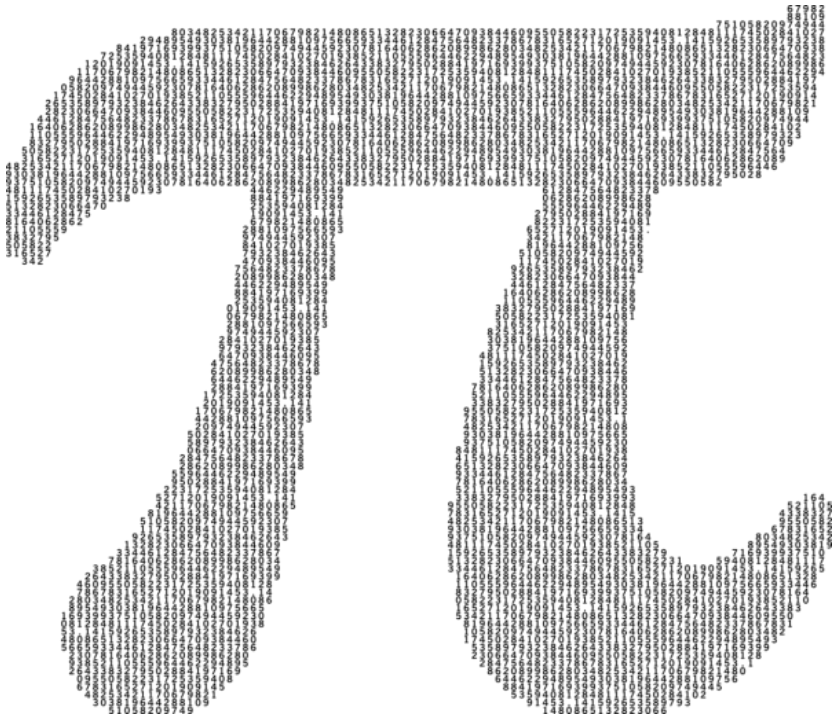
INGENIERÍA DE LOS COMPUTADORES

PRÁCTICA 2: ESTUDIO Y PROPUESTA DE LA PARALELIZACIÓN DE UNA APLICACIÓN

Cálculo de aproximación del número PI mediante el algoritmo de Spigot



AUTORES:
- PABLO SÁNCHEZ PÉREZ
- DEMIAN CALVO RODRÍGUEZ
- JULIO CESAR BASTIDAS SANCHEZ
- JAVIER ORTEGA GARCÍA

ÍNDICE

1. OBJETIVOS

El objetivo de esta práctica es encontrar un problema computacionalmente costoso para practicar su paralelización. Es esencial comprender la estructura y operación del programa, que inicialmente debería ser secuencial. Después, representaremos el programa mediante un gráfico de flujo de control, examinaremos las variables utilizadas y llevaremos a cabo un análisis de rendimiento.

2. INTRODUCCIÓN AL PROBLEMA

¿Qué es el número PI?
Desde una perspectiva matemática, PI representa la relación entre la longitud de una circunferencia y su diámetro en la geometría euclidiana. Es un número infinito e irracional, y figura como una de las constantes matemáticas más fundamentales.

¿Por qué es un buen candidato a paralelizar?
Dado que PI es infinito, en programación podemos establecer un nivel de precisión para visualizar una porción del número. Esto implica calcular una aproximación de PI, lo que lo convierte en un buen candidato para la paralelización.

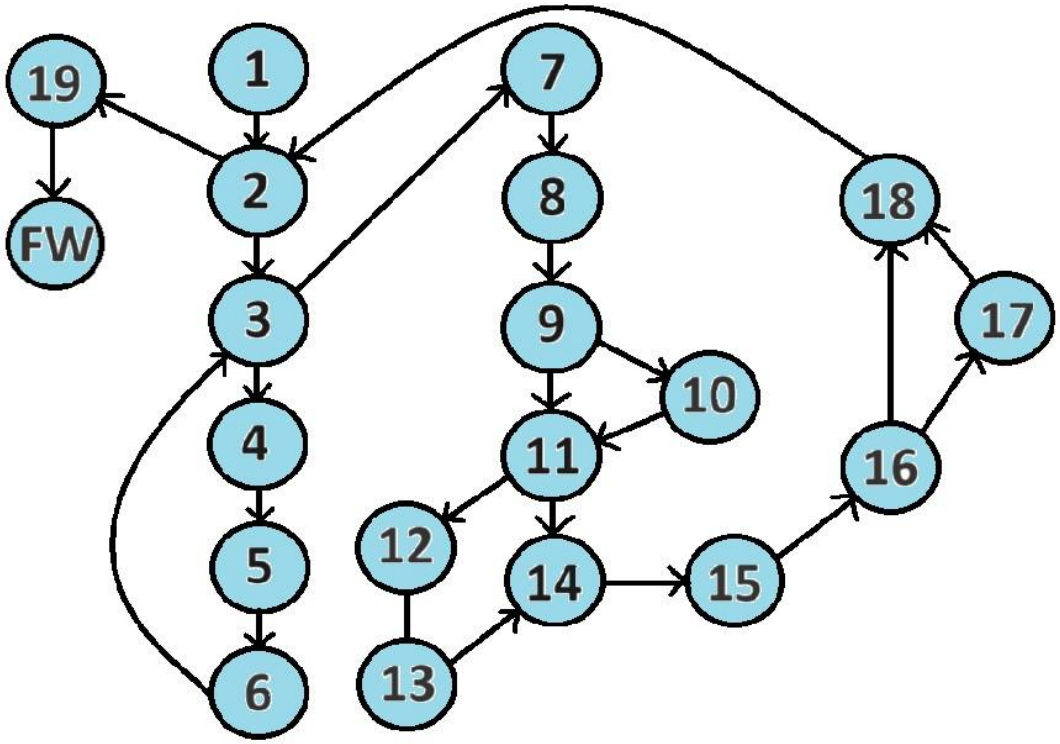
Formas de obtener la aproximación del número pi
Hay diversas maneras de calcular los dígitos de PI, como el uso de **polígonos de lado 2*n**, el **Método Monte Carlo** (que implica dibujar un cuadrado y un círculo con diámetro igual a un lado del cuadrado, y distribuir puntos para estimar áreas) o el método de **Buffon** (lanzar agujas en una superficie plana con líneas trazadas y calcular la probabilidad de que atraviesen una línea). En este caso, hemos empleado el **Algoritmo de Spigot** para alcanzar nuestro resultado.

Algoritmo de Spigot
En 1995, Stanley Rabinowitz y Stan Wagon introdujeron el algoritmo conocido como "Espiga". La peculiaridad de este método radica en que extrae los dígitos de manera progresiva, sin volver a utilizarlos una vez computados. Esta característica lo distingue de otros algoritmos que no revelan un resultado hasta su conclusión. Además, el algoritmo "Espiga" prescinde de operaciones de alta o baja precisión, así como de números en coma flotante, limitándose exclusivamente al uso de enteros de tamaño relativamente reducido.

3. ESTUDIO DEL PROBLEMA

3.1. GRAFO DE CONTROL DE FLUJO

El siguiente grafo refleja la estructura que siguen las operaciones del programa. Se han reflejado tanto los condicionales como los bucles, sin reflejar asignaciones de variables (al menos las que no contienen una operación al inicializarlas). Podemos observar dos bucles en los cuales hay varios condicionales. Esto nos puede indicar que sería provechoso paralelizar el programa, pues encontramos toma de decisiones que se pueden realizar a la vez y bucles que, como veremos más adelante, algunos parámetros de compilación los tratan de diferentes maneras.



3.2. VARIABLES DEL PROGRAMA

La cantidad de variables del programa no es muy elevada, no superando las 10. Estas son enteros que guardan el índice de un bucle, almacena el resto o el dígito que se va a mostrar. También se cuenta con un vector de enteros.

3.3. CARGA DEL PROGRAMA

El programa calcula el número de dígitos de PI. Por lo tanto la forma de aumentar la carga es incrementando el número de dígitos. Vamos a realizar pruebas para la cantidad de 20000 y 50000 dígitos, incrementando el tiempo considerablemente de una para otra.

4. PARÁMETROS DE COMPILACIÓN

Parámetros de tipo -OX

El nivel más bajo de optimización es el -O0 que usa por defecto g++/gcc sin optimización, se usa para la depuración del código y reducir el coste del compilador. El siguiente nivel -O1 optimiza el tamaño omitiendo optimizaciones que tienden a incrementar el tamaño del objeto. El nivel -O2 es el grado de optimización recomendado, este nivel activará algunas opciones del nivel -O1 como la reducción del tamaño y agiliza el tiempo de compilación.

Por último tenemos el nivel más alto de optimización, el nivel -O3 que en términos de coste de compilación y de memoria es bastante caro pero a cambio nos da una mayor optimización aunque también puede suceder lo contrario y ralentizar el sistema.

Parámetro -floop-parallelize-all

El parámetro -floop-parallelize-all paraleliza todos los bucles que pueden ser analizados para que no contengan dependencias transportadas por bucles sin comprobar que sea rentable para paralelizar esos bucles.

Parámetro -march

El parámetro -march indica qué código debería producir para su arquitectura de procesador. Diferentes CPUs tienen diferentes características, soportan diferentes conjuntos de instrucciones y tienen diferentes formas de ejecutar código.

Parámetro -Ofast

Ignora el estricto cumplimiento de las normas. Activa todas las -O3 optimizaciones.También permite optimizaciones que no son válidas para todos los programas que cumplen con los estándares.

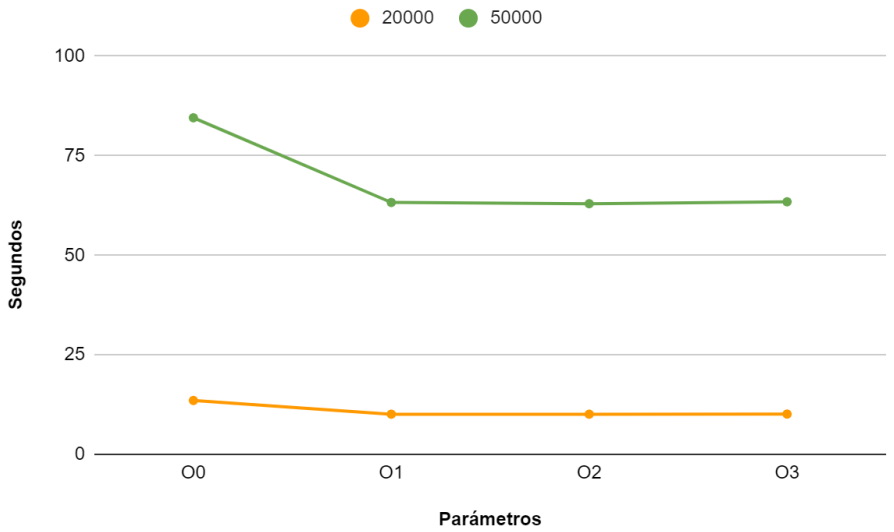
Comparación de resultados segundos:

n(dígitos de pi)	-O0	-O1	-O2	-O3
20000	13.52	10.0806	10.0807	10.11
50000	84.53	63.26	62.95	63.43

Como podemos observar existe un mayor rendimiento del nivel O0 a O1 pero no existe una gran diferencia entre los demás niveles.

Como podemos ver en la siguiente tabla el parámetro **-**floop-parallelize-all en comparación con -O3 no nos resulta muy útil ya que no optimiza nada respecto a los otros niveles de optimización.

Tamaño del problema	-O3	-floop-parallelize-all
50000	63.43	86.70



5. ANÁLISIS DE RENDIMIENTO

Para comparar rendimientos hemos utilizado tres ordenadores diferentes, un portátil con las siguientes especificaciones.

```
demiancr12@LAPTOP-JB9NUVJ1:/mnt/c/Users/demia/Desktop/P2/P2$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
    CPU family:          6
    Model:               165
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            2
    BogoMIPS:            5184.00
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep
                         e2 ss ht syscall nx pdpe1gb rdtscp lm const
                         ni pclmulqdq vmx ssse3 fma cx16 pdcm pcid s
                         rdrand hypervisor lahf_lm abm 3dnowprefetch
                         pr_shadow vnmi ept vpid ept_ad fsgsbase bmi
                         hopt xsaveopt xsavec xgetbv1 xsaves flush_l
Virtualization features:
  Virtualization:        VT-x
  Hypervisor vendor:     Microsoft
  Virtualization type:   full
Caches (sum of all):
  L1d:                   192 KiB (6 instances)
  L1i:                   192 KiB (6 instances)
  L2:                    1.5 MiB (6 instances)
```

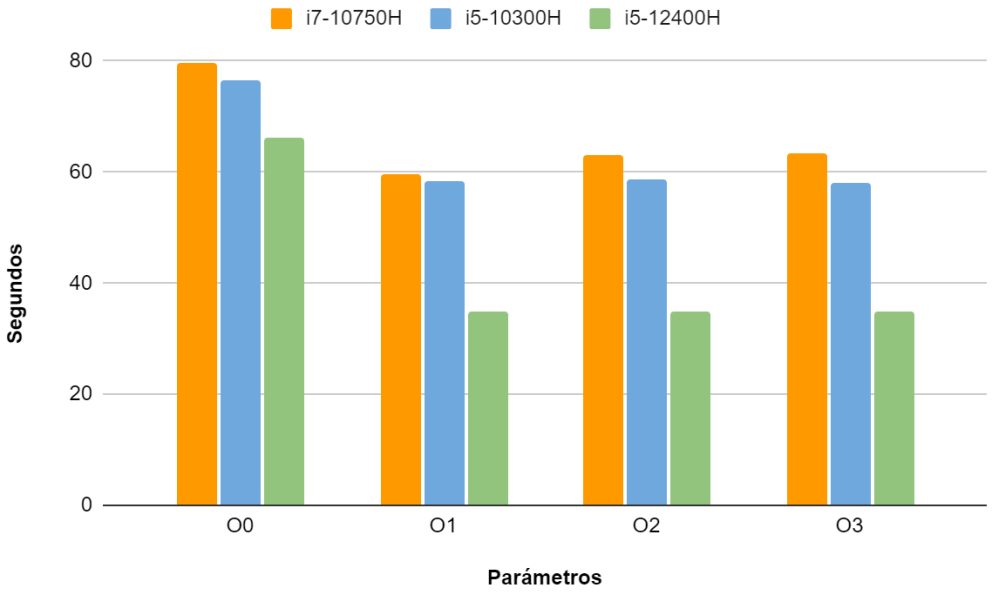
Otro portátil de uno de nuestros compañeros del grupo con otras especificaciones distintas:

```
psp59@psp59-ASUS-TUF:~/Escritorio/IC/P2$ lscpu
Arquitectura:                         x86_64
  modo(s) de operación de las CPUs:   32-bit, 64-bit
  Address sizes:                      39 bits physical, 48 bits virtual
  Orden de los bytes:                 Little Endian
CPU(s):                               8
  Lista de la(s) CPU(s) en línea:     0-7
ID de fabricante:                     GenuineIntel
  Nombre del modelo:                  Intel(R) Core(TM) i5-10300H CPU @ 2.50G
                                      Hz
    Familia de CPU:                   6
    Modelo:                           165
    Hilo(s) de procesamiento por núcleo: 2
    Núcleo(s) por «socket»:           4
    «Socket(s)»                       1
    Revisión:                         2
    CPU MHz máx.:                     4500,0000
    CPU MHz mín.:                     800,0000
    BogoMIPS:                         4999.90
    Indicadores:                      fpu vme de pse tsc msr pae mce cx8 apic
                                       sep mtrr pge mca cmov pat pse36 clflus
                                      h dts acpi mmx fxsr sse sse2 ss ht tm p
                                      be syscall nx pdpe1gb rdtscp lm constan
                                      t_tsc art arch_perfmon pebs bts rep_goo
                                      d nopl xtopology nonstop_tsc cpuid aper
                                      fmperf pni pclmulqdq dtes64 monitor ds_
                                      cpl vmx est tm2 ssse3 sdbg fma cx16 xtp
                                      r pdcm pcid sse4_1 sse4_2 x2apic movbe
                                      popcnt tsc_deadline_timer aes xsave avx
                                       f16c rdrand lahf_lm abm 3dnowprefetch
                                      cpuid_fault epb invpcid_single ssbd ibr
                                      s ibpb stibp ibrs_enhanced tpr_shadow v
                                      nmi flexpriority ept vpid ept_ad fsgsba
                                      se tsc_adjust bmi1 avx2 smep bmi2 erms
                                      invpcid mpx rdseed adx smap clflushopt
                                      intel_pt xsaveopt xsavec xgetbv1 xsaves
                                       dtherm ida arat pln pts hwp hwp_notify
                                       hwp_act_window hwp_epp pku ospke md_cl
                                      ear flush_l1d arch_capabilities
Virtualization features:
  Virtualización:                     VT-x
Caches (sum of all):
  L1d:                                128 KiB (4 instances)
  L1i:                                128 KiB (4 instances)
  L2:                                 1 MiB (4 instances)
  L3:                                 8 MiB (1 instance)
```

Y el ultimo pc elegido ha sido uno de los laboratorios de la EPS con estas otras especificaciones:

Para la comparación hemos usado una n=50000 y estos son los resultados con los diferentes optimizadores:

PCs	O0	O1	O2	O3
i7-10750H	79.52	59.05	62.95	63.43
i5-10300H	76.51	58.40	58.71	58.17
i5-12400H	66.23	35.04	34.96	35.00



6. Problema de los grifos

Para resolver estos problemas, vamos a usar la idea de tasas de trabajo. La tasa de trabajo es la cantidad de trabajo que se hace por unidad de tiempo. En este caso, el "trabajo" es llenar el depósito de agua, y la tasa de trabajo se mide en depósitos por hora.

1. Un grifo tarda 4 horas en llenar el depósito, por lo que su tasa de trabajo es 1/4 depósito por hora.
2. Otro grifo tarda 20 horas en llenar el depósito, por lo que su tasa de trabajo es 1/20 depósito por hora.

Ahora, veamos cada uno de los escenarios:

Escenario 1: Un grifo que tarda 4 horas y otro que tarda 20 horas en llenar el depósito.

Cuando usamos ambos grifos juntos, sus tasas de trabajo se suman:

Tasa de trabajo total = 1/4 + 1/20 = 5/20 + 1/20 = 6/20 = 3/10 depósito por hora.

Ahora, para saber cuánto tiempo se tardará en llenar el depósito, simplemente tomamos la inversa de la tasa de trabajo total:

Tiempo = 1 / Tasa de trabajo total Tiempo = 1 / (3/10) Tiempo = 10/3 horas = 3 horas y 20 minutos.

La ganancia en velocidad es la relación entre el tiempo que se tarda utilizando ambos grifos y el tiempo que se tarda utilizando el grifo más lento:

Ganancia en velocidad = Tiempo con un solo grifo / Tiempo con ambos grifos Ganancia en velocidad = 20 horas / (10/3) horas = 6.

La eficiencia se refiere a la fracción del trabajo que se realiza utilizando ambos grifos en comparación con el trabajo que haría el grifo más rápido solo:

Eficiencia = Tasa de trabajo total / Tasa de trabajo del grifo más rápido Eficiencia = (3/10) / (1/4) = (3/10) / (2.5/10) = 3/2.5 = 6/5.

Escenario 2: Dos grifos, ambos tardan 4 horas.

La tasa de trabajo total en este caso es simplemente la suma de las tasas de trabajo de los dos grifos, ya que son iguales:

Tasa de trabajo total = 1/4 + 1/4 = 2/4 = 1/2 depósito por hora.

Tiempo = 1 / Tasa de trabajo total Tiempo = 1 / (1/2) = 2 horas.

Ganancia en velocidad = 4 horas / 2 horas = 2. Eficiencia = 1/2 / 1/4 = 2.

Escenario 3: Dos grifos, ambos tardan 20 horas.

Tasa de trabajo total = 1/20 + 1/20 = 2/20 = 1/10 depósito por hora.

Tiempo = 1 / (1/10) = 10 horas.

Ganancia en velocidad = 20 horas / 10 horas = 2. Eficiencia = 1/10 / 1/20 = 2.

Escenario 4: Tres grifos, dos de 20 horas y uno de 4 horas.

Tasa de trabajo total = 1/20 + 1/20 + 1/4 = 2/20 + 2/20 + 5/20 = 9/20 depósito por hora.

Tiempo = 1 / (9/20) = 20/9 horas ≈ 2 horas y 13 minutos.

En resumen, la ganancia en velocidad y la eficiencia son constantes en los escenarios 2 y 3, pero en el escenario 4, donde se tienen tres grifos, el tiempo se reduce aún más. En todos los casos, utilizar múltiples grifos hace que el trabajo se realice más rápido y de manera más eficiente en comparación con un solo grifo.