

Universidad del Valle de Guatemala

Curso: Programación de Microprocesadores

Docente: Kimberly Barrera

Sección: 10

Carné: 19943, Orlando Cabrera

Carné: 19707, José Javier Hurtarte

Carné: 19195, Pablo Méndez

Carné 191025, Diana Corado



Bitácora

(Proyecto 3)

Guatemala 26 de octubre de 2020

ÍNDICE

MARCO TEÓRICO.....	3
Programación paralela en ramas de la física y matemáticas.....	3
Aplicación de OpenMP para el cálculo de valores numéricos de precisión.	3
SOLUCIÓN MATEMÁTICA: DEBE INCLUIR LA OPERACIÓN EN CADA INCISO –SIN PROGRAMACIÓN.	4
DIAGRAMAS DE FLUJO/ALGORITMO DESCRIPTIVO PROGRAMA	5
Algoritmo descriptivo del valor de euler:.....	5
Algoritmo descriptivo para el cálculo de número de convergencia de la serie $n = 1 \infty 1n(n + 1)$	6
Diagrama de flujo para el cálculo de número de convergencia de la serie $n = 1 \infty 1 - 1nn$:.....	6
PROGRAMA SOLUCIÓN	7
Serie de Euler: $n = 0 \infty 1n!$	7
Serie 2: $n = 1 \infty 1n(n + 1)$	9
Serie 3: $n = 1 \infty 1 - 1nn$	10
REFERENCIAS	11

MARCO TEÓRICO

Programación paralela en ramas de la física y matemáticas.

El paralelismo dentro de la informática se refiere al procesamiento simultáneo de diversas instrucciones, o cálculos matemáticos. A lo largo de los años en las ramas de la matemática y física se ha buscado la eficiencia y rapidez en la resolución de problemas, por lo que se han diseñado diversos métodos para encontrar la respuesta a un mismo problema.(omni.sci,s.f.)

Sin embargo, aun así, existen operaciones que se vuelven demasiado largas al resolverlas manualmente (a papel y lápiz), por lo que se implementa el apoyo de las computadoras para la resolución de dichas operaciones. Empero, dependiendo de la complejidad de la operación, los ordenadores antiguos podían tardar desde segundos hasta días en devolver el resultado. (Lopategui,s.f.)

Por medio del avance de la tecnología a través de los años se han encontrado alternativas que han permitido disminuir el tiempo de ejecución de instrucciones. Ya que si bien, el mejorar la velocidad y capacidad de procesamiento de los microprocesadores ha sido un gran avance, la implementación de paralelismo ha contribuido a disminuir significativamente el tiempo de ejecución. (Gasca-Hurtado & Machuca-Villegas, 2018, p. 10)

Las computadoras, calculadoras e incluso teléfonos celulares permiten realizar en cuestión de segundos cálculos que para los seres humanos tomaría días. Actualmente gracias al paralelismo, es posible dividir las instrucciones en pequeñas tareas, permitiendo así aumentar la rapidez en la ejecución de instrucciones y la eficiencia de un programa computacional.

Aplicación de OpenMP para el cálculo de valores numéricos de precisión.

A pesar de que las computadoras puedan realizar procesos muy complicados, tanto que para una persona sería imposible realizar el mismo proceso en el mismo tiempo en que lo puede realizar una computadora, esto no significa que las computadoras sean muy inteligentes. Las computadoras necesitan a una persona que les diga que instrucciones seguir a partir de unos datos, los cuales son proporcionados por los usuarios. Las computadoras suelen trabajar con algunas operaciones aritméticas básicas, tales como las sumas y restas. A partir de estas operaciones las computadoras pueden realizar procesos mucho más complicados (UaesSharkComputing, sf).

A partir de las operaciones básicas que las computadoras pueden realizar, las personas han sido capaces de realizar los cálculos de algunas operaciones muy complicadas. Un claro ejemplo de esto es el número π (pi), el cual es muy complicado de calcular si no se tiene una computadora. Para la realización de estos cálculos, las computadoras necesitan realizar un número muy grande de operaciones (sumatorias) para poder determinar un valor muy cercano al valor real de estos valores (las computadoras no pueden calcular el valor exacto, ya que, tomando el ejemplo de pi, la computadora necesitaría determinar una cantidad infinita de decimales, lo cual es imposible ya que requeriría de realizar un número infinito de cálculos) (Alfonso I, 06/2009). Es aquí en donde entra la implementación de OpenMp para calcular estos valores.

Como se mencionó anteriormente, para el cálculo de estos valores se necesita realizar muchas operaciones y ya que estas operaciones no requieren mucho procesamiento para el CPU, estos se pueden realizar utilizando programación paralela, más en específico, utilizar OpenMp con threads para realizar cada operación, en donde se realiza la sumatoria en conjunto con todos los threads y que

estos puedan compartir sus resultados de una forma sencilla, así reduciendo el tiempo de ejecución y obteniendo el valor de dicho cálculo.

SOLUCIÓN MATEMÁTICA: DEBE INCLUIR LA OPERACIÓN EN CADA INCISO –SIN PROGRAMACIÓN.

Serie de Euler:

$$\sum_{n=0}^{\infty} \frac{1}{n!}$$

$$\lim_{n \rightarrow \infty} \frac{\frac{1}{(n+1)!}}{\frac{1}{n!}}$$

$$\lim_{n \rightarrow \infty} \frac{\frac{1}{(n+1)n!}}{\frac{1}{n!}}$$

$$\lim_{n \rightarrow \infty} \frac{1}{n+1} = 0$$

∴ La serie converge

Calculo Manual:

$$\sum_{n=0}^5 \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} = 2.717 \approx e$$

Serie 2:

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$$

Por el criterio de comparación directa se compara con:

$$\sum_{n=1}^{\infty} \frac{1}{n^2}$$

$$L = \lim_{n \rightarrow \infty} \frac{\frac{1}{n(n+1)}}{\frac{1}{n^2}} = 1$$

∴ la serie $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$ converge

Calculo Manual:

$$\sum_{n=1}^5 \frac{1}{n(n+1)} = \frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \frac{1}{20} + \frac{1}{30} = 0.833 \approx 1$$

Serie 3:

$$\sum_{n=1}^{\infty} \left(1 - \frac{1}{\sqrt{n}}\right)^n$$

Utilizando el criterio del logartimo

$$\begin{aligned} & \lim_{n \rightarrow \infty} \frac{n \ln \left(\frac{\sqrt{n}}{\sqrt{n}-1} \right)}{\ln n} \\ \stackrel{H}{=} & \lim_{n \rightarrow \infty} \frac{\frac{1}{2} + \ln(\sqrt{n}) - \ln(\sqrt{n}-1) - \frac{\sqrt{n}}{2\sqrt{n}-2}}{\frac{1}{n}} \\ \stackrel{H}{=} & \lim_{n \rightarrow \infty} \frac{n^3 \sqrt{n} - 4n^2 + 2n^3 + n\sqrt{n}}{16n - 16\sqrt{n} + 4 - 4n^2} = \infty \\ & \therefore \text{la serie } \sum_{n=1}^{\infty} \left(1 - \frac{1}{\sqrt{n}}\right)^n \text{ diverge} \end{aligned}$$

Calculo Manual:

$$\sum_{n=1}^5 \left(1 - \frac{1}{\sqrt{n}}\right)^n = 0 + \frac{3-2\sqrt{2}}{2} + \frac{18-10\sqrt{3}}{9} + \frac{1}{16} + 0.05161629 = 0.2754 \neq 0.61$$

DIAGRAMAS DE FLUJO/ALGORITMO DESCRIPTIVO PROGRAMA

Algoritmo descriptivo del valor de euler:

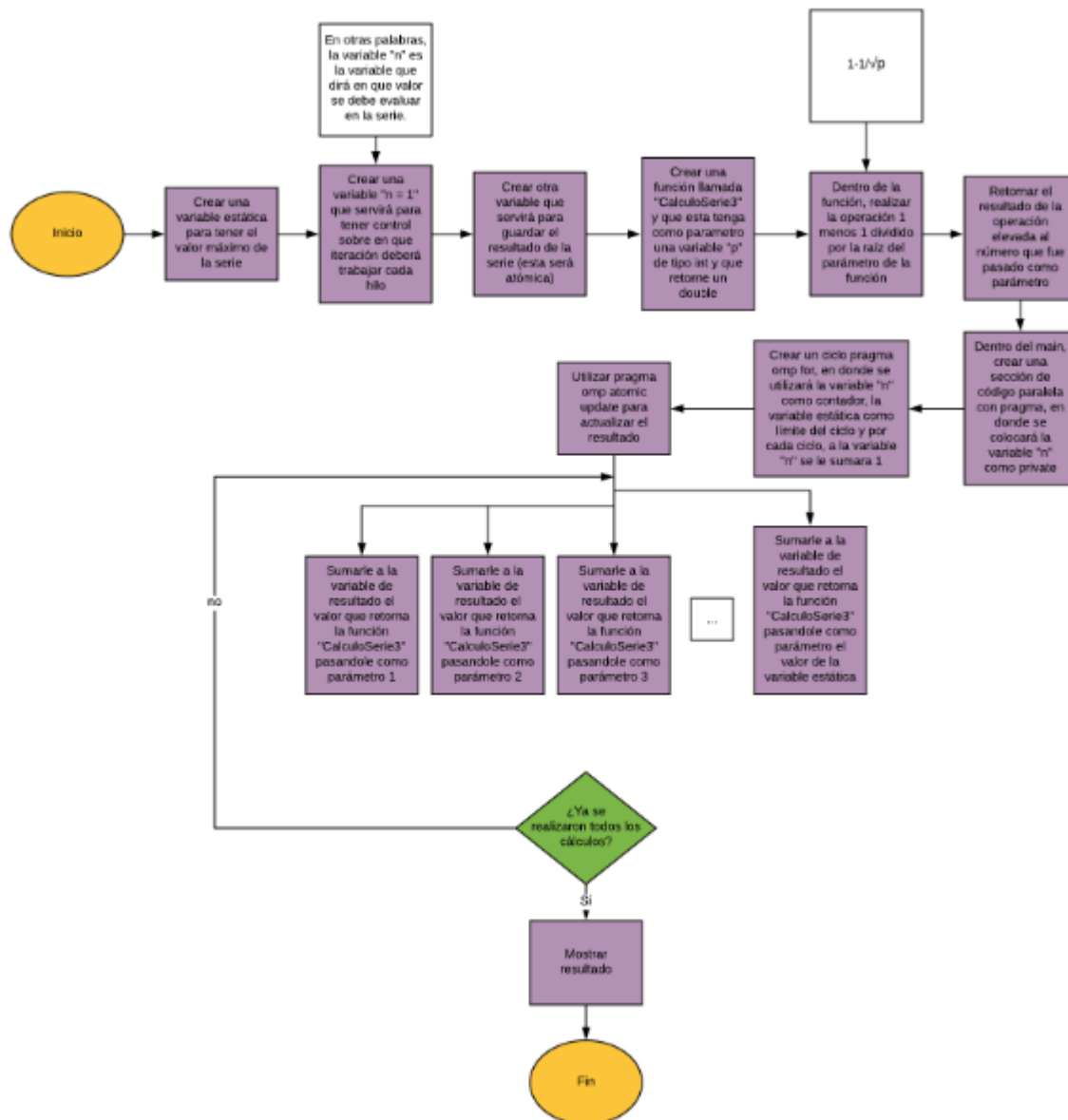
1. Se importan las librerías para OpenMP y para la lectura en consola.
2. Se define una función que actúe como factorial para usar en los cálculos.
3. Posteriormente, se define la cantidad de términos que se utilizarán para calcular Euler.
4. Se definen las variables donde se guardará el resultado final y la de cada término
5. Se realiza un for con OpenMP, donde todos los hilos se distribuyen los diferentes términos
6. Cada término es $1/n!$
7. Cada hilo suma el resultado del término al resultado final usando atomic
8. Se imprime el resultado de la serie.
9. Se calcula el porcentaje de error

10. Se imprime el porcentaje de error

Algoritmo descriptivo para el cálculo de número de convergencia de la serie $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$

1. Primero se definió las librerías a utilizar dentro del programa, las cuales en este caso incluyen librerías para trabajar con openMP y realizar cálculos matemáticos.
2. Seguido a esto se definió una constante de 10000000 la cual determina la cantidad de hilos que se utilizaran en la parte paralela para la realización de la sumatoria
3. Luego se definió la función principal y se declaró la variable suma de tipo float, la cual almacenara el resultado final de convergencia de la sumatoria.
4. Después se inició la parte paralela con `#pragma omp parallel`, y antes de iniciar el cálculo de la sumatoria se definió por medio de la directiva `#if` la cantidad de hilos a utilizar en la sumatoria con la constante definida anteriormente. Cuando ya se habían definido la cantidad hilos a utilizar, se declaró un for en paralelo con `#pragma omp parallel for`, pero en este caso para poder sincronizar los datos sin perder la paralelización se utilizó `reduction(+sum)`, con la cual el resultado que haya obtenido cada hilo será *reducido* a un solo resultado por medio de la suma.
5. Finalmente se concluye la parte paralela y por medio de un `printf` se imprime le resultado obtenido.

Diagrama de flujo para el cálculo de número de convergencia de la serie $\sum_{n=1}^{\infty} \left(1 - \frac{1}{\sqrt{n}}\right)^n$:



PROGRAMA SOLUCIÓN

Serie de Euler: $\sum_{n=0}^{\infty} \frac{1}{n!}$

Código fuente

```

// =====CONSTANTES =====

// Cantidad de términos que se van a usar en el cálculo
#define N_TERMINOS 100
// El valor real de euler para porcentaje de error
#define REAL_EULER 2.71828182846

// ===== FUNCIONES ÚTILES =====

/**
 * Función factorial
 * Calcula el valor del factorial de un número
 * No es recursiva, pero podría ser.
 *
 * @param numero - El número al que se le sacará el factorial
 * @return El valor del factorial del número
 */
long double factorial(int numero){
    long double resultado = 1;

    while(numero > 1){
        resultado *= numero;
        numero--;
    }

    return resultado;
}

// ===== MAIN =====

// ===== MAIN =====

int main(){
    int i = 0;
    long double euler = 0;
    long double termino = 0;

    cout.precision(30);          // Para que imprima todos los decimales.

    // Hago un ciclo con paralelismo para que cada hilo calcule el término
    #pragma omp for private (termino)
    for(i = 0; i < N_TERMINOS; i++){
        // El valor de cada término es 1/n!
        termino = 1/factorial(i);
        cout << "Término " << i << ": " << termino << endl;

        // Actualizo el resultado global
        #pragma omp atomic
        euler += termino;
    }

    cout << endl;
    cout << "El valor de euler calculado con los primeros " << N_TERMINOS << " términos es: " << euler << endl;

    cout.precision(3);
    cout << "El porcentaje de error fue de " << 100 * abs(euler - REAL_EULER) / REAL_EULER << "%" << endl;
    return 0;
}

```

Prueba


```

Término 93: 8.64474210683694486394869066019e-145
Término 94: 9.19653415620951458310014992335e-147
Término 95: 9.68056226969422125229322015184e-149
Término 96: 1.00839190309314877066815753049e-150
Término 97: 1.03957928153932765051872625191e-152
Término 98: 1.06079518524421217556434778711e-154
Término 99: 1.07151028812546685838012365776e-156

El valor de euler calculado con los primeros 100 términos es: 2.71828182845904553488480814849
El porcentaje de error fue de 3.51e-11%

```

Comparación de resultado obtenido:

Infinite sum:

$$\sum_{n=0}^{\infty} \frac{n}{n!} = e$$

Decimal approximation:

2.7182818284590452353602874713526624977572470936999595749669676277
...

Serie 2: $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$

Código fuente

```

int main() {

    double sum = 0;
    int a = 1;

    #pragma omp parallel reduction(+:sum)
    {
        #ifdef _OPENMP
        omp_set_num_threads(NTH);
        #endif

        #pragma omp for
        for(a = 1; a < NTH; a++) {
            sum += (1.0 / (pow(a, 2) + a));
        }
    }
    printf("La sumatoria converge a = %f\n", sum);
    return 0;
}

```

Prueba





```

pi@raspberrypi:~ $ g++ -o serie2 -fopenmp serie2.cpp
pi@raspberrypi:~ $ ./serie2
La sumatoria converge a = 1.000000

```

Comparación de resultado obtenido:

sum 1/(n*(n+1)), n=1 to infinity =

 Extended Keyboard
 Upload
 Examples
 Random

Infinite sum:

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = 1$$

Serie 3: $\sum_{n=1}^{\infty} \left(1 - \frac{1}{\sqrt{n}}\right)^n$

Código fuente:

```

using namespace std;
#define limite 100000 //Simulación de infinito
#define threads 100000 //Hilos que trabajaran
int n = 1; //Inicio de la serie
double resultado = 0; //Resultado de la serie
//Función para realizar la operación de la serie (1-1/√n)^n
double CalculoSerie1(int p){
    double calculo = 1-(1/sqrt(p));
    return pow(calculo,p);
}

int main()
{
    #pragma omp parallel private(n)
    {
        #pragma omp for
        #pragma omp set_num_threads(threads);
        #pragma omp for
        for(n = 1;n<=limite;n++) //Ciclo for para realizar el calculo de la serie
        {
            #pragma omp atomic update //Sección atómica para actualizar la variable "resultado" por todos los hilos
            resultado += CalculoSerie1(n);
        }
    }
    printf ("Resultado de la sumatoria = X/n", resultado);
}

```

Comparación de resultado obtenido:

$$\text{Sum}\left[\left(1 - \left(\frac{1}{\sqrt{n}}\right)\right)^n, \{n, 1, \infty\}\right]$$

Extended Keyboard
Upload
Examples
Random

Input interpretation:

$$\sum_{n=1}^{\infty} \left(1 - \frac{1}{\sqrt{n}}\right)^n$$

Approximated sum: More digits

$$\sum_{n=1}^{\infty} \left(1 - \frac{1}{\sqrt{n}}\right)^n \approx 0.615917$$

Resultado del programa usando como límite 100000:

```

pi@raspberrypi:~$ g++ -o Serie3 -fopenmp Serie3.cpp
pi@raspberrypi:~$ ./Serie3
Resultado de la sumatoria = 0.615930

```

REFERENCIAS

- Alfonso I (06/2009). Introducción a la matemática discreta. Extraído de:
https://www.youtube.com/watch?v=in48xZip2IA&ab_channel=ildefonsoarevalo
- UaesSharkComputing (sf). Operaciones básicas de una computadora. Extraído de:
<https://sites.google.com/site/uaesharkcomputing/operaciones-basicas-de-una-computadora>
- Lopategui, E. (s.f.). *Historia de las computadoras*. Extraído de:
<http://biblio3.url.edu.gt/Libros/provinciales/computadoras.pdf>

Omni.sci. (s.f.) *Parallel Computing*. Extraído de:

<https://www.omnisci.com/technical-glossary/parallel-computing>

Gasca-Hurtado, G. P., & Machuca-Villegas, L. (2018). El impacto de las Ciencias de la Computación en el mundo real. *RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação*, 29, 10. <https://doi.org/10.17013/risti.29.0>