

Submit II

Juan Diego Collazos
Juan Sebastián Garizao

1. Compilación con y sin OpenMP

Por un lado, al compilar sin OpenMP, el compilador ignora la línea:

```
#pragma omp parallel
```

El programa se ejecuta con un único hilo y el bloque paralelo se ejecuta una sola vez. En consecuencia, la salida contiene un “Hello” y una línea:

```
GoodBye -PUJPP- Exiting Program
```

Por otro lado, con la opción `-fopenmp`, la directiva crea una región paralela: se crea una cantidad N de hilos y cada hilo ejecuta el mismo bloque. Por eso, la salida contiene N ocurrencias de “Hello” y N líneas de “GoodBye...”, donde N es el número de hilos creados.

```
japeto@2ecec902fd5:~/app$ gcc -O0 -g hello_omp.c -o hello_seq
japeto@2ecec902fd5:~/app$ ./hello_seq
Hello
GoodBye -PUJPP- Exiting Program
japeto@2ecec902fd5:~/app$ gcc -O0 -g -fopenmp hello_omp.c -o hello_omp
japeto@2ecec902fd5:~/app$ OMP_NUM_THREADS=4 ./hello_omp | tee hello_omp_4.txt
Hello
GoodBye -PUJPP- Exiting Program
Hello
GoodBye -PUJPP- Exiting Program
Hello
GoodBye -PUJPP- Exiting Program
Hello
GoodBye -PUJPP- Exiting Program
Hello
GoodBye -PUJPP- Exiting Program
```

Figura 1: Caption

2. Número de hilos

Sí es posible especificar el número de hilos que se desea crear en el “team” mediante la función:

```
omp_set_num_threads(nthreads);
```

En este caso, si se fija `nthreads = 2`, el programa creará exactamente dos hilos, y cada uno de ellos ejecutará la sección paralela. Al cambiar `nthreads` a 4 u 8, se observa que la cantidad de mensajes impresos se corresponde con el número de hilos solicitados, es decir, se generan 4 u 8 repeticiones del bloque paralelo.

¿Se pueden usar números impares de hilos?

Sí, es totalmente posible indicar un número impar de hilos (por ejemplo, 3 o 5). OpenMP no impone ninguna restricción en cuanto a la paridad del número de hilos; la librería simplemente crea la cantidad exacta que se le solicita. Sin embargo, el beneficio de hacerlo depende de la naturaleza del problema. Si el trabajo a realizar se reparte de forma no divisible entre los hilos, algunos procesadores quedarán con más carga que otros, generando un cierto *desbalance* en la ejecución.

¿Se puede especificar un número de hilos mayor al disponible en hardware?

También es posible solicitar más hilos de los que hay disponibles en el hardware (más que los núcleos físicos o lógicos del procesador). En este caso, OpenMP creará los hilos, pero el sistema operativo los planificará sobre los núcleos existentes. Esto significa que varios hilos compartirán el mismo núcleo, alternando su ejecución mediante *time-sharing*. Aunque esto garantiza la ejecución correcta, no necesariamente implica mayor eficiencia. Al contrario, al aumentar excesivamente el número de hilos respecto al número de núcleos disponibles, aparecen sobrecostos de planificación y conmutación de contexto, lo que puede incluso degradar el rendimiento.

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./hello_set_omp 2
Hello Hello
GoodBye -PUJPP- Exiting Program
japeto@aa642b9ad45a:/home/workspace/scripts$ ./hello_set_omp 3
Hello Hello Hello
GoodBye -PUJPP- Exiting Program
japeto@aa642b9ad45a:/home/workspace/scripts$ ./hello_set_omp 4
Hello Hello Hello Hello
GoodBye -PUJPP- Exiting Program
japeto@aa642b9ad45a:/home/workspace/scripts$ ./hello_set_omp 8
Hello Hello Hello Hello Hello Hello Hello Hello
GoodBye -PUJPP- Exiting Program
```

Figura 2: Caption

3. Identificador de hilos

La función `omp_get_thread_num()` permite recuperar el identificador (ID) de cada hilo dentro del equipo creado por OpenMP. La numeración comienza desde 0, asignado al **hilo maestro**, y continúa de manera consecutiva para los demás hilos: 1, 2, 3, ..., `nthreads-1`.

Al reemplazar la instrucción de impresión del ejercicio anterior con:

```
printf("Hello thread = %d", omp_get_thread_num());
```

cada hilo imprimirá su propio identificador junto al mensaje "Hello".

Ejemplo de ejecución

Si se especifica `nthreads = 2`, la salida puede ser algo como:

```
Hello thread = 0
Hello thread = 1
GoodBye -PUJPP- Exiting Program
```

Con `nthreads = 4`, los IDs irán de 0 a 3, y así sucesivamente.

Orden de ejecución

Es importante notar que el orden en que los hilos muestran sus mensajes no está garantizado. Esto se debe a que el planificador del sistema operativo distribuye los hilos sobre los núcleos disponibles, y cada hilo puede avanzar en su ejecución a un ritmo distinto.

Por ejemplo, al ejecutar varias veces con `nthreads = 4`, se pueden observar salidas como:

```
Hello thread = 0
Hello thread = 1
Hello thread = 2
Hello thread = 3
```

o en otro intento:

```
Hello thread = 2
Hello thread = 0
Hello thread = 3
Hello thread = 1
```

El contenido es el mismo (los hilos 0 a 3 imprimen su mensaje), pero el **orden de aparición es no determinista**.

Caso especial: ejecución secuencial

Si se establece `nthreads = 1`, el programa vuelve a ejecutarse de manera secuencial con un único hilo. En este caso, el único mensaje proviene del hilo con `ID = 0`, que corresponde al hilo maestro. Esto es equivalente a la ejecución sin OpenMP.

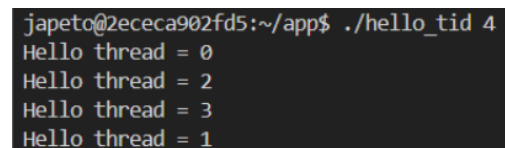
Nociones importantes

- El hilo maestro siempre tiene `ID = 0`.
- Los demás hilos reciben identificadores consecutivos hasta `nthreads-1`.
- El orden en que los mensajes aparecen en pantalla varía en cada ejecución, lo cual refleja la naturaleza concurrente y no determinista de la planificación de hilos.
- Con `nthreads = 1` se vuelve al modo secuencial, útil para comparar comportamientos.



```
japeto@2ecec902fd5:~/app$ ./hello_tid 2
Hello thread = 0
Hello thread = 1
```

Figura 3: Execution 1 - script 3



```
japeto@2ecec902fd5:~/app$ ./hello_tid 4
Hello thread = 0
Hello thread = 2
Hello thread = 3
Hello thread = 1
```

Figura 4: Execution 2 - script 3

```
japeto@2ecec902fd5:~/app$ ./hello_tid 8
Hello thread = 7
Hello thread = 0
Hello thread = 4
Hello thread = 1
Hello thread = 5
Hello thread = 2
Hello thread = 3
Hello thread = 6
GoodBye -PUJPP- Exiting Program
```

Figura 5: Execution 3 - script 3

```
japeto@2ecec902fd5:~/app$ ./hello_tid 16
Hello thread = 1
Hello thread = 6
Hello thread = 7
Hello thread = 8
Hello thread = 10
Hello thread = 13
Hello thread = 5
Hello thread = 15
Hello thread = 12
Hello thread = 2
Hello thread = 9
Hello thread = 3
Hello thread = 4
Hello thread = 14
Hello thread = 0
Hello thread = 11
GoodBye -PUJPP- Exiting Program
```

Figura 6: Execution 4 - script 3

4. Distribución de iteraciones en OpenMP

En OpenMP, la directiva:

```
#pragma omp parallel for
```

divide automáticamente las iteraciones de un bucle entre los hilos disponibles del equipo. A diferencia del modelo con `fork()`, donde el programador debía separar manualmente el trabajo de cada proceso, OpenMP reparte las iteraciones de forma **contigua** y lo más equilibrada posible entre los hilos.

Caso general

Si el número total de iteraciones N es divisible exactamente por el número de hilos `nthreads`, cada hilo recibe el mismo número de iteraciones. Por ejemplo, con $N = 16$ y `nthreads = 4`, cada hilo ejecuta $16/4 = 4$ iteraciones.

Cuando N no es divisible por `nthreads`, OpenMP asigna a los primeros hilos una iteración extra. Esto se hace aplicando la función piso:

$$\text{iteraciones por hilo} = \lfloor N/\text{nthreads} \rfloor$$

y distribuyendo el residuo entre los primeros hilos.

Ejemplos de ejecución

- **2 hilos**, $N = 16$: cada hilo recibe 8 iteraciones (división exacta).
- **4 hilos**, $N = 16$: cada hilo recibe 4 iteraciones.
- **8 hilos**, $N = 16$: cada hilo recibe 2 iteraciones.
- **16 hilos**, $N = 16$: cada hilo recibe exactamente 1 iteración.
- **5 hilos**, $N = 16$: se asignan 3 iteraciones a cada hilo como base, y el residuo de 1 iteración se reparte en los primeros hilos. Por lo tanto, algunos hilos ejecutan 4 iteraciones y otros sólo 3.
- **7 hilos**, $N = 16$: cada hilo recibe al menos 2 iteraciones, y los 2 hilos iniciales reciben una extra (3 en total).

Observaciones sobre el orden de ejecución

Aunque las iteraciones se reparten de forma contigua, el orden en que aparecen impresadas en pantalla puede variar debido a la concurrencia. Cada hilo procesa sus asignaciones de manera independiente y el sistema operativo decide cuál avanza primero. Por eso, las salidas en consola muestran las iteraciones desordenadas en términos de numeración, pero sí se puede verificar que la carga de trabajo corresponde con el reparto descrito.

OpenMP facilita el paralelismo en bucles al dividir automáticamente las iteraciones. El reparto es:

- Equilibrado cuando N es múltiplo de `nthreads`.
- Casi equilibrado cuando no lo es, asignando iteraciones extra a los primeros hilos.

Esto garantiza una buena distribución de trabajo sin intervención manual del programador. Las imágenes obtenidas de las ejecuciones con 2, 4, 5, 7, 8 y 16 hilos ilustran claramente esta política de reparto.

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./for_trace 16 2
thread number: 0, iteracion = 0
thread number: 1, iteracion = 0
thread number: 2, iteracion = 0
thread number: 3, iteracion = 0
thread number: 4, iteracion = 0
thread number: 5, iteracion = 0
thread number: 6, iteracion = 0
thread number: 7, iteracion = 0
thread number: 8, iteracion = 1
thread number: 9, iteracion = 1
thread number: 10, iteracion = 1
thread number: 11, iteracion = 1
thread number: 12, iteracion = 1
thread number: 13, iteracion = 1
thread number: 14, iteracion = 1
thread number: 15, iteracion = 1
```

Figura 7: Hilos 2 - script 4

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./for_trace 16 4
thread number: 0, iteracion = 0
thread number: 1, iteracion = 0
thread number: 2, iteracion = 0
thread number: 3, iteracion = 0
thread number: 8, iteracion = 2
thread number: 9, iteracion = 2
thread number: 10, iteracion = 2
thread number: 11, iteracion = 2
thread number: 12, iteracion = 3
thread number: 13, iteracion = 3
thread number: 14, iteracion = 3
thread number: 15, iteracion = 3
thread number: 4, iteracion = 1
thread number: 5, iteracion = 1
thread number: 6, iteracion = 1
thread number: 7, iteracion = 1
```

Figura 8: Hilos 4 - script 4

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./for_trace 16 8
thread number: 0, iteracion = 0
thread number: 4, iteracion = 2
thread number: 8, iteracion = 4
thread number: 9, iteracion = 4
thread number: 1, iteracion = 0
thread number: 14, iteracion = 7
thread number: 15, iteracion = 7
thread number: 2, iteracion = 1
thread number: 3, iteracion = 1
thread number: 6, iteracion = 3
thread number: 7, iteracion = 3
thread number: 10, iteracion = 5
thread number: 11, iteracion = 5
thread number: 12, iteracion = 6
thread number: 13, iteracion = 6
thread number: 5, iteracion = 2
```

Figura 9: Hilos 8 - script 4

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./for_trace 16 16
thread number: 3, iteracion = 3
thread number: 11, iteracion = 11
thread number: 2, iteracion = 2
thread number: 15, iteracion = 15
thread number: 12, iteracion = 12
thread number: 6, iteracion = 6
thread number: 5, iteracion = 5
thread number: 7, iteracion = 7
thread number: 8, iteracion = 8
thread number: 9, iteracion = 9
thread number: 4, iteracion = 4
thread number: 13, iteracion = 13
thread number: 10, iteracion = 10
thread number: 14, iteracion = 14
thread number: 0, iteracion = 0
thread number: 1, iteracion = 1
```

Figura 10: Hilos 16 - script 4

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./for_trace 16 5
thread number: 0, iteracion = 0
thread number: 1, iteracion = 0
thread number: 2, iteracion = 0
thread number: 3, iteracion = 0
thread number: 10, iteracion = 3
thread number: 11, iteracion = 3
thread number: 12, iteracion = 3
thread number: 13, iteracion = 4
thread number: 14, iteracion = 4
thread number: 15, iteracion = 4
thread number: 7, iteracion = 2
thread number: 8, iteracion = 2
thread number: 9, iteracion = 2
thread number: 4, iteracion = 1
thread number: 5, iteracion = 1
thread number: 6, iteracion = 1
```

Figura 11: Hilos 5 - script 4

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./for_trace 16 7
thread number: 10, iteracion = 4
thread number: 11, iteracion = 4
thread number: 0, iteracion = 0
thread number: 1, iteracion = 0
thread number: 2, iteracion = 0
thread number: 6, iteracion = 2
thread number: 7, iteracion = 2
thread number: 12, iteracion = 5
thread number: 13, iteracion = 5
thread number: 14, iteracion = 6
thread number: 15, iteracion = 6
thread number: 8, iteracion = 3
thread number: 9, iteracion = 3
thread number: 3, iteracion = 1
thread number: 4, iteracion = 1
thread number: 5, iteracion = 1
```

Figura 12: Hilos 5 - script 4

5. Grado de Paralelización

Para este experimento se corrió el código en modo paralelo, variando el número de hilos (NTHREADS) y el número de iteraciones (COUNT). Se realizaron 100 ejecuciones para cada configuración y se registró el número de resultados correctos.

COUNT	2 hilos	4 hilos	8 hilos	32 hilos	64 hilos
10	100	14	0	97	98
100	99	0	0	66	53
1000	0	0	4	29	15

Cuadro 1: Número de ejecuciones correctas en 100 corridas por configuración.

- a. **¿Por qué algunas respuestas son incorrectas?** Las respuestas incorrectas aparecen por una condición de carrera en la variable compartida `sum`. Cada hilo ejecuta la operación `sum = sum + i` en varios pasos (leer, sumar, escribir). Cuando dos o más hilos acceden a `sum` al mismo tiempo, se pierden actualizaciones. La tabla muestra que, incluso con configuraciones pequeñas como 4 hilos y 10 iteraciones, ya se producen errores frecuentes (solo 14/100 correctas).
- b. **¿Qué está causando los errores no deterministas?** La ejecución concurrente depende del planificador del sistema operativo. El orden en que los hilos leen y escriben cambia en cada corrida, lo que explica la variación: en algunos casos el resultado es correcto, en otros no. Esto se observa claramente en configuraciones como 32 y 64 hilos con `COUNT=100`, donde los resultados correctos fueron 66 y 53 de 100, respectivamente: no siempre falla, pero la probabilidad de error es alta.
- c. **¿Cuál es la probabilidad de obtener una respuesta incorrecta como función de NTHREADS y COUNT?** La probabilidad de error depende de dos factores:
 - Con más hilos (`NTHREADS`), hay más accesos simultáneos, lo que incrementa la chance de colisión.
 - Con más iteraciones (`COUNT`), hay más operaciones sobre la variable compartida, aumentando la probabilidad de que ocurra al menos una pérdida de actualización.

Por ejemplo:

- Con 2 hilos y `COUNT=10` no hubo errores (100 % correctas).
 - Con 4 hilos y `COUNT=10` la probabilidad de error fue 86 %.
 - Con 64 hilos y `COUNT=1000`, el error se presentó en el 85 % de las ejecuciones.
- d. **¿Existe una tendencia? ¿Cuál es la explicación intuitiva?** Sí, se observa una tendencia clara:

- Aumentar el número de hilos tiende a incrementar la probabilidad de error.
- Aumentar el número de iteraciones también eleva la probabilidad de error.

La explicación intuitiva es que más hilos implican más competencia por la variable global, y más iteraciones multiplican el número de oportunidades de conflicto. Aunque existen algunos casos atípicos (por ejemplo, 32 y 64 hilos con `COUNT=10` tuvieron un número sorprendentemente alto de ejecuciones correctas), en general el patrón confirma la teoría: sin sincronización, la concurrencia degrada la confiabilidad de los resultados.

6. Variable `sum` como privada

Cuando la variable `sum` se declara como `private`, cada hilo crea y usa una copia local independiente. Esto elimina la condición de carrera, ya que ningún hilo compite por actualizar una variable global.

Sin embargo, el resultado global deja de acumularse: cada hilo inicializa su propia `sum`, hace sus operaciones y luego descarta el valor al terminar. El valor final impreso en el programa corresponde únicamente al hilo maestro (u otro que sobrescriba al final), y por lo tanto el resultado ya no es aleatorio, pero sí incorrecto y constante para todas las ejecuciones.

En conclusión, `private(sum)` **no resuelve el problema de fondo**:

- Se elimina la no determinación debida a la condición de carrera.
- Pero se pierde la acumulación correcta entre hilos, lo que hace que el resultado global nunca represente la suma real.

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./point5_private 1000 64 1000
De 1000 ejecuciones, 0 fueron correctas y 1000 incorrectas.
```

Figura 13: Ejecución con 64 hilos, `COUNT=1000`, 1000 pruebas. El resultado se mantiene constante pero incorrecto.

7. Cláusula `reduction`

La cláusula `reduction(+:sum)` resuelve completamente el problema presentado en (5). Cuando se especifica esta cláusula, cada hilo obtiene una copia privada

de la variable `sum` y en ella acumula sus resultados parciales de manera independiente, evitando cualquier condición de carrera.

Al finalizar el bucle paralelo, OpenMP combina automáticamente todas las copias privadas aplicando la operación de reducción indicada (en este caso, la suma `+`) y coloca el resultado final en la variable global `sum`. Gracias a este mecanismo:

- El resultado es siempre correcto, coincidiendo exactamente con el valor que produciría el programa secuencial.
- El cálculo se mantiene determinista y estable en todas las ejecuciones.
- La solución es escalable, ya que funciona correctamente sin importar el número de hilos ni el valor de `COUNT`.

En las pruebas realizadas, incluso con configuraciones de alta concurrencia (por ejemplo, 64 hilos y `COUNT=1000`), el valor obtenido coincide con el esperado de la suma secuencial. De esta forma, la cláusula `reduction` garantiza tanto la corrección como la reproducibilidad del programa paralelo.

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./point7_n_runs 1000 64 1000  
De 1000 ejecuciones, 1000 fueron correctas y 0 incorrectas.
```

Figura 14: Ejecución con 64 hilos, `COUNT=1000`, 1000 pruebas. El resultado es correcto y se mantiene constante en todas las corridas.

8. Programa secuencial

La versión secuencial del programa calcula una aproximación precisa de π , obteniendo un valor de $\pi \approx 3,141592653590$, el cual coincide con la referencia matemática esperada.

En cuanto al rendimiento, el tiempo de ejecución medido para esta versión fue de aproximadamente 3,411 segundos en la máquina utilizada. Este resultado constituye la línea base de comparación para evaluar las mejoras que introduce la paralelización con OpenMP.

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_seq  
The value of PI is 3.141592653590  
Elapsed time (seconds): 3.411
```

Figura 15: Ejecución secuencial: aproximación de π y tiempo de ejecución.

9. Tabla *Time vs. NTHREADS*

NTHREADS	Tiempo (segundos)
2	1.839
4	1.003
8	0.706
16	0.705
32	0.683
64	0.738
128	0.698
256	0.689

Cuadro 2: Tiempo de ejecución en función del número de hilos.

El programa paralelo con OpenMP calcula correctamente el valor de π y reduce significativamente el tiempo de ejecución respecto a la versión secuencial, cuyo tiempo base fue de 3,411 segundos.

Se observa una clara tendencia de mejora hasta llegar a 8–16 hilos, donde el tiempo se estabiliza cerca de 0,7 segundos. Esto representa un **speedup de alrededor de 4.8 veces** con respecto a la ejecución secuencial.

Al aumentar el número de hilos por encima de los núcleos lógicos disponibles, el rendimiento ya no mejora. Incluso se aprecia una ligera degradación en 64 hilos debido a la sobrecarga de planificación y sincronización de OpenMP. No obstante, los tiempos se mantienen estables para 128 y 256 hilos, mostrando que el algoritmo es robusto pero limitado por el hardware.

En conclusión:

- La paralelización ofrece una mejora sustancial respecto al programa secuencial.
- El rendimiento máximo se alcanza alrededor de los 16–32 hilos, coincidiendo con la capacidad de paralelismo real de la máquina usada.
- Usar más hilos que núcleos disponibles no aporta beneficios y puede introducir sobrecostos de gestión.

Análisis de speedup y eficiencia

Tomando como referencia el tiempo secuencial medido $T_{seq} = 3,411$ s, calculamos:

$$\text{Speedup } S = \frac{T_{seq}}{T_p}, \quad \text{Eficiencia } E = \frac{S}{\text{NTHREADS}} \times 100 \%$$

NTHREADS	Tiempo (s)	Speedup (S)	Eficiencia (E)
2	1.839	1.855	92.74 %
4	1.003	3.401	85.02 %
8	0.706	4.831	60.39 %
16	0.705	4.838	30.24 %
32	0.683	4.994	15.61 %
64	0.738	4.622	7.22 %
128	0.698	4.887	3.82 %
256	0.689	4.951	1.93 %

Cuadro 3: Resultados de rendimiento: tiempo, speedup y eficiencia.

Interpretación y observaciones

- **Mejora inicial significativa:** Al pasar de 1 (secuencial) a 2–8 hilos se observa un speedup notable (máximo observado 4.83 con 8 hilos). Esto muestra que la parte paralelizable del algoritmo se beneficia claramente de varios hilos.
- **Eficiencia decreciente:** La eficiencia por hilo cae al incrementar el número de hilos. Con 2 hilos la eficiencia es alta (92.7%), con 4 hilos sigue siendo buena (85%), pero a partir de 8 hilos y sobre todo >16 hilos la eficiencia cae de forma pronunciada.
- **Techo de aceleración (speedup 4.6–5):** Observamos un límite práctico del speedup alrededor de 4.6–5 para esta máquina y este código: añadir más hilos ya no aumenta sustancialmente el speedup.
- **Conclusión práctica:** Para esta máquina y este problema, usar entre 8 y 32 hilos produce el mejor balance entre speedup y eficiencia; más allá de eso el coste de gestión de hilos y la contención hacen que la eficiencia por hilo sea muy baja.

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 2
The value of PI is 3.141592653590
Elapsed time (seconds) with 2 threads: 1.839
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 4
The value of PI is 3.141592653590
Elapsed time (seconds) with 4 threads: 1.003
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 8
The value of PI is 3.141592653590
Elapsed time (seconds) with 8 threads: 0.706
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 16
The value of PI is 3.141592653590
```

Figura 16: Ejecuciones con 2, 4, 8 y 16 hilos.

```
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 32
The value of PI is 3.141592653590
Elapsed time (seconds) with 32 threads: 0.683
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 64
The value of PI is 3.141592653590
Elapsed time (seconds) with 64 threads: 0.738
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 128
The value of PI is 3.141592653590
Elapsed time (seconds) with 128 threads: 0.698
japeto@aa642b9ad45a:/home/workspace/scripts$ ./pi_omp 256
The value of PI is 3.141592653590
Elapsed time (seconds) with 256 threads: 0.689
```

Figura 17: Ejecuciones con 32, 64, 128 y 256 hilos.