_threads = 4; // Crear e inicializar un vector con valores std::vector<int> vec(N);  for (int i = 0; i < N; i++) { vec[i] = i + 1; } int sum = 0;  // Establecer el número de hilos a utilizar omp\_set\_num\_threads(num\_threads);  // Paralelizar el bucle for utilizando pragma omp parallel for  #pragma omp parallel for reduction(+:sum)  for (int i = 0; i < N; i++) { sum += vec[i]; }  // Mostrar el resultado  std::cout << "La suma de los elementos del vector es: " << sum << std::endl;  return 0;  } |

Este código crea un vector de tamaño N e inicializa sus elementos. Luego, paraleliza la suma de los elementos utilizando OpenMP. La directiva #pragma omp parallel for reduction(+:sum) permite paralelizar el bucle for y aplicar una reducción para calcular la suma total de forma eficiente.

Recuerda que para compilar este código con OpenMP, necesitas habilitar el soporte para OpenMP en tu compilador, por ejemplo, utilizando la opción -fopenmp con g++:

|  |
| --- |
| g++ -fopenmp main.cpp -o main |

Y luego, ejecuta el programa:

|  |
| --- |
| ./main |