

Universidad del Valle de Guatemala	Programación Paralela
Laboratorio 1	Jorge Caballeros Pérez 20009 Alejandra Guzmán

## Laboratorio 1: $\pi$

### Competencias a desarrollar:

Implementa y diseña programas para la paralelización de procesos con memoria compartida usando OpenMP.

### 1. Instrucciones

En equipos de 3 personas, realice las siguiente implementación de un algoritmo secuencial y su transformación en versión paralela usando OpenMP.

**En todos los incisos debe dejar copia (en el PDF a entregar) de lo siguiente:**

- Función(es) principal(es) del código. La parte que contiene el cálculo o tarea más importante. No incluya encabezados ni operaciones de interacción con el usuario.
  - Bloque de código modificado
- Captura de pantalla de la salida (x1)
- Mediciones de tiempo (min 5, usar N o tamaño adecuado)
- Métrica de desempeño: speedup, efficiency

### 2. Estimación del valor de $\pi$

Existen muchas aproximaciones numéricas para estimar el valor de PI. Estas aproximaciones generalmente son series numéricas infinitas que nos permiten calcular PI con mayor precisión según el número de términos de la serie. Para nuestro laboratorio usaremos la siguiente serie:

Para implementar la serie anterior, podemos usar el siguiente algoritmo secuencial;

```

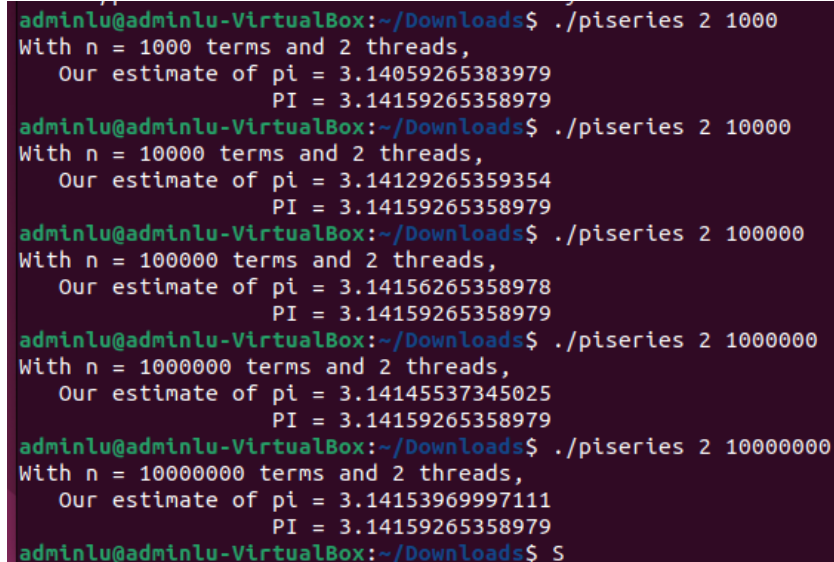
1. double factor = 1.0;
2. double sum = 0.0;
3. for (k = 0; k < n; k++) {
4.   sum += factor/(2*k+1);
5.   factor = -factor;
6. }
7. pi_approx = 4.0*sum;
```

Como primer intento, vamos a agregar una directiva **#pragma omp for** y una cláusula **reduction(+:sum)** para acumular las sumas parciales de la serie numérica.

```
1. double factor = 1.0;
2. double sum = 0.0;
3. #pragma omp parallel for num_threads(thread_count) \
   reduction(+:sum)
4. for (k = 0; k < n; k++) {
5.     sum += factor/(2*k+1);
6.     factor = -factor;
7. }
8. pi_approx = 4.0*sum;
```

a. (5 pts) Implemente la version secuencial y pruebe que esté correcta (piSeriesSeq.c). Implemente la versión paralela mencionada (piSeriesNaive.c) , compílelo y ejecútelo. Realice al menos 5 mediciones del valor con *threads*  $\geq 2$  y *n*  $\geq 1000$  (pruebe ir incrementando, registre todos los números resultantes). Describa lo que sucede con el resultado respecto al valor preciso de PI (3.1415926535 8979323846).

Screenshot de la ejecución:



```
adminlu@adminlu-VirtualBox:~/Downloads$ ./piseriess 2 1000
With n = 1000 terms and 2 threads,
  Our estimate of pi = 3.14059265383979
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./piseriess 2 10000
With n = 10000 terms and 2 threads,
  Our estimate of pi = 3.14129265359354
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./piseriess 2 100000
With n = 100000 terms and 2 threads,
  Our estimate of pi = 3.14156265358978
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./piseriess 2 1000000
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14145537345025
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./piseriess 2 10000000
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14153969997111
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ s
```

b. (5 pts) Analice el código fuente de piSeriesNaive.c. Identifique el tipo de dependencia que se da con la variable **factor**.

**Respuesta:**

El código tiene una dependencia de datos en la variable factor, la cual se encuentra en un bucle for y depende del factor previo de "factor" en cada iteración.

c. (5 pts) Observe el algoritmo y la serie numérica. Describa en sus propias palabras la razón por la cual **factor = - factor**.

**Respuesta:**

El primer término es positivo, el segundo es negativo, el tercero es positivo nuevamente, y así sucesivamente. Para lograr este patrón de signos alternos, el código multiplica el valor de factor por -1 en cada iteración del bucle. De esta forma, el valor de factor alterna entre 1 y -1, lo que permite obtener el signo correcto para cada término de la serie.

d. (5 pts) Para eliminar la dependencia de loop, debemos modificar la forma como calculamos el valor **factor**. Guarde una copia del programa anterior y reemplace el siguiente segmento de código. Realice al menos 5 mediciones del valor con *threads*  $\geq 2$  y *n*  $\geq 10e6$  (registre todos los números

<b>sum += factor/(2*k+1);</b> <b>factor = -factor;</b>	if (k % 2 == 0)  factor = 1.0;  else  factor = -1.0;  sum += factor/(2*k+1);
(versión con operador ternario)	factor = (k % 2 == 0) ? 1.0 : -1.0;  sum += factor/(2*k+1);

resultantes). Describa lo que sucede con el resultado respecto al valor preciso de PI (3.1415926535 8979323846):

**Respuesta:**

Al eliminar la dependencia la estimación de pi es más precisa y de esta forma se acerca más al valor real de pi a medida que aumenta en términos de n.

Hemos eliminado la race condition en la variable factor y por lo tanto, el algoritmo es mejor para usarlo en paralelo.

```
adminlu@adminlu-VirtualBox: ~/Downloads
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.13862968083545
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 2 1000000
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14118546715530
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 2 10000000
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14156688667683
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 2 100000000
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.14163459921238
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 2 1000000000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.09531821925989
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 2 10000000000
With n = 10000000000 terms and 2 threads,
  Our estimate of pi = 3.14894542498555
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$
```

e. (10 pts) Ejecute el mismo código pero threads = 1 y realice al menos 5 mediciones (registre todos los números resultantes). Describa en sus propias palabras la razón por la cual el resultado es diferente.

```
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 1 1000000
With n = 1000000 terms and 1 threads,
  Our estimate of pi = 3.14159165358977
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 1 10000000
With n = 10000000 terms and 1 threads,
  Our estimate of pi = 3.14159255358979
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 1 100000000
With n = 100000000 terms and 1 threads,
  Our estimate of pi = 3.14159264358933
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 1 1000000000
With n = 1000000000 terms and 1 threads,
  Our estimate of pi = 3.14159265258805
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ ./pi 1 10000000000
```

f. (10 pts) Debemos cambiar el ámbito (scope) de una variable para resolver el problema que pueda darse respecto a los resultados en la versión paralela con threads > 1. Modifique el programa usando la cláusula de cambio de scope private(). Realice al menos 5 mediciones del valor con threads >= 2 y n >= 10e6 (registre todos los números resultantes). Describa lo que sucede con el resultado respecto al valor preciso de PI (3.1415926535

tiempo secuencial (threads = 1)	tiempo paralelo (threads = cores)	tiempo paralelo (threads = 2*cores)	tiempo paralelo (n = n*10 && threads = cores)
No hay paralelismo y cada iteración se ejecuta una tras otra. Garantiza que no hayan race conditions o inconsistencias.	Las iteraciones se dividen entre los núcleos disponibles, esto acelera el calculo pero puede verse afectada la precisión.	Al tener más hilos, algunos deben esperar para ser ejecutados, y ralentiza el resultado.	El paralelismo ayuda a acelerar el cálculo pero lleva más tiempo con un n más pequeño.

8979323846). Incluya una captura de pantalla del resultado final.g. (15 pts) Use la última versión paralela con n = 10e6 (o más si e6 es poco para la computadora de cada quien) y el número de hilos según la cantidad de cores de su sistema (i.e: nproc). Realice el cálculo de **speedup**, **eficiencia**, **escalabilidad fuerte** y **escalabilidad débil** para las siguientes condiciones (solamente modifique un parámetro a la vez). Tome por lo menos 5 medidas para sus datos:

h. (15 pts) Usando la versión final de su programa paralelo, modificarlo y pruebe las diferentes políticas de planificación y block\_size. Registre sus datos y calcule las diferencias en speedup para cada uno de los mecanismos de scheduling (**static**, **dynamic**, **guided**, **auto**) usando los siguientes parámetros: n = 10e6 (o más si aplica), threads = cores, probar block\_size de 16, 64, 128 (en todos menos auto). Tome por lo menos 5 medidas de cada una. (tip: recuerde la opción runtime que permite pasarle el tipo para agilizar el proceso) Con cuál política de planificación obtuvo mejores resultados?

### Bloques de 16:

Static:

```
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,16" ./run 2 1000000
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14152579616681
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,16" ./run 2 10000000
0
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14157690039589
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,16" ./run 2 100000000
00
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.12798318888324
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,16" ./run 2 1000000000
000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.12431618730773
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,16" ./run 2 10000000000
0000
```

Dynamic:

```
adminlu@adminlu-VirtualBox: ~/Downloads
^C
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,16" ./run 2 100000
0
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.12712247887790
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,16" ./run 2 1000000
00
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.10401331134613
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,16" ./run 2 100000000
000
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.08342546305799
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,16" ./run 2 1000000000
0000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.06411683928163
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,16" ./run 2 10000000000
00000
```

Guided:

```
adminlu@adminlu-VirtualBox: ~/Downloads
00000
^C
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,16" ./run 2 1000000
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14160095901450
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,16" ./run 2 1000000
0
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14145133472699
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,16" ./run 2 1000000
00
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.14159912137732
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,16" ./run 2 1000000
000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.14985786316444
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,16" ./run 2 1000000
0000
```

Auto:

```
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,16" ./run 2 1000000
0000
^C
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,16" ./run 2 1000000
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14157642132624
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,16" ./run 2 10000000
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14159240963639
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,16" ./run 2 100000000
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.14167109867474
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,16" ./run 2 1000000000
0
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.14548329710439
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,16" ./run 2 1000000000
00
```

**Conclusión bloque 16:** El que obtuvo mejores resultados fue dynamic al ser el más rápido y certero.

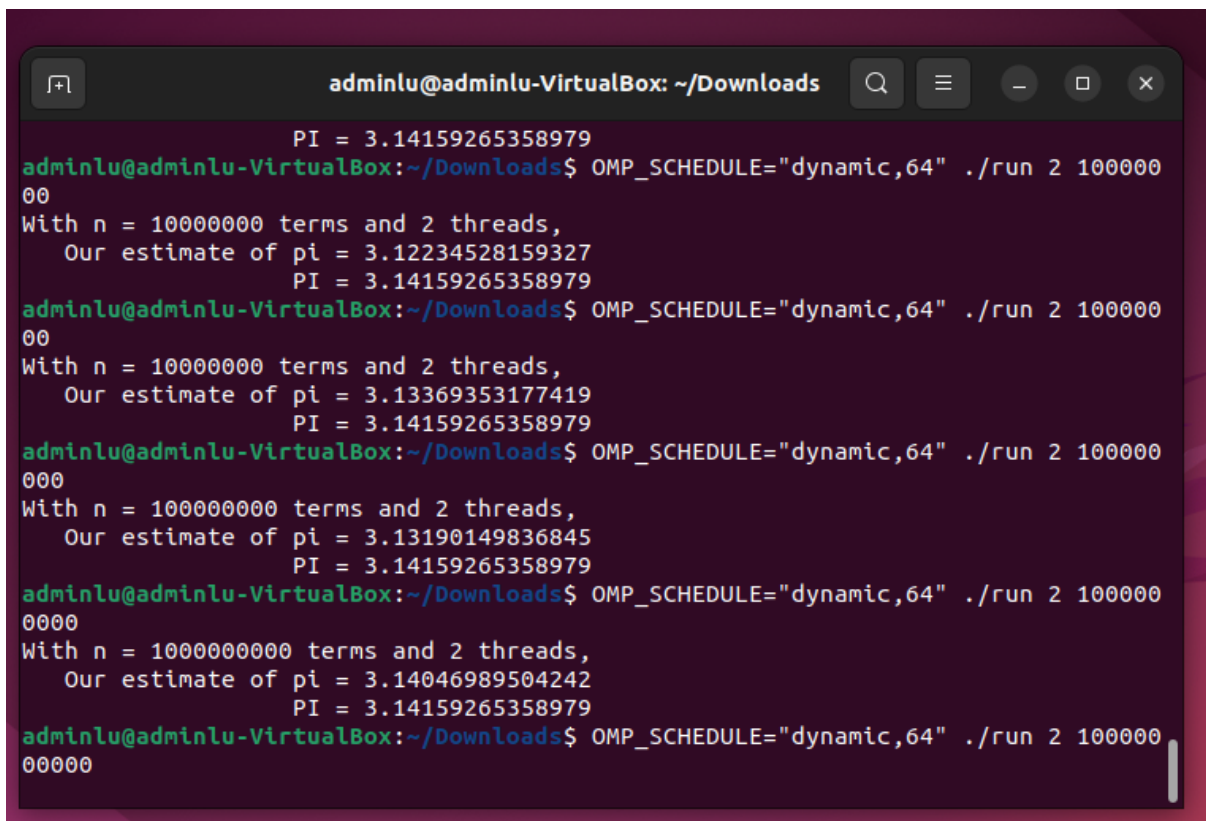
**Bloques de 64:**

STATIC:

```
adminlu@adminlu-VirtualBox: ~/Downloads
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,64" ./run 2 10000
With n = 10000 terms and 2 threads,
Our estimate of pi = 3.14149265359005
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,64" ./run 2 100000
With n = 100000 terms and 2 threads,
Our estimate of pi = 3.14158265358975
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,64" ./run 2 1000000
With n = 1000000 terms and 2 threads,
Our estimate of pi = 3.14161232712877
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,64" ./run 2 10000000
0
With n = 10000000 terms and 2 threads,
Our estimate of pi = 3.14159238645349
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static,64" ./run 2 100000000
00
With n = 100000000 terms and 2 threads,
Our estimate of pi = 3.12225953004313
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ s
```



Dynamic:



```
adminlu@adminlu-VirtualBox: ~/Downloads
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,64" ./run 2 100000
00
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.12234528159327
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,64" ./run 2 100000
00
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.13369353177419
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,64" ./run 2 100000
000
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.13190149836845
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,64" ./run 2 100000
0000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.14046989504242
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic,64" ./run 2 100000
00000
```

Guided:

```

adminlu@adminlu-VirtualBox: ~/Downloads
With n = 1000000 terms and 2 threads,
    Our estimate of pi = 3.14159930041518
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,64" ./run 2 1000000
0
With n = 10000000 terms and 2 threads,
    Our estimate of pi = 3.14172887117790
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,64" ./run 2 10000000
00
With n = 100000000 terms and 2 threads,
    Our estimate of pi = 3.14155189967191
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,64" ./run 2 100000000
000
With n = 1000000000 terms and 2 threads,
    Our estimate of pi = 3.14843374486291
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided,64" ./run 2 1000000000
0000
With n = 10000000000 terms and 2 threads,
    Our estimate of pi = 3.16221546306917
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$

```

Auto:

```

adminlu@adminlu-VirtualBox: ~/Downloads
0000
With n = 10000000000 terms and 2 threads,
    Our estimate of pi = 3.16221546306917
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,64" ./run 2 1000000
With n = 1000000 terms and 2 threads,
    Our estimate of pi = 3.14158408655284
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,64" ./run 2 10000000
With n = 10000000 terms and 2 threads,
    Our estimate of pi = 3.14159680961514
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,64" ./run 2 100000000
With n = 100000000 terms and 2 threads,
    Our estimate of pi = 3.14428556343983
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,64" ./run 2 1000000000
0
With n = 1000000000 terms and 2 threads,
    Our estimate of pi = 3.14619419072801
    PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto,64" ./run 2 1000000000
00

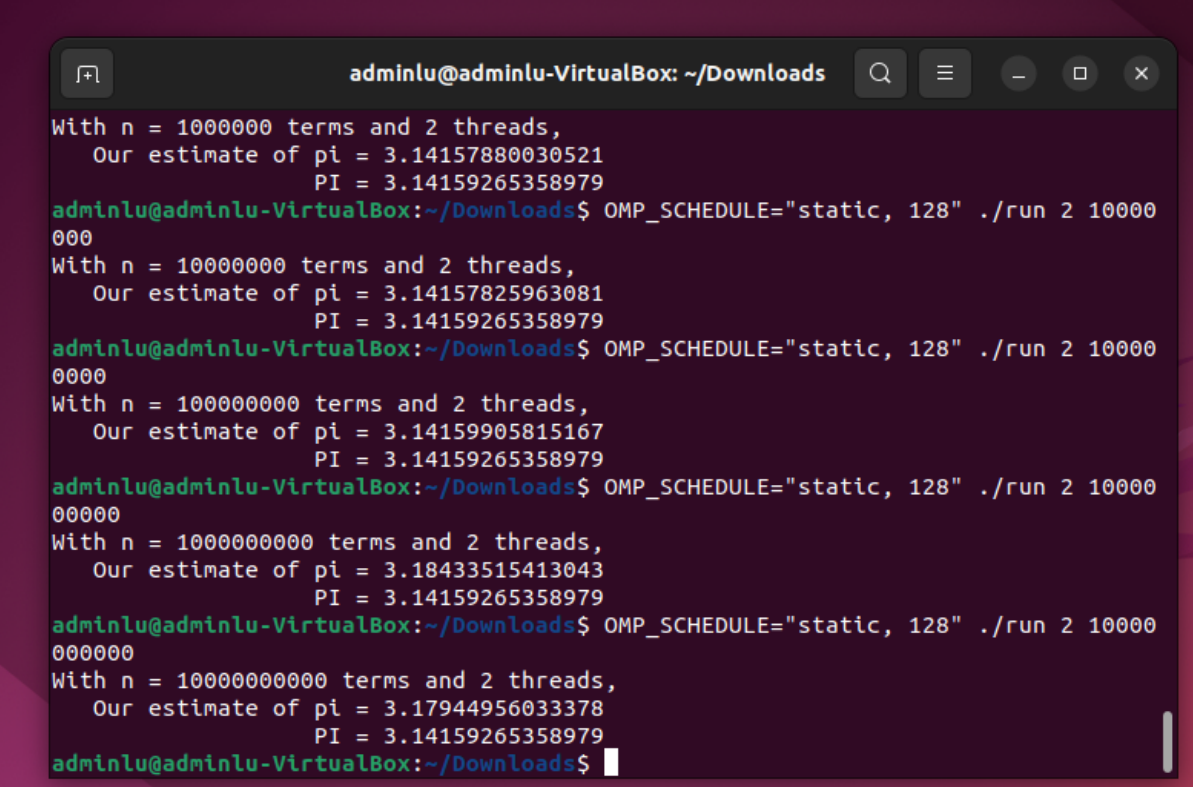
```

### **Conclusión de bloque 64:**

El que dio mejores resultados fue guided, debido a que si puedo procesar 10 ceros, y además tuvo buenos resultados y apegados.

### **Bloques de 128:**

Static:



```
adminlu@adminlu-VirtualBox: ~/Downloads
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14157880030521
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static, 128" ./run 2 10000
000
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14157825963081
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static, 128" ./run 2 10000
0000
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.14159905815167
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static, 128" ./run 2 10000
00000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.18433515413043
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="static, 128" ./run 2 10000
000000
With n = 10000000000 terms and 2 threads,
  Our estimate of pi = 3.17944956033378
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$
```

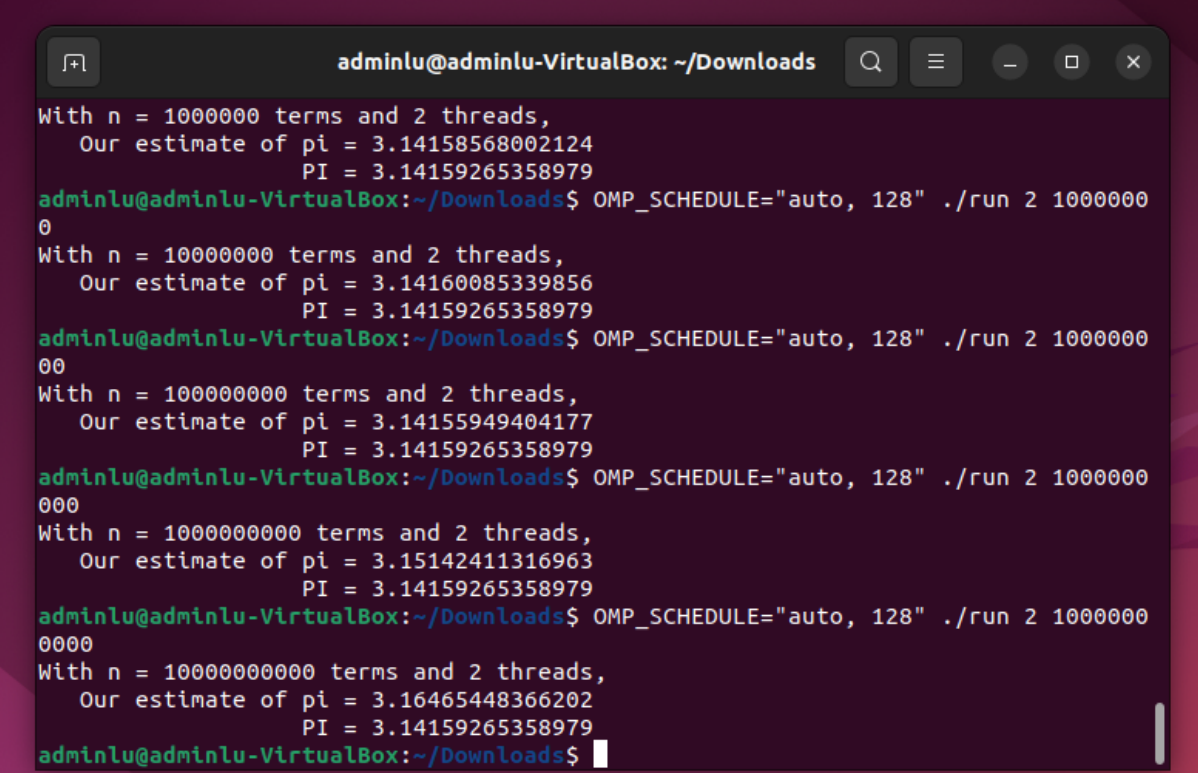
Dynamic:

```
adminlu@adminlu-VirtualBox: ~/Downloads
PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic, 128" ./run 2 1000
000
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.13977968881163
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic, 128" ./run 2 1000
0000
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.13558240214599
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic, 128" ./run 2 1000
00000
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.18193800682304
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic, 128" ./run 2 1000
000000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.13228155354289
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="dynamic, 128" ./run 2 1000
0000000
```

Guided:

```
adminlu@adminlu-VirtualBox: ~/Downloads
^C
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided, 128" ./run 2 10000
00
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14159743907406
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided, 128" ./run 2 10000
000
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14162238584635
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided, 128" ./run 2 10000
0000
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.15345651607241
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided, 128" ./run 2 10000
00000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.13441201975959
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="guided, 128" ./run 2 10000
000000
```

Auto:



```
adminlu@adminlu-VirtualBox: ~/Downloads
With n = 1000000 terms and 2 threads,
  Our estimate of pi = 3.14158568002124
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto, 128" ./run 2 1000000
0
With n = 10000000 terms and 2 threads,
  Our estimate of pi = 3.14160085339856
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto, 128" ./run 2 1000000
00
With n = 100000000 terms and 2 threads,
  Our estimate of pi = 3.14155949404177
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto, 128" ./run 2 1000000
000
With n = 1000000000 terms and 2 threads,
  Our estimate of pi = 3.15142411316963
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$ OMP_SCHEDULE="auto, 128" ./run 2 1000000
0000
With n = 10000000000 terms and 2 threads,
  Our estimate of pi = 3.16465448366202
  PI = 3.14159265358979
adminlu@adminlu-VirtualBox:~/Downloads$
```

**Conclusión bloque 128:** Auto obtuvo mejor performance a comparación de los otros.

### 3. Optimizaciones y otro acercamiento a la aproximación

Podemos notar que la serie mencionada sigue un patrón específico: suma los elementos con índices pares y resta los que tienen índices impares. Podemos agrupar la sumatoria de la siguiente manera:

$$\pi = 4 \left[ \sum_{i \in \text{Even}}^{\infty} \frac{1}{2i+1} - \sum_{j \in \text{Odd}}^{\infty} \frac{1}{2j+1} \right] \quad (2)$$

Siguiendo este planteamiento alternativo (dos sumatorias versus una) podemos evitar el tener que calcular el factor  $1/-1$  en cada paso del ciclo for, ahorrando 1 instrucción (de 2) por ciclo (la mitad).

- a. (15 pts.) Implemente el programa descrito por la ecuación anterior (**piSeriesAlt.c**) , compílelo y ejecútelo. Realice al menos 5 mediciones del valor con *threads*  $\geq 2$  y *n*  $\geq 10e6$  o adecuado (registre todos los números resultantes). Describa lo que sucede con el resultado respecto al valor preciso de PI (3.1415926535 8979323846). Haga una comparación con los mismos parámetros (threads, n) de esta versión y su mejor versión del inciso h.

#### Resultados:

```
● Introduce el valor de n: 10
Aproximaci|n de pi usando 10 t|-rminos: 3.14159265358979222782
PS C:\Users\jorge\OneDrive\Documentos\GitHub\Laboratorio1-Paralela\main> ./machin_pi
● Introduce el valor de n: 100
Aproximaci|n de pi usando 100 t|-rminos: 3.14159265358979400418
PS C:\Users\jorge\OneDrive\Documentos\GitHub\Laboratorio1-Paralela\main> ./machin_pi
● Introduce el valor de n: 1000
Aproximaci|n de pi usando 1000 t|-rminos: 3.14159265358979400418
PS C:\Users\jorge\OneDrive\Documentos\GitHub\Laboratorio1-Paralela\main> ./machin_pi
Introduce el valor de n: 10000
Aproximaci|n de pi usando 10000 t|-rminos: 3.14159265358979400418
● PS C:\Users\jorge\OneDrive\Documentos\GitHub\Laboratorio1-Paralela\main> ./machin_pi
Introduce el valor de n: 100000
Aproximaci|n de pi usando 100000 t|-rminos: 3.14159265358979400418
○ PS C:\Users\jorge\OneDrive\Documentos\GitHub\Laboratorio1-Paralela\main> █
```

R // La fórmula de Machin combinada con la serie de Taylor para  $\arctan(x)$  proporciona una aproximación efectiva de pi. Sin embargo, al utilizar el tipo ``double`` en C++, se observa que la precisión de la aproximación se estanca después de cierto número de términos, alcanzando el límite inherente de precisión de 15-17 dígitos decimales de este tipo de dato. Aunque la aproximación es cercana al valor real de pi, para una precisión superior sería necesario emplear herramientas que soporten aritmética de precisión arbitraria.

b. (15 pts.) Como se mencionó en clase, los compiladores (como gcc/g++) tienen muchas opciones de las cuales incluyen optimizaciones de código. En términos de speedup, lo que nos interesa son las opciones que priorizan tiempo de ejecución sobre tamaño del programa/código o facilidad para debugeo (sugerencia de lectura: <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>). Pruebe compilar su mejor versión al momento pero esta vez agregando la opción de optimización “-O2”. Mida varias veces el tiempo de ejecución y compare con la versión sin la bandera de optimización. ¿Qué pudieron observar? Comenten entre el grupo e incluyan un resumen de su discusión.

Sin Optimización:

```
Aproximaci| n de pi usando 10000 t|-rminos: 3.14159265358979400418
PS C:\Users\jorge\OneDrive\Documentos\Github\Laboratorio1-Paralela\main> $before = Get-Date; ./no_opt.exe; $after = Get-Date; $after - $before
Aproximaci| n de pi usando 10 t|-rminos: 3.14159265358979222782
Aproximaci| n de pi usando 100 t|-rminos: 3.14159265358979400418
Aproximaci| n de pi usando 1000 t|-rminos: 3.14159265358979400418
Aproximaci| n de pi usando 10000 t|-rminos: 3.14159265358979400418
Aproximaci| n de pi usando 100000 t|-rminos: 3.14159265358979400418

Days          : 0
Hours         : 0
Minutes       : 0
Seconds       : 0
Milliseconds   : 45
Ticks         : 458755
TotalDays     : 5,30966435185185E-07
TotalHours    : 1,27431944444444E-05
TotalMinutes  : 0,000764591666666667
TotalSeconds  : 0,0458755
TotalMilliseconds : 45,8755
```

Con Optimización:

```
PS C:\Users\jorge\OneDrive\Documentos\Github\Laboratorio1-Paralela\main> $before = Get-Date; ./opt.exe; $after = Get-Date; $after - $before
Aproximaci| n de pi usando 10 t|-rminos: 3.14159265358979222782
Aproximaci| n de pi usando 100 t|-rminos: 3.14159265358979400418
Aproximaci| n de pi usando 1000 t|-rminos: 3.14159265358979400418
Aproximaci| n de pi usando 10000 t|-rminos: 3.14159265358979400418
Aproximaci| n de pi usando 100000 t|-rminos: 3.14159265358979400418

Days          : 0
Hours         : 0
Minutes       : 0
Seconds       : 0
Milliseconds   : 50
Ticks         : 500824
TotalDays     : 5,79657407407407E-07
TotalHours    : 1,39117777777778E-05
TotalMinutes  : 0,000834706666666667
TotalSeconds  : 0,0500824
TotalMilliseconds : 50,0824
```

1. Resultados de Aproximación: En ambos casos (con y sin optimización), la aproximación de pi para diferentes valores de n es idéntica. Esto muestra que la optimización no cambia la precisión del cálculo, lo cual es esperado y deseado.

2. Tiempo de Ejecución: Aunque hay una diferencia en el tiempo de ejecución entre la versión no optimizada y la optimizada, esta es pequeña. La versión no optimizada tardó aproximadamente 45,88 milisegundos, mientras que la versión optimizada tardó aproximadamente 50,08 milisegundos. Contrario a lo que podríamos esperar, la versión optimizada tomó un poco más de tiempo. Sin embargo, esta diferencia es mínima y puede deberse a factores externos o variaciones normales en la ejecución.



3. Implicaciones de la Optimización: La bandera ``-O2`` introduce optimizaciones que en muchos casos pueden mejorar el rendimiento. Sin embargo, no garantiza siempre una ejecución más rápida. La naturaleza de las optimizaciones y cómo interactúan con el código específico pueden producir resultados variados. Además, la optimización podría tener un impacto más notable en programas más grandes o complejos.

Conclusión:

La optimización ``-O2`` no tuvo un impacto significativo en el tiempo de ejecución para este programa específico. Sin embargo, es crucial comprender que el efecto de las optimizaciones del compilador puede variar según el código y el contexto. En este caso, los tiempos de ejecución son muy similares, lo que sugiere que la naturaleza computacional del programa no se benefició significativamente de las optimizaciones introducidas por ``-O2``. Es esencial medir y no asumir que las optimizaciones siempre resultarán en una ejecución más rápida.

#### **4. Entrega: 18 agosto 19:50. Incluir:**

- Sus respuestas a los incisos requeridos en formato PDF.
- Código fuente (.c/.cpp) de programas finales (secuencial, paralelo, alternativo)
- Instrucciones de compilación (o bien un makefile)