Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3056 Programación de Microprocesadores
Catedrático: Kimberly Barrera
Ciclo 2 de 2020

Proyecto de investigación y aplicación 3

Integrantes:
Donaldo Sebastián García Jiménez 19683
Bryann Eduardo Alfaro 19372
Diego de Jesús Arredondo 19422
Oscar René Saravia Donis 19322
Raúl Ángel Jiménez Hernández 19017

Guatemala, 26 de octubre de 2020

Link repositorio de Github:
https://github.com/bryannalfaro/Proyecto3Microprocesadores

Planificación/Aporte de cada integrante

Tarea ↓ Día →	Encargado	1	2	3	4	5	6	7	8	9
Primer programa	**Oscar Saravia**									
Segundo programa	**Donaldo García y Raúl Jiménez**									
Tercer programa	**Bryann Alfaro y Diego Arredondo**									
Cuarto programa	**Todos**									
Bitácora	**Todos**									

Marco teórico

- Programación paralela en ramas de la física y matemáticas: La programación paralela es muy importante en las ramas de física y matemática, pues al estudiar estas ramas se hace uso constante de las sumatorias de riemann así como límites que tienden al infinito. Al tener muchos valores tendiendo al infinito se debe buscar una tecnología que permita estudiar dicho comportamiento en poco tiempo. Es ahí donde la programación paralela hace su aparición, con la facilidad que se tiene de hacer múltiples tareas a la vez los matemáticos y físicos han optado por utilizar la programación para comprobar teorías o hacer cálculos de manera eficaz.

- Aplicación de OpenMP para el cálculo de valores numéricos de precisión: OpenMP nos permite trabajar con hilos paralelamente, esto agiliza el proceso de los cálculos realizados en las ramas de matemática y física. Al realizar sumatorias infinitas nos permite realizar esta operación en distintos hilos de manera continua y luego ser sumados en la variable atómica. (Bernal, F).

Partes en paralelo(incluir funciones disponibles en OpenMP) y secuencial

1. Primer programa:

- Se utilizó programación paralela para poder calcular todos los valores que al ser sumados dan como resultado el número de Euler, específicamente se utilizó la función *omp parallel for*, la cual recorre 12 veces el proceso de calcular $\frac{1}{i!}$ y sumarlo al resultado final. También se utilizaron rutinas bloqueadas para evitar las condiciones de carrera entre los hilos. Las partes utilizadas para la parte secuencial fueron únicamente la impresión de los resultados al finalizar el programa.

2. Segundo programa:

    - Se utilizó programación secuencial al momento de mostrarle al usuario el límite de la sumatoria evaluado en $\left|\frac{n+1}{n}\right|$ para que se pudiera observar el comportamiento del límite.
    - El paralelismo se utilizó al momento de hacer la sumatoria con valores "i" muy grandes para simular el comportamiento al infinito. Se utilizó un "pragma for" para que las iteraciones se hicieran en paralelo y se hizo uso de "pragma atomic" que funciona como una barrera para que la variable en la que se iba guardando la sumatoria no se viera afectada al tener muchos hilos modificando a la vez.

3. Tercer y cuarto programa:

    - Se utilizó programación secuencial para poder evaluar el límite de la función cuando tiende a infinito para poder comprobar el teorema de la divergencia de una serie. Esto debido a que como era solamente un cálculo era innecesario utilizar paralelismo ya que no se aprovecharía todas sus propiedades.
    - Se utilizó programación paralela en los cálculos del valor de la serie para una cantidad "n" especifica ya que para valores muy grandes el proceso sería tardado y al paralelizar esta parte el cálculo se distribuye equitativamente en los hilos por medio de la utilización de un "pragma for". De igual manera, como forma de sincronización se utilizó la instrucción "pragma atomic" para tener el control sobre el acumulador de la sumatoria y que no afectara en el resultado.

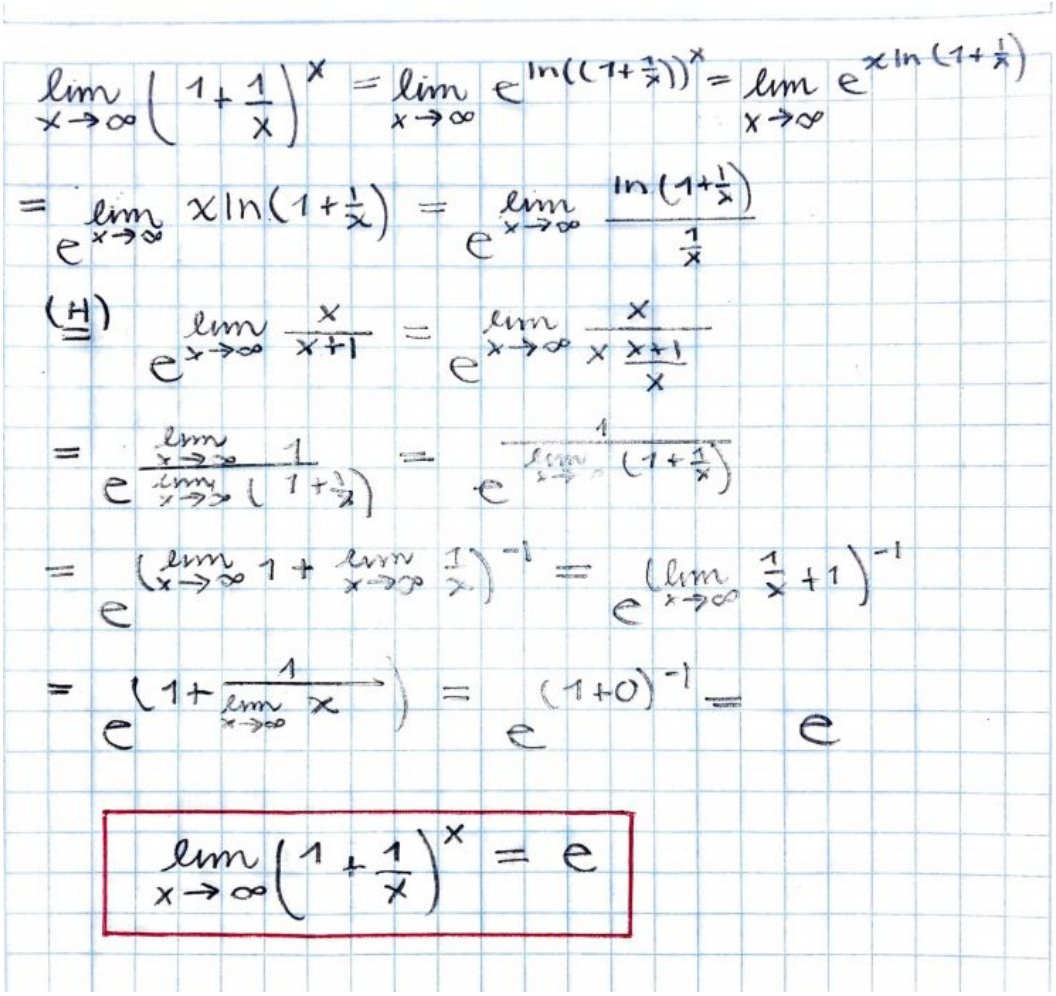
Solución matemática

Para la programación de las series que divergen se utilizó el siguiente teorema el cual menciona que al encontrar el límite de la serie cuando tiende a infinito y su resultado no es igual a 0 entonces la serie diverge.

7 La prueba de la divergencia Si $\lim_{n\to\infty} a_n$ no existe o si $\lim_{n\to\infty} a_n \neq 0$, entonces la serie $\sum_{n=1}^{\infty} a_n$ es divergente.

Imagen tomada de: Cálculo de varias variables.

1. Primera solución



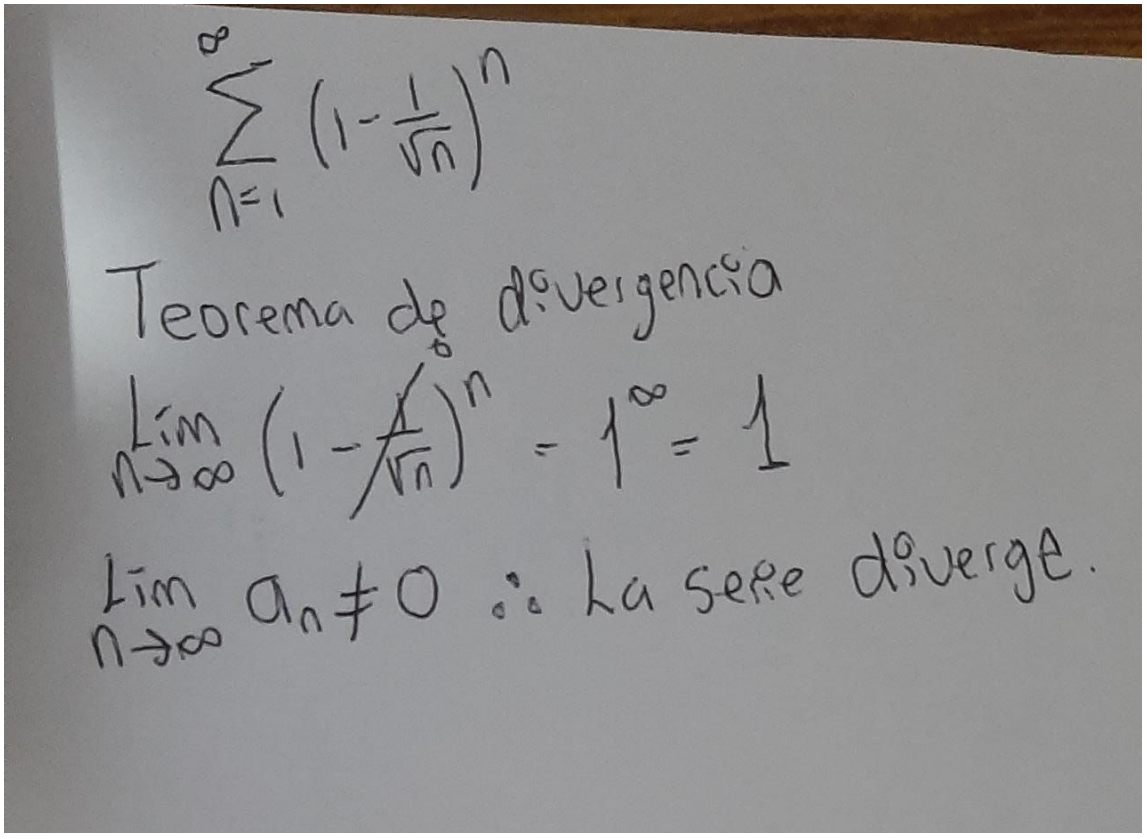
2. Segunda solución

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} \to \lim_{n\to\infty} \left| \frac{\frac{1}{n+1(n+1+1)}}{\frac{1}{n(n+1)}} \right| = \frac{n\cancel{(n+1)}}{\cancel{(n+1)}(n+2)} \cdots$$

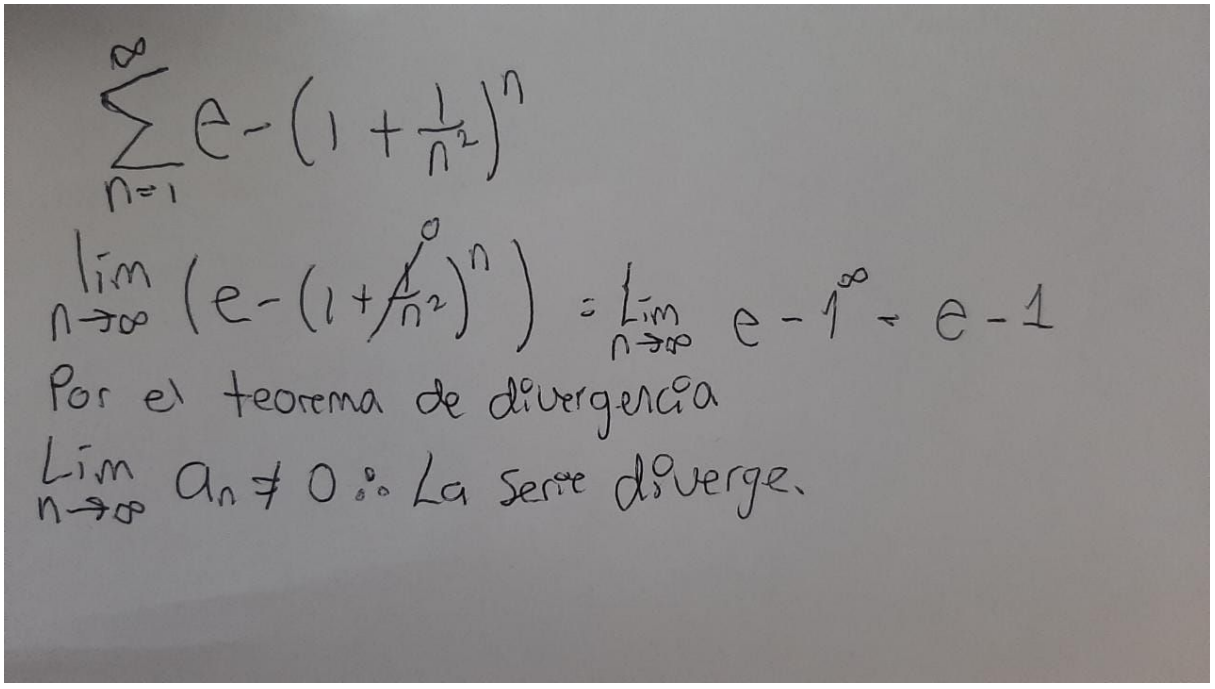
$$\cdots \frac{n}{n+2} \quad \text{l'hopital} \quad \frac{1}{1} \to \lim_{n\to\infty} 1 = 1$$

Converge en 1.

3. Tercera solución

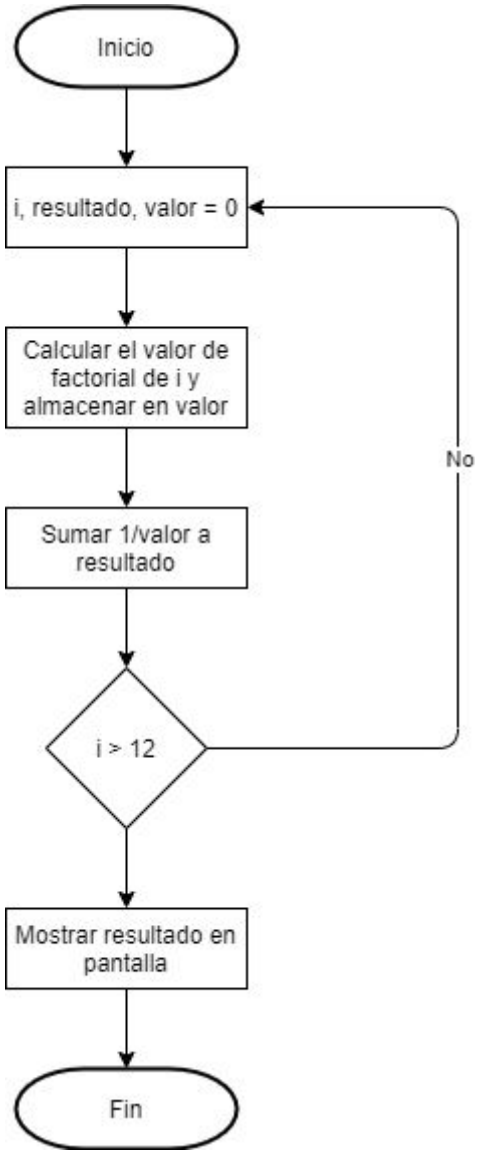


4. Cuarta solución

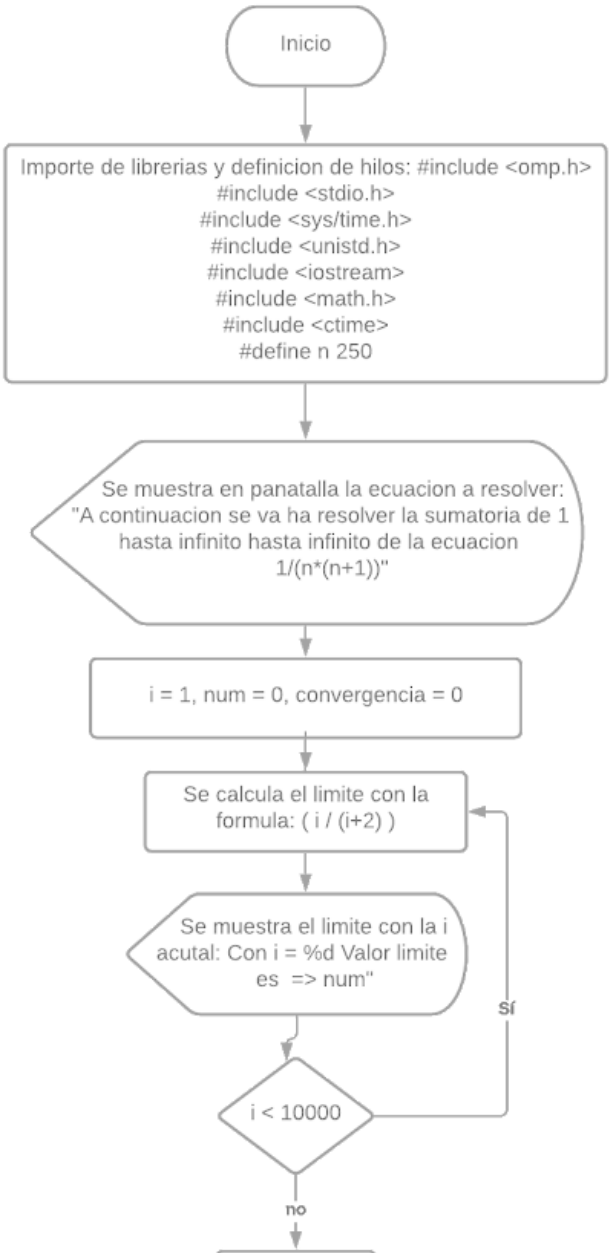


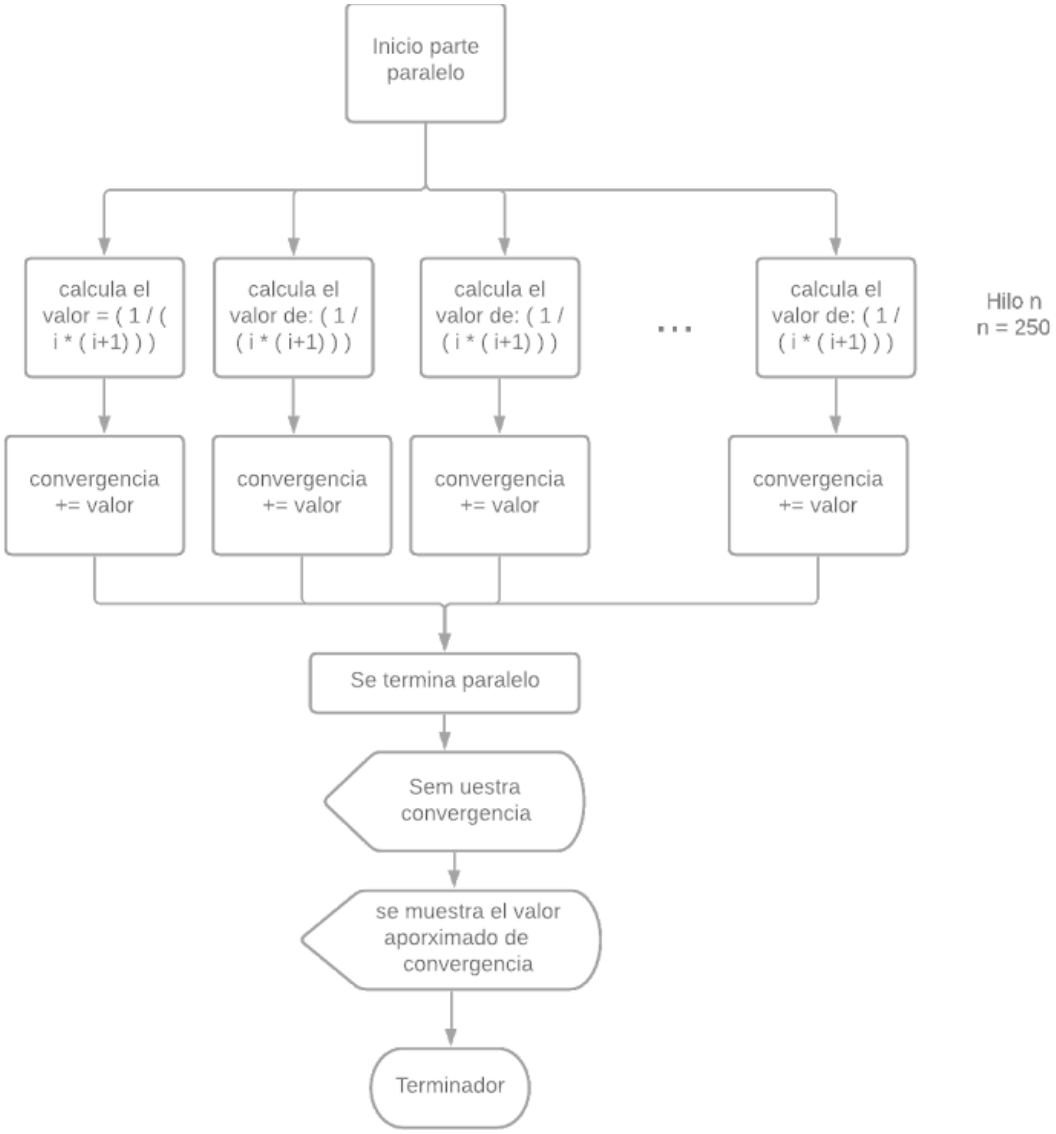
Diagramas de flujo

1. Primer diagrama de flujo

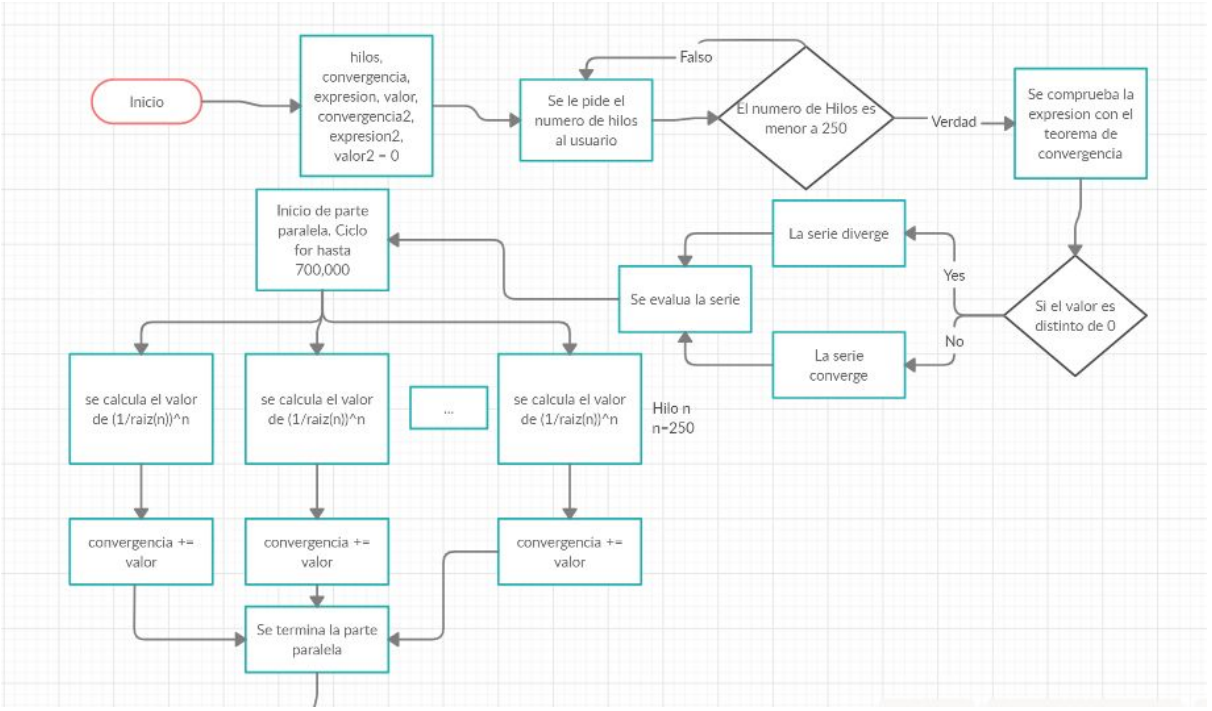


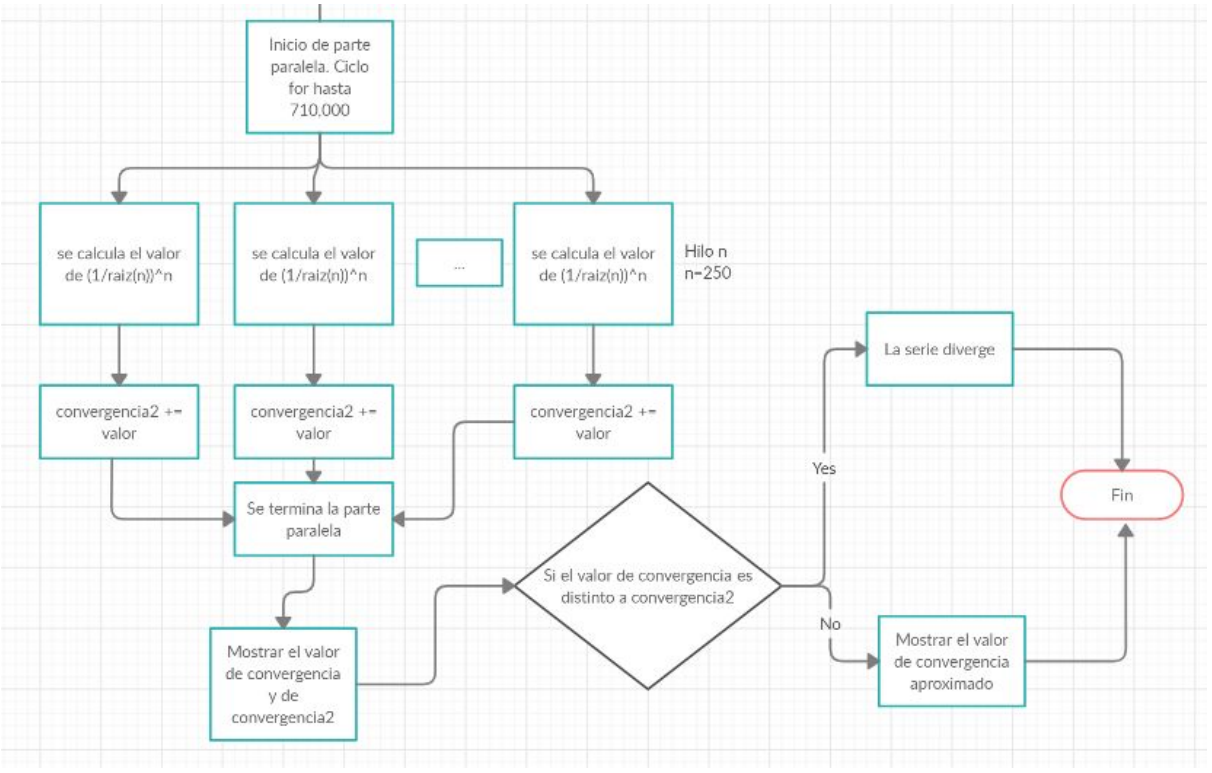
2. Segundo diagrama de flujo



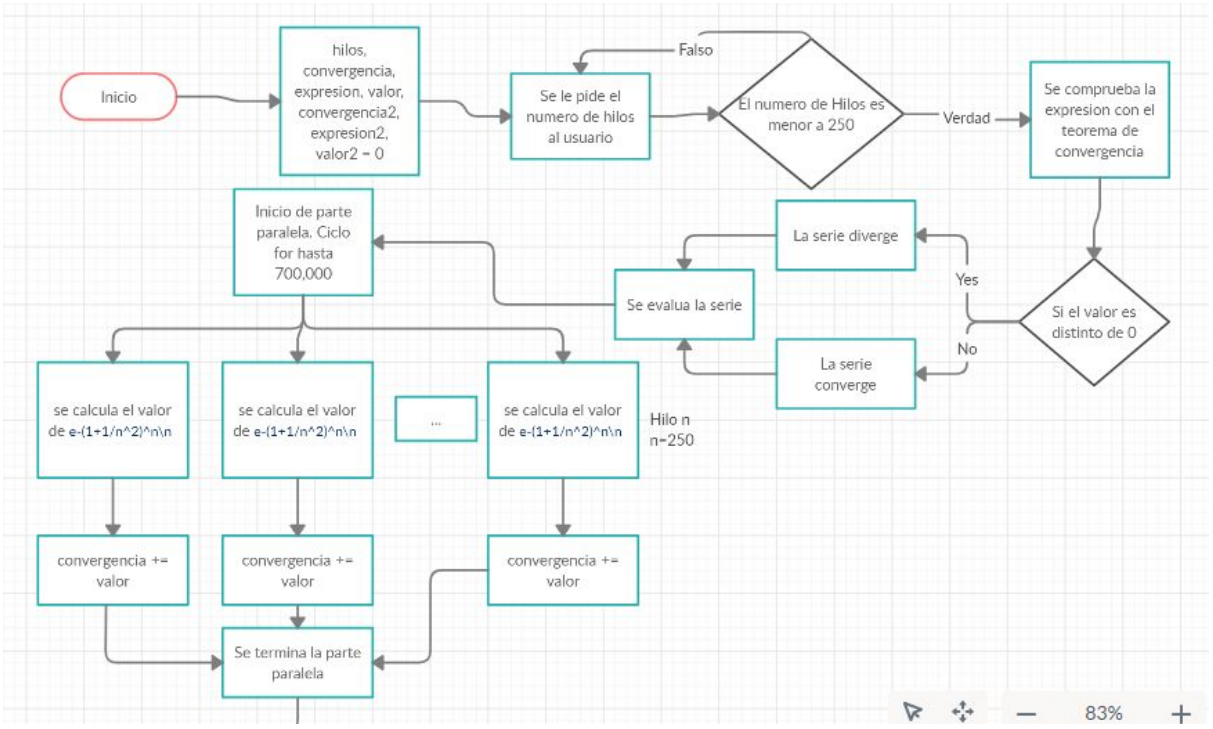

Inicio parte paralelo
calcula el valor = (1 / (i * (i+1)))
calcula el valor de: (1 / (i * (i+1)))
calcula el valor de: (1 / (i * (i+1)))
...
calcula el valor de: (1 / (i * (i+1)))
Hilo n
n = 250
convergencia += valor
convergencia += valor
convergencia += valor
convergencia += valor
Se termina paralelo
Sem uestra convergencia
se muestra el valor aporximado de convergencia
Terminador

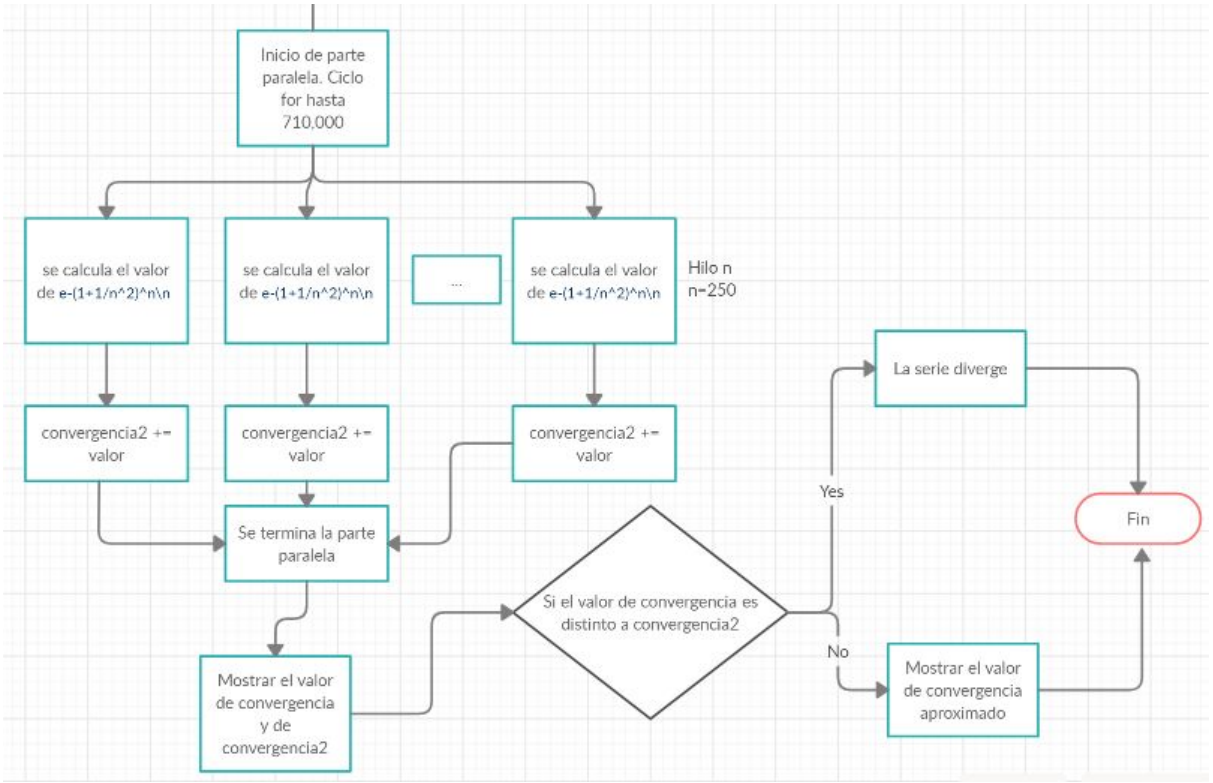
3. Tercer diagrama de flujo





4. Cuarto diagrama de flujo





Programa solución

1. Primer programa

```
/*------------------------------------------------------------
* UNIVERSIDAD DEL VALLE DE GUATEMALA
* CC3056-10 Programación de microprocesadores
* Autor: Oscar Saravia
*
* Euler.cpp
* Fecha: 18/10/2020
* ------------------------------------------------------------*/
// omp_init_lock.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <iostream>
#include <iomanip>

using namespace std;


// Metodo para calcular factorial
double fact(int n) {
   if ((n==0)||(n==1))
   return 1;
   else
   return n*fact(n-1);
}

// Inicizalizacion de variables
omp_lock_t lck;
double resultado = 0;
double   valor = 0;

int main() {

   omp_init_lock(&lck);

   #pragma omp parallel for
   for (int i = 0; i < 13; i++)
   {
       omp_set_lock(&lck);
       valor = 1/(fact(i));
       resultado = resultado + valor;
       omp_unset_lock(&lck);

   }


    cout <<"El valor de e es aproximadamente: " << setprecision(10)<< resultado<<"\n";
   omp_destroy_lock(&lck);

   return 0;
}
```

2. Segundo programa

```
*Realiza el ejercicio B del proyecto
*numero 3 de dos maneras distintas
* y muestra el resultado al final
*-----------------------------------
*Raul Jimenez 19017
*Donaldo Garcia 19683
*Bryann Alfaro 19372
*Oscar Saravia 19322
*Diego Arredondo 19422
*-----------------------------------*/
#include <omp.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>
#include <iostream>
#include <math.h>
#include <ctime>

#define n 250

int main()
{
    //Informacion de lo que va a hacer le programa
    printf("A continuacion se va ha resolver la sumatoria de 1 hasta infinito hasta infinito de la ecuacion  1/(n*(n+1)) \n");

    //variables para convergencia
    double convergencia = 0;
    int convergenciaEntera = 0;
    int valorLimite = 0;

    //valor con limites
    for(int i = 1; i < 10000; i+=1000)
    {
        //se utiliza la ecuacion obtenida con limites
        double num = i/(i+2.f);

        //se muestran los cambios en cada corrida del limite
        printf("Con i = %d Valor limite es  => %lf \n", i, num);
        //se hace una pausa para ver la diferencia entre datos
        sleep(1);
    }
```

```
    //parte paralela para obtener las sumatoria con otra n
    #pragma omp parallel for num_threads(n)
        for(int i =1; i<710000; i++)
        {
            //cada hilo realiza la operacion dada
            double valor2 = 1.f/(i*(i+1.f));

            //cada hilo almacena el dato en la variable convergencia
            #pragma omp atomic
                convergencia2+=valor2;
        }

    //se redondea la la sumatoria
    convergenciaEntera = round(convergencia);
    convergenciaEntera2 = round(convergencia2);
    //Se muestran los valores originales y redondeados
    printf("El valor de convergencia aproximado con n= 598461 es: %lf \n", convergencia);
    printf("El valor de convergencia aproximado con n= 710000 es: %lf \n", convergencia2);
    printf("El valor de convergencia entero es: %d \n", convergenciaEntera);

    return 0;
}
```

3. Tercer programa

```
/*
*   Proyecto 3 ejercicio C
*-----------------------------------
*Realiza el ejercicio C del proyecto
*numero 3 de dos maneras distintas
* y muestra el resultado al final
*-----------------------------------
*Raul Jimenez 19017
*Donaldo Garcia 19683
*Bryann Alfaro 19372
*Oscar Saravia 19322
*Diego Arredondo 19422
*-----------------------------------*/
#include <omp.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>
#include <iostream>
#include <math.h>
#include <ctime>



int main(){

    #define n 10000
    int hilos;
    double convergencia=0;
    double expresion=0;
    double valor=0;
    double convergencia2=0;
    double expresion2=0;
    double valor2=0;

    printf("Bienvenido al programa de calculo de serie\n");
    printf("Se calculara la serie de (1-1/raiz(n))^n\n\n");

    printf("-----------------------------------------------\n");
    printf("Porfavor ingrese la cantidad de hilos que desea para el programa: (menor a 250)\n");

    scanf("%d",&hilos);
```

```
    if(hilos>250){
        printf("El numero de hilos debe ser menor a 250. Intente de nuevo.\n");
        return(1);
        }


    //Limite cuando la serie tiende a infinito
    printf("Teorema de la divergencia: \n");

    expresion = 1-(1/sqrt(n));

    valor = pow(round(expresion),n);
    printf("Valor del limite: %lf\n",valor);
    if(valor!=0)
    {
        printf("Por el teorema la serie diverge\n");
    }
    else
    {
        printf("La serie es potencialmente convergente\n");
    }

    printf("-----------------------------------------------\n");
    printf("EVALUANDO LA SERIE...........\n");

    //parte paralela para obtener las sumatorias
    #pragma omp parallel for num_threads(hilos)
        for(int i =1; i<700000; i++)
        {
            valor = pow(1-(1/sqrt(i)),i);

            #pragma omp atomic
                convergencia+=valor;
        }

    #pragma omp parallel for num_threads(hilos)
        for(int i =1; i<710000; i++)
        {
            valor2 = pow(1-(1/sqrt(i)),i);

            #pragma omp atomic
                convergencia2+=valor2;
        }
```

```
printf("El valor de la serie con n = 700,000 es: %lf \n", convergencia);
printf("El valor de la serie con n = 710,000 es: %lf \n", convergencia2);
if(convergencia2!=convergencia)
{
    printf("La serie diverge\n");
}

else
{
    printf("El valor de convergencia aproximado es: %lf \n", convergencia);
}

return 0;

}
```

4. Cuarto programa

```
/*
*   Proyecto 3 ejercicio D
*-------------------------------------
*Realiza en ejercicio D del proyecto
*numero 3 de dos maneras distintas
* y muestra el resultado al final
*-------------------------------------
*Raul Jimenez 19017
*Donaldo Garcia 19683
*Bryan Alfaro 19372
*Oscar Saravia 19322
*Diego Arredondo 19422
*-------------------------------------*/
#include <omp.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>
#include <iostream>
#include <math.h>
#include <ctime>


int main(){

    #define n 10000
    int hilos;
    double convergencia=0;
    double expresion=0;
    double valor=0;
    double convergencia2=0;
    double expresion2=0;
    double valor2=0;

    printf("Bienvenido al programa de calculo de serie\n");
    printf("Se calculara la serie de e-(1+1/n^2)^n\n\n");

    //Se permite especificar los hilos que desea
    printf("-----------------------------------------------\n");
    printf("Porfavor ingrese la cantidad de hilos que desea para el programa: (menor a 250)\n");

    scanf("%d",&hilos);
```

```c
if(hilos>250){
    printf("El numero de hilos debe ser menor a 250. Intente de nuevo.\n");
    return(1);
    }


//Limite cuando la serie tiende a infinito
printf("Teorema de la divergencia: \n");

expresion = exp(1)-pow(1+(1/pow(n,2)),n);

printf("Valor del limite: %lf\n",expresion);
if(expresion!=0)
{
    printf("Por el teorema la serie diverge\n");
}
else
{
    printf("La serie es potencialmente convergente\n");
}

printf("-----------------------------------------------\n");
printf("EVALUANDO LA SERIE..........\n");

//parte paralela para obtener las sumatorias
#pragma omp parallel for num_threads(hilos)
    for(int i =1; i<700000; i++)
    {
        valor = exp(1)-pow(1+(1/pow(i,2)),i);

        #pragma omp atomic
            convergencia+=valor;

    }

#pragma omp parallel for num_threads(hilos)
    for(int i =1; i<710000; i++)
    {

        valor2 = exp(1)-pow(1+(1/pow(i,2)),i);

        #pragma omp atomic
            convergencia2+=valor2;

    }


printf("El valor de la serie con n=700,000 es: %lf \n", convergencia);
printf("El valor de la serie con n=710,000 es: %lf \n", convergencia2);
if(convergencia2!=convergencia)
{
    printf("La serie diverge\n");
}

else
{
    printf("El valor de convergencia aproximado es: %lf \n", convergencia);
}

return 0;

}
```

Bibliografía

- CPLUSPLUS. (s.f). exp. Extraído de: http://www.cplusplus.com/reference/cmath/exp/

- Wikipedia. (s.f). Exponential Function. Extraído de: https://en.wikipedia.org/wiki/Exponential_function

- Stewart, James. Cálculo de varias variables. Trascendentes tempranas, octava edición. ISBN: 978-607-526-553-7.

  Bernal, F. (s.f). Programación Paralela. Extraído de: http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html