Universidad del Valle De Guatemala

Facultad de Ingeniería

Computación Paralela y Distribuida

Laboratorio 1: Pi

Javier Valle 20159
Mario de León 19019

Guatemala, 15 de agosto 2023

Inciso a:

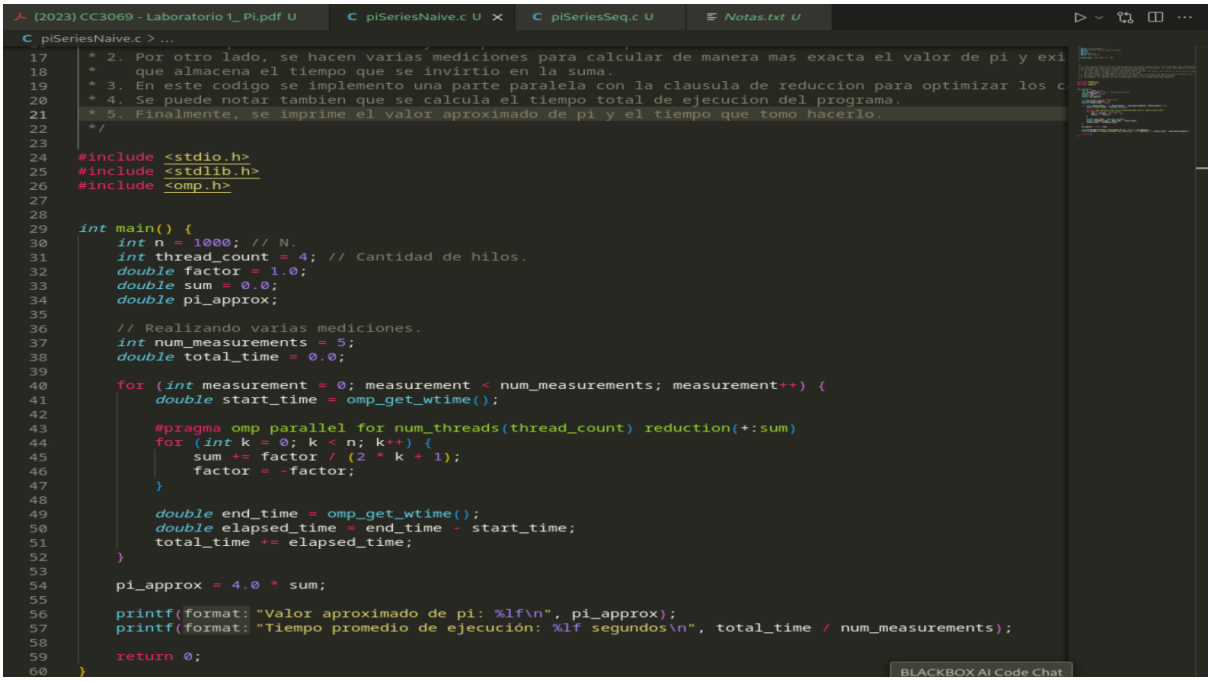
piSeriesSeq

```c
/**
 * @file piSeriesSeq.c
 * @author Javier Valle, Mario de León
 * @brief
 * @version 0.1
 * @date 2023-08-15
 *
 * @copyright Copyright (c) 2023
 *
 */

/**
 * Programa secuencial que calcula el valor de pi.
 * factor es un termino que ayuda a calcular la aprox. de pi.
 * sum es la suma iterativa que se va haciendo a lo largo de la serie.
 * pi_approx es el valor final que se obtiene luego de haber hecho las iteraciones de la serie.
 */

#include <stdio.h>

int main() {
    int n = 1000; // Valor de iteraciones.
    double factor = 1.0;
    double sum = 0.0;

    for (int k = 0; k < n; k++) {
        sum += factor / (2 * k + 1);
        factor = -factor;
    }

    double pi_approx = 4.0 * sum;

    printf(format: "Valor aproximado de pi: %lf\n", pi_approx);

    return 0;
}
```

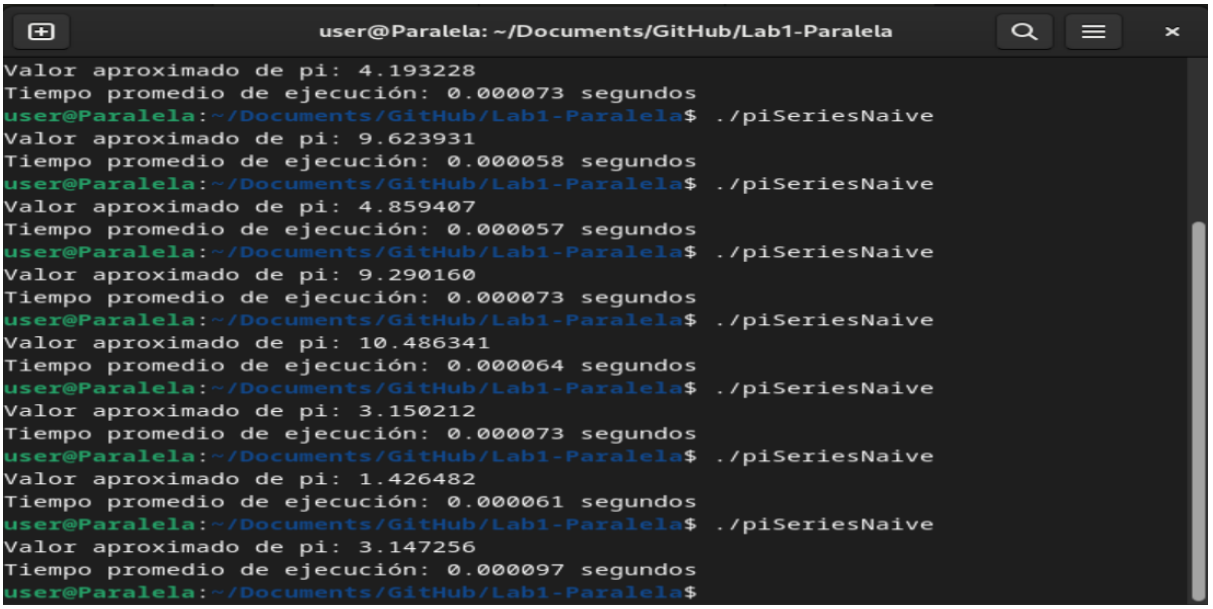
```
user@Paralela: ~/Documents/GitHub/Lab1-Paralela
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesSeq
Valor aproximado de pi: 3.140593
user@Paralela:~/Documents/GitHub/Lab1-Paralela$
```

Se puede notar que en este primer programa secuencial, el valor de pi se obtuvo con una precisión bastante alta luego de haber hecho 1000 iteraciones en la serie. También se puede notar que se tuvo que hacer únicamente una corrida para poder obtener el valor más exacto de pi.

piSeriesNaive
Versión 1



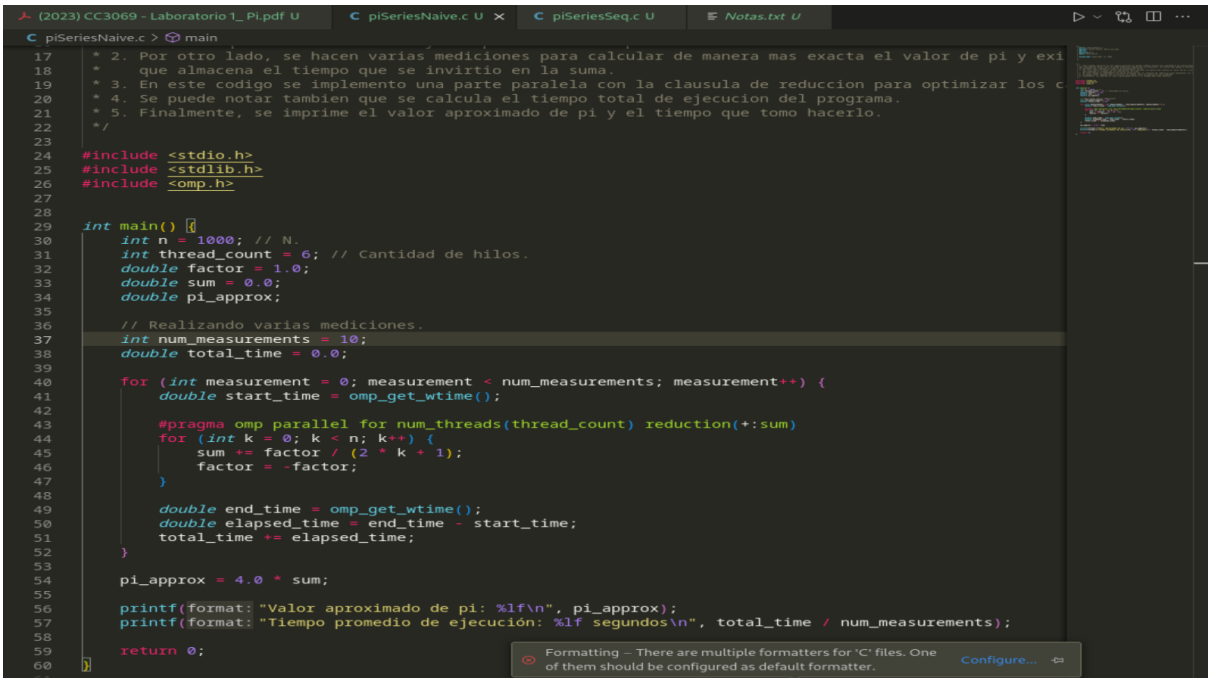
```c
/*
 * 2. Por otro lado, se hacen varias mediciones para calcular de manera mas exacta el valor de pi y exi
 *    que almacena el tiempo que se invirtio en la suma.
 * 3. En este codigo se implemento una parte paralela con la clausula de reduccion para optimizar los c
 * 4. Se puede notar tambien que se calcula el tiempo total de ejecucion del programa.
 * 5. Finalmente, se imprime el valor aproximado de pi y el tiempo que tomo hacerlo.
 */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>


int main() {
    int n = 1000; // N.
    int thread_count = 4; // Cantidad de hilos.
    double factor = 1.0;
    double sum = 0.0;
    double pi_approx;

    // Realizando varias mediciones.
    int num_measurements = 5;
    double total_time = 0.0;

    for (int measurement = 0; measurement < num_measurements; measurement++) {
        double start_time = omp_get_wtime();

        #pragma omp parallel for num_threads(thread_count) reduction(+:sum)
        for (int k = 0; k < n; k++) {
            sum += factor / (2 * k + 1);
            factor = -factor;
        }

        double end_time = omp_get_wtime();
        double elapsed_time = end_time - start_time;
        total_time += elapsed_time;
    }

    pi_approx = 4.0 * sum;

    printf(format: "Valor aproximado de pi: %lf\n", pi_approx);
    printf(format: "Tiempo promedio de ejecución: %lf segundos\n", total_time / num_measurements);

    return 0;
}
```



```
user@Paralela: ~/Documents/GitHub/Lab1-Paralela
Valor aproximado de pi: 4.193228
Tiempo promedio de ejecución: 0.000073 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 9.623931
Tiempo promedio de ejecución: 0.000058 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 4.859407
Tiempo promedio de ejecución: 0.000057 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 9.290160
Tiempo promedio de ejecución: 0.000073 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 10.486341
Tiempo promedio de ejecución: 0.000064 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 3.150212
Tiempo promedio de ejecución: 0.000073 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 1.426482
Tiempo promedio de ejecución: 0.000061 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 3.147256
Tiempo promedio de ejecución: 0.000097 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$
```

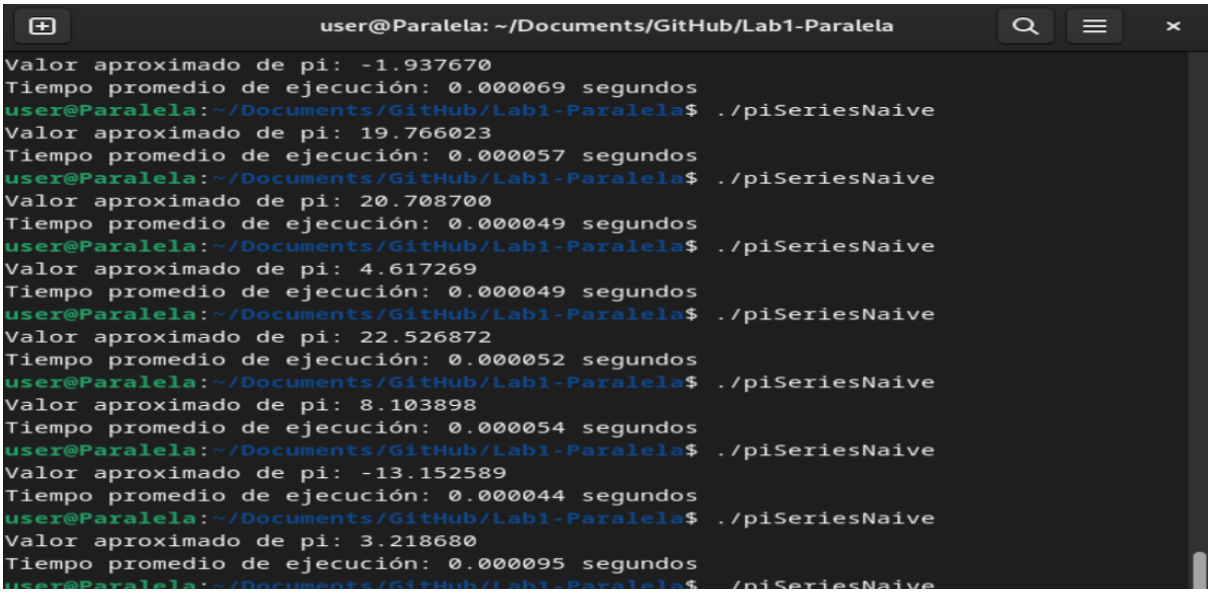
En este primer programa se usó un N = 1000, 4 threads y 5 mediciones para poder realizar la aproximación de Pi. Se puede notar que tomó varias corridas querer llegar a un resultado más exacto.

Versión 2



```
 * 2. Por otro lado, se hacen varias mediciones para calcular de manera mas exacta el valor de pi y exi
 *    que almacena el tiempo que se invirtio en la suma.
 * 3. En este codigo se implemento una parte paralela con la clausula de reduccion para optimizar los c
 * 4. Se puede notar tambien que se calcula el tiempo total de ejecucion del programa.
 * 5. Finalmente, se imprime el valor aproximado de pi y el tiempo que tomo hacerlo.
 */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>


int main() {
    int n = 1000; // N.
    int thread_count = 6; // Cantidad de hilos.
    double factor = 1.0;
    double sum = 0.0;
    double pi_approx;

    // Realizando varias mediciones.
    int num_measurements = 10;
    double total_time = 0.0;

    for (int measurement = 0; measurement < num_measurements; measurement++) {
        double start_time = omp_get_wtime();

        #pragma omp parallel for num_threads(thread_count) reduction(+:sum)
        for (int k = 0; k < n; k++) {
            sum += factor / (2 * k + 1);
            factor = -factor;
        }

        double end_time = omp_get_wtime();
        double elapsed_time = end_time - start_time;
        total_time += elapsed_time;
    }

    pi_approx = 4.0 * sum;

    printf(format: "Valor aproximado de pi: %lf\n", pi_approx);
    printf(format: "Tiempo promedio de ejecución: %lf segundos\n", total_time / num_measurements);

    return 0;
}
```

Formatting – There are multiple formatters for 'C' files. One of them should be configured as default formatter. Configure...



```
user@Paralela: ~/Documents/GitHub/Lab1-Paralela
Valor aproximado de pi: -1.937670
Tiempo promedio de ejecución: 0.000069 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 19.766023
Tiempo promedio de ejecución: 0.000057 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 20.708700
Tiempo promedio de ejecución: 0.000049 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 4.617269
Tiempo promedio de ejecución: 0.000049 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 22.526872
Tiempo promedio de ejecución: 0.000052 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 8.103898
Tiempo promedio de ejecución: 0.000054 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: -13.152589
Tiempo promedio de ejecución: 0.000044 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 3.218680
Tiempo promedio de ejecución: 0.000095 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
```

En esta segunda versión del programa, se utilizó como parámetros N = 1000, 6 hilos y 10 mediciones. Cabe descartar que a esta versión del programa le tomó más corridas poder llegar a un resultado más aproximado y más certero. También es importante mencionar que las corridas en esta versión del programa fueron un poco más tardadas que en la versión anterior.

Versión 3

piSeriesNaive.c > main

```c
/*
 * 2. Por otro lado, se hacen varias mediciones para calcular de manera mas exacta el valor de pi y exi
 *    que almacena el tiempo que se invirtio en la suma.
 * 3. En este codigo se implemento una parte paralela con la clausula de reduccion para optimizar los c
 * 4. Se puede notar tambien que se calcula el tiempo total de ejecucion del programa.
 * 5. Finalmente, se imprime el valor aproximado de pi y el tiempo que tomo hacerlo.
 */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>


int main() {
    int n = 3000; // N.
    int thread_count = 8; // Cantidad de hilos.
    double factor = 1.0;
    double sum = 0.0;
    double pi_approx;

    // Realizando varias mediciones.
    int num_measurements = 15;
    double total_time = 0.0;

    for (int measurement = 0; measurement < num_measurements; measurement++) {
        double start_time = omp_get_wtime();

        #pragma omp parallel for num_threads(thread_count) reduction(+:sum)
        for (int k = 0; k < n; k++) {
            sum += factor / (2 * k + 1);
            factor = -factor;
        }

        double end_time = omp_get_wtime();
        double elapsed_time = end_time - start_time;
        total_time += elapsed_time;
    }

    pi_approx = 4.0 * sum;

    printf(format: "Valor aproximado de pi: %lf\n", pi_approx);
    printf(format: "Tiempo promedio de ejecución: %lf segundos\n", total_time / num_measurements);

    return 0;
}
```

Select Language Mode

```
user@Paralela: ~/Documents/GitHub/Lab1-Paralela
Valor aproximado de pi: -12.057685
Tiempo promedio de ejecución: 0.000151 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 1.558585
Tiempo promedio de ejecución: 0.000664 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: -11.410956
Tiempo promedio de ejecución: 0.000254 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: -10.384414
Tiempo promedio de ejecución: 0.000330 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: -34.644844
Tiempo promedio de ejecución: 0.000925 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: -17.704699
Tiempo promedio de ejecución: 0.000187 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: -5.024043
Tiempo promedio de ejecución: 0.000172 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
Valor aproximado de pi: 3.143528
Tiempo promedio de ejecución: 0.000730 segundos
user@Paralela:~/Documents/GitHub/Lab1-Paralela$ ./piSeriesNaive
```

En esta última versión, se usaron los siguientes parámetros. N = 3000, 8 hilos y 15 unidades de medición. Aquí es importante mencionar que no tomó tantas corridas poder llegar al resultado aproximado de pi. Sin embargo, es importante mencionar que cada corrida tomaba bastante más tiempo que las versiones anteriores y esto se debe a que la cantidad de iteraciones era mayor.

Inciso b:

En este código existen dos tipos de dependencias: RAW y WAR.

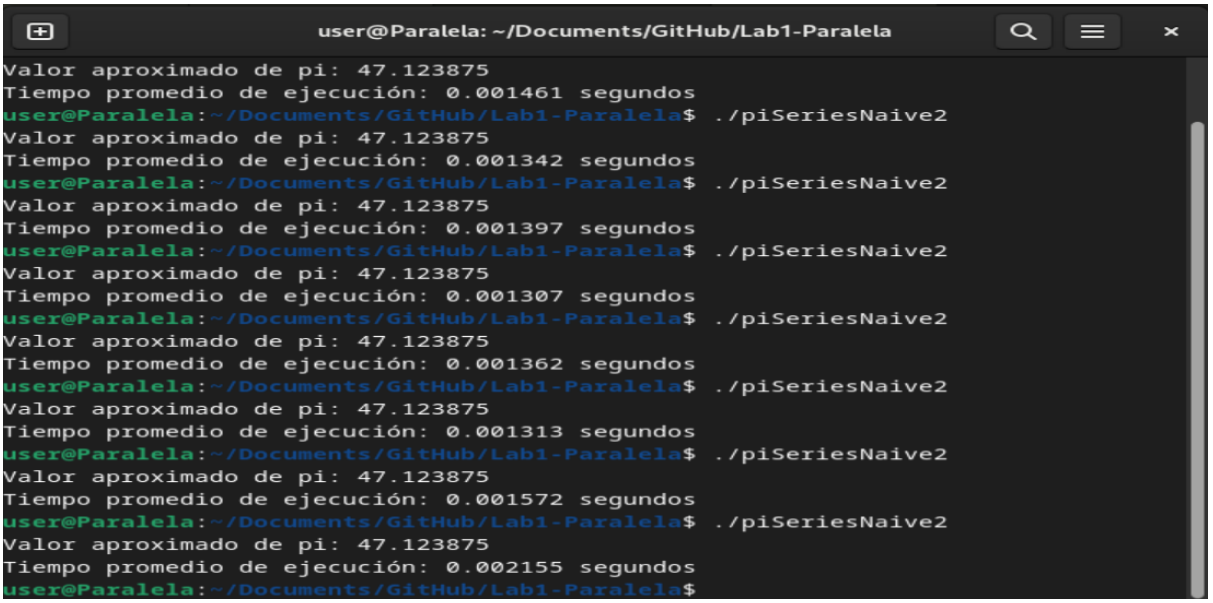
La dependencia RAW se da de la siguiente manera: en la operación sum += factor / (2 * k + 1), dado que sum está usando el valor de factor para poder realizar el cálculo en cada iteración del for. Lo anterior quiere decir que el valor factor es leído y usado en una operación de escritura.

La segunda dependencia, WAR, se da con la siguiente instrucción: factor = -factor en donde el valor de factor está siendo utilizado después de haber sido leído en la línea anterior. Primero se realiza la lectura de factor con la y luego se modifica su valor con su valor opuesto.

Inciso c:

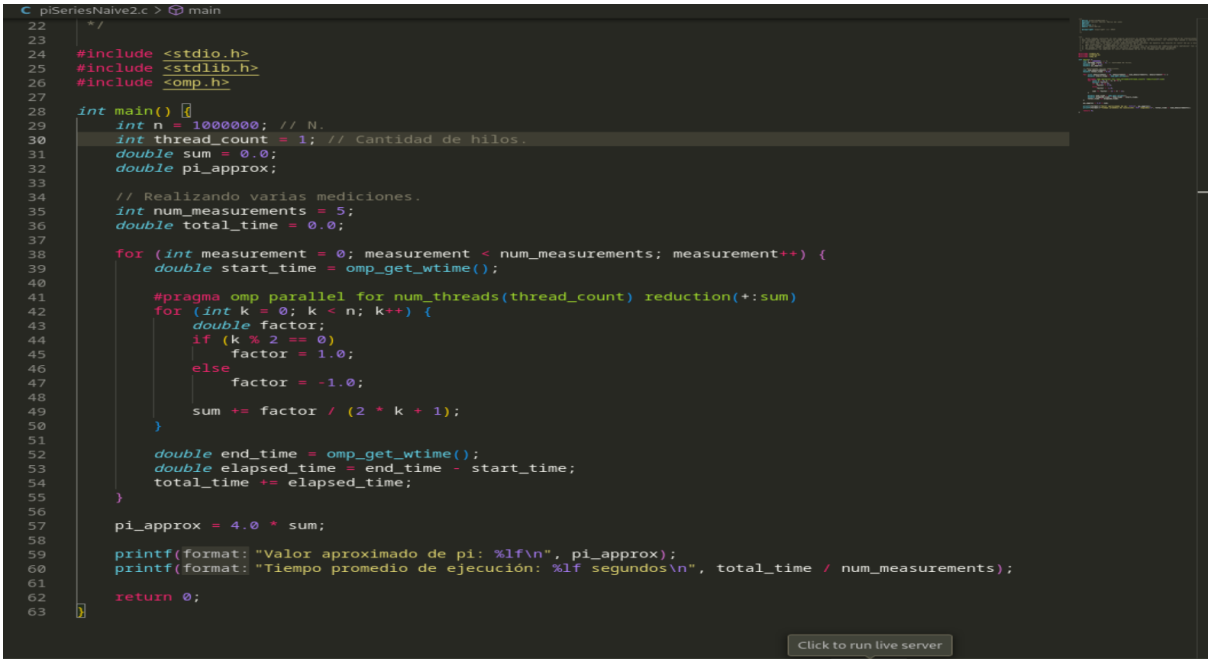
La razón por la cual se hace la operación factor = -factor es porque en la serie de Leibniz va calculando el valor aproximado de pi de la siguiente manera: $\pi = 4 * (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} \ldots)$.

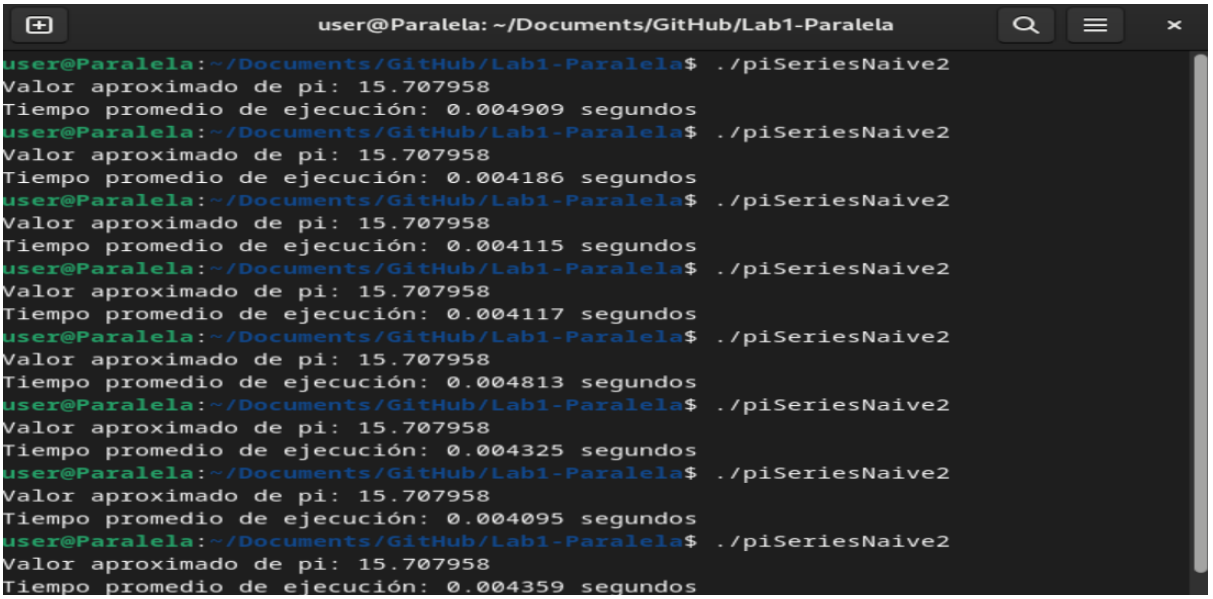
Inciso d:



Se pueden notar que en este programa, con N = 1 000 000, 15 threads y 15 unidades de procesamiento, el valor aproximado de pi subió radicalmente a a 47.123875, lo cual nos indica que al usar una cantidad de iteraciones excesiva y un aumento radical de parámetros como hilos y unidades de procesamiento, el valor aproximado se incrementa por mucho en la ejecución y deja de ser exacto.

Inciso e:





Inciso f:

Como el valor calculado es consistentemente cercano al valor real de π a pesar de los cambios en el código (como la paralelización), significa que las modificaciones en el código para la paralelización no han afectado la precisión del cálculo.

Un indicador directo de la eficiencia de la paralelización es el tiempo de ejecución. Este tiempo mucho más reducido indica que la paralelización ha mejorado el rendimiento del programa.

Inciso g:

10 corridas:

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ gcc -o piSeriesNaive2 piSeriesNaive2.c -fopenmp -lm
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaive2
Run     Secuencial      Paralelo (cores)        Paralelo (2*cores)      Paralelo (n*10 & cores)
--------------------------------------------------------------------------------
1       0.001963        0.000603        0.005440        0.005118
2       0.002026        0.000508        0.001415        0.005013
3       0.002080        0.001116        0.001395        0.005018
4       0.001999        0.000512        0.010025        0.005025
5       0.002014        0.000539        0.008884        0.005060
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaive2
Run     Secuencial      Paralelo (cores)        Paralelo (2*cores)      Paralelo (n*10 & cores)
--------------------------------------------------------------------------------
1       0.001959        0.000588        0.004440        0.005718
2       0.001999        0.000487        0.001399        0.005319
3       0.002075        0.000491        0.002027        0.008801
4       0.001997        0.000491        0.014024        0.005618
5       0.002030        0.003347        0.010649        0.005043
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$
```

Inciso h:

Scheduling	Block Size	Run	Parallel Time	Speedup
static	16	1	0.006356	3.116537
static	16	2	0.004972	3.983936
static	16	3	0.008615	2.299280
static	16	4	0.004942	4.007843
static	16	5	0.005102	3.882904
static	64	1	0.004931	4.016981
static	64	2	0.009305	2.128710
static	64	3	0.005004	3.958588
static	64	4	0.004928	4.019637
static	64	5	0.004914	4.030822
static	128	1	0.004901	4.041422
static	128	2	0.009269	2.137126
static	128	3	0.004978	3.978895
static	128	4	0.004946	4.005185
static	128	5	0.004912	4.033046
dynamic	16	1	0.004939	4.010598
dynamic	16	2	0.005519	3.589372
dynamic	16	3	0.008863	2.234893
dynamic	16	4	0.005007	3.956143
dynamic	16	5	0.005000	3.962098
dynamic	64	1	0.004954	3.998879
dynamic	64	2	0.004961	3.993249
dynamic	64	3	0.009155	2.163699
dynamic	64	4	0.004962	3.992455
dynamic	64	5	0.004973	3.983165
dynamic	128	1	0.004950	4.002103
dynamic	128	2	0.004909	4.035100
dynamic	128	3	0.005911	3.351264
dynamic	128	4	0.008409	2.355704
dynamic	128	5	0.004918	4.027834

guided	16	1	0.004966	3.988924
guided	16	2	0.004975	3.981497
guided	16	3	0.005511	3.594605
guided	16	4	0.005318	3.725037
guided	16	5	0.008506	2.328889
guided	64	1	0.004939	4.011063
guided	64	2	0.005050	3.922856
guided	64	3	0.004976	3.980622
guided	64	4	0.005947	3.331071
guided	64	5	0.004931	4.017062
guided	128	1	0.008557	2.315031
guided	128	2	0.004979	3.978175
guided	128	3	0.004989	3.970446
guided	128	4	0.009273	2.136069
guided	128	5	0.004927	4.020836

auto	1	0.004891	4.049991
auto	2	0.004932	4.016181
auto	3	0.004930	4.017878
auto	4	0.009155	2.163585
auto	5	0.004899	4.043060

La política con mejores resultados fue:

dynamic	128	1	0.004950	4.002103
dynamic	128	2	0.004909	4.035100
dynamic	128	3	0.005911	3.351264

Por un speedup del 4.0351, haciéndola la más eficiente.

3. Optimizaciones y otro acercamiento a la aproximación

Inciso a:

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ gcc -o piSeriesNaiveAlt piSerie
sNaiveAlt.c -fopenmp -lm
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaiveAlt
Run 1:
Método Original:
Tiempo de ejecución: 0.005735 segundos
PI aproximado: 3.141593
Error: 0.000003%

Método Alternativo:
Tiempo de ejecución: 0.006268 segundos
PI aproximado: 3.141593
Error: 0.000003%

====================================

Run 2:
Método Original:
Tiempo de ejecución: 0.005567 segundos
PI aproximado: 3.141593
Error: 0.000003%

Método Alternativo:
Tiempo de ejecución: 0.004922 segundos
PI aproximado: 3.141593
Error: 0.000003%

====================================

Run 3:
Método Original:
Tiempo de ejecución: 0.008627 segundos
PI aproximado: 3.141593
Error: 0.000003%

Método Alternativo:
Tiempo de ejecución: 0.004967 segundos
PI aproximado: 3.141593
Error: 0.000003%
```

```
====================================

Run 4:
Método Original:
Tiempo de ejecución: 0.005758 segundos
PI aproximado: 3.141593
Error: 0.000003%

Método Alternativo:
Tiempo de ejecución: 0.004928 segundos
PI aproximado: 3.141593
Error: 0.000003%

====================================

Run 5:
Método Original:
Tiempo de ejecución: 0.005569 segundos
PI aproximado: 3.141593
Error: 0.000003%

Método Alternativo:
Tiempo de ejecución: 0.005666 segundos
PI aproximado: 3.141593
Error: 0.000003%

====================================
```

Podemos ver que ambos métodos son bastante exactos, compartiendo el mismo porcentaje de error en los cálculos. Sin embargo, la diferencia cae en el tiempo de ejecución. En ocasiones, el método alternativo es más rápido.

Inciso b:

Sin optimización:

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ gcc -fopenmp piSeriesNaiveAlt2.
c -o piSeriesNaiveAlt2 -lm
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaiveAlt2
Tiempo de ejecución: 0.005074 segundos
Aproximación de PI: 3.141592553587436
Porcentaje de error con respecto a PI: 0.00000%
```

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaiveAlt2
Tiempo de ejecución: 0.005058 segundos
Aproximación de PI: 3.141592553587436
Porcentaje de error con respecto a PI: 0.00000%
```

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaiveAlt2
Tiempo de ejecución: 0.006711 segundos
Aproximación de PI: 3.141592553587436
Porcentaje de error con respecto a PI: 0.00000%
```

Con optimización

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaiveAlt2
Tiempo de ejecución: 0.002293 segundos
Aproximación de PI: 3.141592553587440
Porcentaje de error con respecto a PI: 0.00000%
```

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaiveAlt2
Tiempo de ejecución: 0.002309 segundos
Aproximación de PI: 3.141592553587440
Porcentaje de error con respecto a PI: 0.00000%
```

```
daviddm@daviddm-LM:/media/sf_C,C++/Lab1Paralela$ ./piSeriesNaiveAlt2
Tiempo de ejecución: 0.004567 segundos
Aproximación de PI: 3.141592553587436
Porcentaje de error con respecto a PI: 0.00000%
```

¿Cuál fue el speedup observado al usar -O2?

En el primer caso, por ejemplo:
El speedup se calcula como el tiempo de ejecución de la versión sin optimizar dividido entre el tiempo de ejecución de la versión optimizada.
Sp = TsinOp / TO2 = 0.005074/0.002293 = 2.21
Por lo tanto, el speedup es de aproximadamente 2.21.

¿Qué otras optimizaciones se podrían explorar para mejorar aún más el rendimiento?

- Usar banderas de optimización adicionales de gcc, como -O3, que mejora la optimización aún más que -O2.
- Revisar el código fuente para encontrar áreas que puedan refactorizarse para aumentar la eficiencia.
- Usar perfiles de rendimiento para identificar y optimizar los cuellos de botella.

¿Cómo creen que -O2 mejora el rendimiento?
-O2 activa una serie de optimizaciones en el compilador para mejorar el rendimiento sin aumentar el tamaño del binario demasiado. Algunas de las técnicas que emplea incluyen:

- El proceso de eliminación de código muerto elimina el código que no tiene ningún impacto en el resultado del programa.
- Reordenamiento de instrucciones: cambia el orden de las instrucciones para optimizar la arquitectura de la CPU y reducir los tiempos de espera.
- Para reducir el overhead de llamadas, reemplaza las llamadas a funciones con el cuerpo de la función.
- El desenrollado de bucles, que ejecuta el cuerpo del bucle una y otra vez por iteración, reduce el overhead de control del bucle.
- Optimización de la localidad de referencia: reorganiza el acceso a la memoria para maximizar el uso de cachés de CPU.

¿Hay algún caso en el que no sería recomendable usar -O2?

- Durante la fase de depuración, la optimización puede reordenar el código, lo que dificulta la depuración.
- Si la predictibilidad y la consistencia se priorizan sobre el rendimiento máximo.
- Es recomendable desactivar las optimizaciones cuando se sospecha de un error en el compilador para aislar el problema.