ARQUITECTURA DE COMPUTADORES: PRÁCTICA 4

ALEJANDRO PASCUAL[1] Y VÍCTOR YRAZUSTA[2]

ÍNDICE

[1] *alejandro.pascualp@estudiante.uam.es*
[2] *victor.yrazusta@estudiante.uam.es*

ÍNDICE DE FIGURAS

0 EJERCICIO 0: INFORMACIÓN SOBRE LA CACHÉ DEL SISTEMA

0.1 Arquitectura de la caché de los equipos del laboratorio

Tras la ejecución del primer comando obtenemos información acerca del procesador y de su cantidad de núcleos:

```
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 60
model name : Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
cpu MHz : 3200.009
stepping : 3
physical id : 0
siblings : 1
core id : 0
cpu cores : 1

processor : 1
vendor_id : GenuineIntel
cpu family : 6
model : 60
model name : Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
cpu MHz : 3200.009
physical id : 2
siblings : 1
core id : 0
cpu cores : 1
```

Recorte 1: cat /proc/cpuinfo (parte importante)

Como podemos observar contamos con dos núcleos dedicados, los cuales no poseen hiperthreading (lo podemos ver por la cantidad de siblings que tienen), que funcionan a 3.2 GHz de velocidad de reloj.

1 EJERCICIO 1: PROGRAMAS BÁSICOS DE OPENMP

1.1 ¿Se pueden lanzar más threads que cores tenga el sistema? ¿Tiene sentido hacerlo?

Sí que se pueden lanzar más threads que cores tenga el sistema. Puede tener sentido hacerlo para aprovechar los tiempos de bloqueo (E/S), pero no se va a poder ejecutar todos de manera paralela.

1.2 ¿Cuántos threads debería utilizar en los ordenadores del laboratorio? ¿Y en el clúster? ¿Y en su propio equipo?

Suponiendo que los hilos son "workers"que se reparten una misma tarea, el número de hilos debería ser proporcional al número de núcleos. Por lo tanto debería haber más hilos en un clúster que en un ordenador personal.

1.3 ¿Cómo se comporta OpenMP cuando declaramos una variable privada?

Para las variables privadas OpenMP asigna posiciones de memoria diferentes a cada hilo.

1.4 ¿Qué ocurre con el valor de una variable privada al comenzar a ejecutarse la región paralela?

El valor de la variable privada no está inicializado.

1.5 ¿Qué ocurre con el valor de una variable privada al finalizar la región paralela?

El valor de la variable se pierde.

1.6 ¿Ocurre lo mismo con las variables públicas?

No, las variables públicas se comparten entre hilos. Su valor al iniciar la región paralela es el valor que tenía asignado previamente. Al terminar la región paralela no se pierde su valor.

2 EJERCICIO 2: PARALELIZAR EL PRODUCTO ESCALAR

Tomo de datos

Es necesario repetir varias veces la ejecución de las pruebas para conseguir información fiable. Nosotros hemos ejecutado cada test 10 veces de manera intercalada, y hemos realizado la media de todas estas métricas para la realización de las gráficas de los diferentes ejecicios mediante el siguiente código Python:

```python
import sys

if len(sys.argv) != 2:
    print('Numero incorrecto de argumentos.')
    exit()

file = open(sys.argv[1], 'r')

averages = {}

for line in file.readlines():
    words = line.split()
    n = int(words[0].replace(",", ""))
    values = []
    for word in words[1:]:
        values.append(float(word.replace(",", "")))
    if averages.get(n) is None:
        averages[n] = {
            'values': values,
            'samples': 1
        }
    else:
        values_averages = averages[n]['values']
        samples = averages[n]['samples']
        for i in range(0, len(values_averages)):
            values_averages[i] *= samples
            values_averages[i] += values[i]
            values_averages[i] /= samples+1
        averages[n]['samples'] += 1

file.close()
file = open(sys.argv[1], 'w')

for n, averages in averages.items():
    file.write(str(n))
    for average in averages['values']:
        file.write('\t' + str(average))
    file.write('\n')

file.close()
```

Recorte 2: calculo_media.py

2.1 ¿En qué caso es correcto el resultado? ¿A qué se debe la diferencia?

El resultado correcto es el del archivo "pescalar_par2". En el primero hay una condición de carrera, ya que los diferentes hilos van pisándose los valores de las sumas parciales. Por esto el valor obtenido en el primer ejecutable es sistemáticamente menor al obtenido en el segundo, además de ser fluctuante según la ejecución (el segundo ejecutable no lo es porque los valores generados son pseudoaleatorios y la semilla es igual en las diferentes ejecuciones).

2.2 Gráficas

Las gráficas que obtenemos después de ejecutar nuestras pruebas son estas:

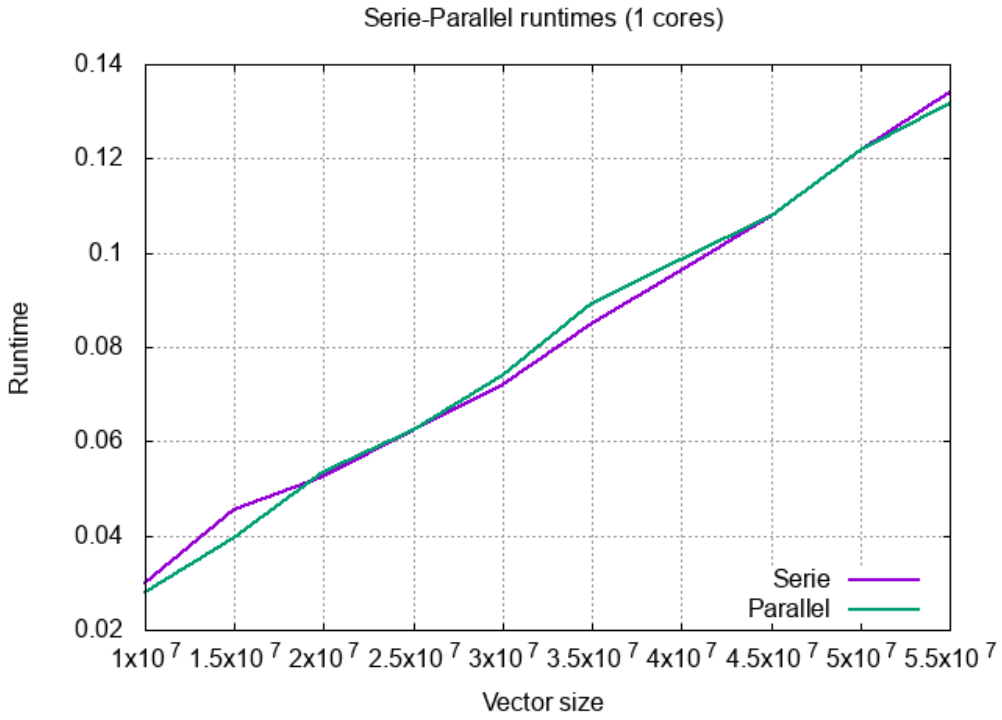

Figura 1: Gráfica tiempo_nucleos_1.png.

Como podemos observar, el tiempo de ejecución al ejecutar las dos versiones con un solo núcleo es muy similar, al no contar con hyperthreading, el procesador no es capaz de paralelizar el trabajo, y las dos versiones resultan prácticamente equivalentes. Los resultados cambian al aumentar el número de núcleos:

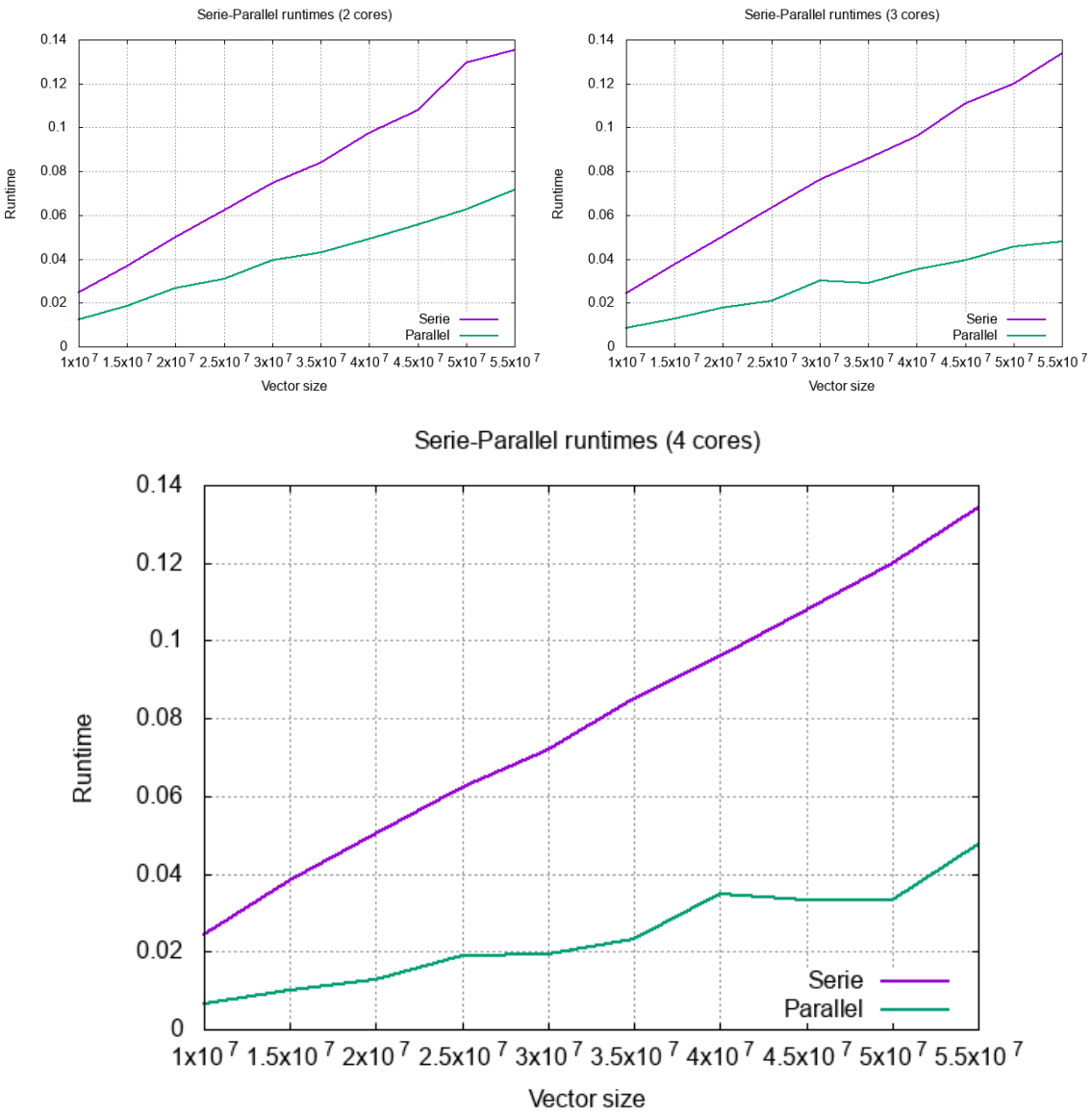

Figura 2: Gráfica tiempo_nucleos_2, 3 y 4.

Cuando el procesador es capaz de disponer de varios núcleos entre los que repartir la carga de trabajo, puede paralelizar el cómputo, y por lo tanto vemos que la versión paralela resulta significativamente más rápida que la versión en serie. Esta diferencia es más acusada cuanto más grande es el vector y mayor número de núcleos están disponibles.

2.3 En términos del tamaño de los vectores, ¿compensa siempre lanzar hilos para realizar el trabajo en paralelo, o hay casos en los que no? Si compensara siempre, ¿en qué casos no compensa y por qué? ¿Se mejora siempre el rendimiento al aumentar el número de hilos a trabajar?

En el tests que hemos realizado todos los tamaños son lo suficientemente grandes como para que el trabajo adicional de crear los nuevos hilos sea insignificante en comparación al beneficio obtenido. Podría no compensar si los tamaños fuesen muy pequeños y no se rentabilizase la creación de hilos, si el procesador estuviese muy saturado y no dispusiese de núcleos ociosos o si los problemas generados por la paralelización del trabajo (procesos muy dependientes entre sí) superasen los beneficios obtenidos.

2.4 Si no fuera así, ¿a qué debe este efecto?

Siempre es así.

2.5 Valore si existe algún tamaño del vector a partir del cual el comportamiento de la aceleración va a ser muy diferente del obtenido en la gráfica.

Como ya se ha mencionado en el apartado anterior[2,3], con vectores muy pequeños el trabajo que supone crear los hilos tendría un impacto mucho mayor. Con vectores muy grandes seguirá funcionando en torno a n veces más rápido una paralelización en n núcleos.

3 EJERCICIO 3: PARALELIZAR LA MULTIPLICACIÓN DE MATRICES

Todas las tablas han sido rellenadas multiplicando matrices de 1000x1000.

Tiempos de ejecución (s)				
Versión\#hilos	1	2	3	4
Serie	6.6	6.6	6.6	6.6
Paralela (1)	8.4	3.9	44	46
Paralela (2)	6.4	3.2	4.6	3.5
Paralela (3)	6.7	3.2	3.1	3.2

Speedup				
Versión\#hilos	1	2	3	4
Serie	1	1	1	1
Paralela (1)	0.8	1.7	0.15	0.14
Paralela (2)	1	2.06	1.43	1.88
Paralela (3)	1	2.06	2.1	2.06

3.1 ¿Cuál de las tres versiones obtiene peor rendimiento? ¿A qué se debe?

El que peor rendimiento tiene es el que paraleliza el bucle más interno (la primera versión). El motivo principal es que es necesaria una reducción posterior, ya que los diferentes hilos modifican el mismo elemento de la matriz.

3.2 ¿Cuál de las tres versiones obtiene mejor rendimiento? ¿A qué se debe?

El que mejor rendimiento tiene (con poca diferencia) es el más externo (la tercera versión). Su ventaja respecto al intermedio es que los hilos solo deben ”sincronizarse“ al finalizar todo el proceso y no en cada iteración del bucle externo. Por otro lado tiene la desventaja de que, al repartir bloques de trabajo de mayor tamaño, es más probable que el resultado tenga que esperar a que unos pocos hilos acaben su parte, mientras que el resto podrían haber terminado.

3.3 Gráficas

Después de la realización de los tests automáticos obtenemos la siguiente gráfica:

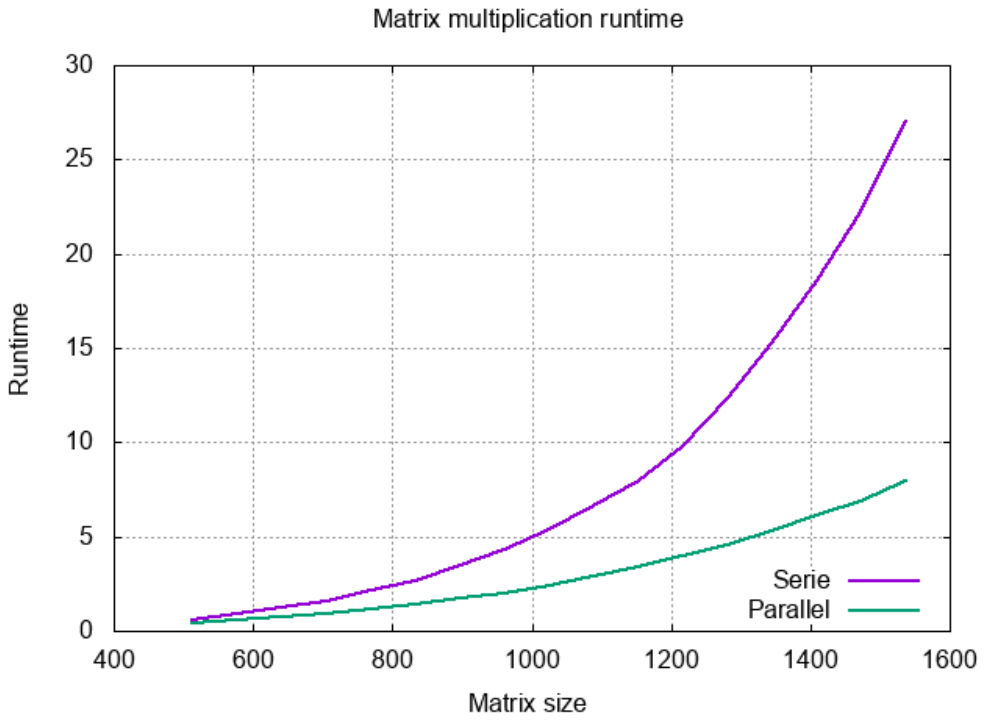

Figura 3: Gráfica tiempos.png.

En ella podemos observar el coste de $O(N^3)$ que conlleva la multiplicación de matrices. Observamos que la versión en paralelo es significativamente más rápida, diferencia que se hace más grande conforme crece el tamaño de las matrices a multiplicar.

4 EJERCICIO 4: EJEMPLO DE INTEGRACIÓN NUMÉRICA

4.1 ¿Cuántos rectángulos se utilizan en la versión del programa que se da para realizar la integración numérica?

El programa utiliza cien mil rectángulos para realizar el cómputo.

4.2 ¿Qué diferencias observa entre estas dos versiones?

La principal diferencia entre los dos programas consiste en que la versión 4 utiliza una variable auxiliar para almacenar los resultados parciales de cada hilo, mientras que en la primera, al estar modificando todos los hilos regiones cercanas de memoria, el funcionamiento de la caché ralentiza la ejecución, produciendose false sharing.

4.3 Ejecute las dos versiones recién mencionadas. ¿Se observan diferencias en el resultado obtenido? ¿Y en el rendimiento? Si la respuesta fuera afirmativa, ¿sabría justificar a qué se debe este efecto?

El resultado calculado es el mismo, pero la versión en la que se produce false sharing resulta significativamente más lenta que en la que no se produce (gracias a que se guarda cada hilo sus resultados parciales).

4.4 Ejecute las versiones paralelas 2 y 3 del programa. ¿Qué ocurre con el resultado y el rendimiento obtenido? ¿Ha ocurrido lo que se esperaba?

En las versiones 3 y 4 se observa una mejoría en el rendimiento en comparación con la versión 1, ya que la versión 2 hace que las sumas parciales sean privadas para cada proceso (evitando el false sharing) y la versión 3 se vale del padding para generar espacio entre los valores de las sumas parciales, de manera que las regiones de memoria son lo suficientemente distantes como para evitar el false sharing.

4.5 Abra el fichero pi_par3.c y modifique la línea 32 del fichero para que tome los valores fijos 1, 2, 4, 6, 7, 8, 9, 10 y 12. Ejecute este programa para cada uno de estos valores. ¿Qué ocurre con el rendimiento que se observa?

El tiempo de ejecución es menor cuanto menor es el tamaño de datasz, ya que de esta manera el padding es mayor y por lo tanto el efecto del false sharing es menos acusado.

4.6 Gráficas

Tras ejecutar las pruebas obtenemos las siguientes gráficas:

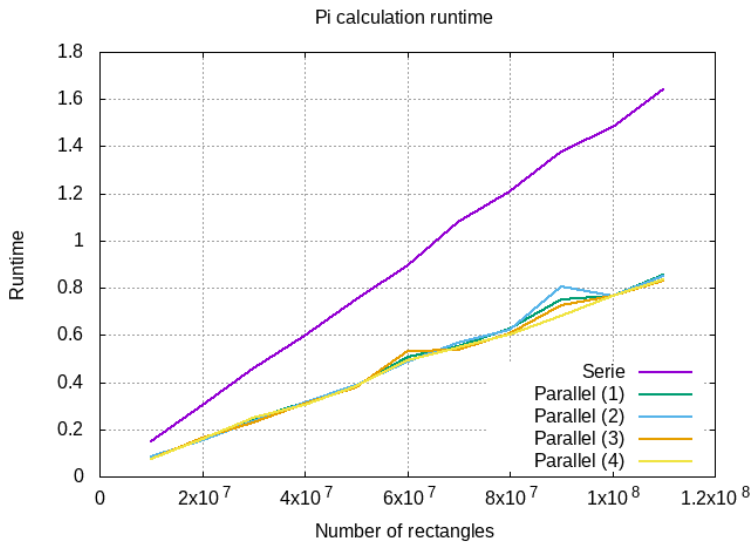

Figura 4: Gráfica tiempos_serie_par.png.

En esta primera gráfica podemos observar que los tiempos de ejecución de la versión en serie son más altos, mientras que los de las versiones en paralelo crecen de manera más lenta.

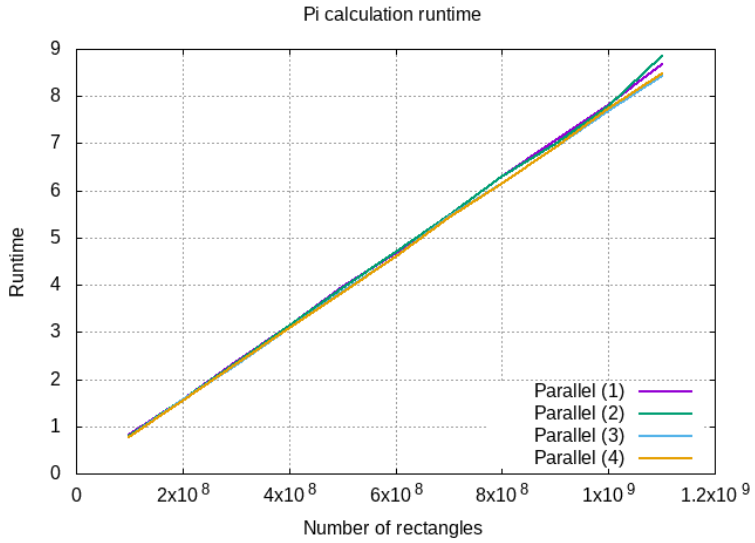

Figura 5: Gráfica tiempos_par.png.

En esta gráfica únicamente se comparan los tiempos de las versiones en paralelo. Se observa que la diferencia en el tiempo de ejecución para tamaños medianos es prácticamente insignificante, acusandose un poco más para tamaños más grandes.

5 EJERCICIO 5: USO DE LA DIRECTIVA CRITICAL Y REDUCTION

5.1 Ejecute las versiones 4 y 5 del programa. Explique el efecto de utilizar la directiva critical.¿Qué diferencias de rendimiento se aprecian? ¿A qué se debe este efecto?

La directiva critical restringe la ejecución de esa sección de código a un hilo de manera simultanea. Esto impacta de manera negativa en el rendimiento, ya que los diferentes hilos han de esperarse para ejecutar la sección restringida.

5.2 Ejecute las versiones 6 y 7 del programa. Explique el efecto de utilizar las directivas utilizadas.¿Qué diferencias de rendimiento se aprecian? ¿A qué se debe este efecto?

En la versión 6 cada hilo paraleliza a su vez el bucle for, lo que genera una gran cantidad de hilos que realizan el cálculo. La versión 7 paraleliza directamente el bucle for. Ambas versiones logran un buen rendimiento, al evitar el false sharing permitiendo a OpenMP que paralelice él los bucles for.

5.3 Gráficas

Después de la realización de algunas pruebas hemos conseguido la siguiente gráfica:

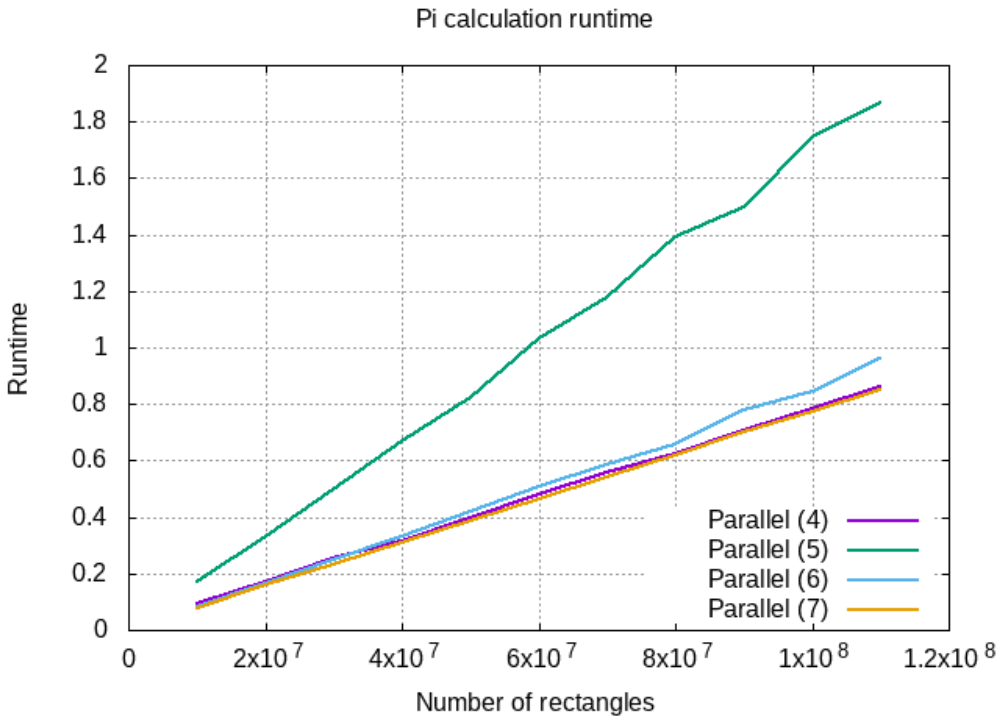

Figura 6: Gráfica tiempos_par_ej_5.png.

En ella se puede observar que, mientras que los tiempos de ejecución de las versiones 4 y 7 son casi idénticos, los de la versión 6 son algo superiores, y los de la versión 5 son significativamente superiores. Esto es debido al uso que se le da en esta versión a la directiva "critical", tal como explicamos en el punto anterior [5,1].