Tarea 2: Calcular Pi usando mensajes

Esteban Rodríguez Betancourt

2024-04

Introducción

El objetivo de esta tarea es calcular el valor de π utilizando el método de Monte Carlo. Para ello, se generan puntos aleatorios en un cuadrado de lado 1 y se cuentan cuántos de estos puntos caen dentro de un círculo de radio 1. El valor de π se puede aproximar mediante la siguiente fórmula:

$$\pi \approx 4 \times \frac{\text{puntos dentro del círculo}}{\text{puntos totales}}$$

A diferencia de la tarea anterior, en esta ocasión deberán implementar el cálculo de π utilizando mensajes. Cada hilo generará una cantidad igual de puntos y enviará un mensaje al hilo principal con la cantidad de puntos dentro del círculo. El hilo principal se encargará de sumar estos valores y calcular el valor de π.

Implementación

Al igual que en la tarea anterior, pueden usar el siguiente código como base para su implementación:

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <math.h>
#include <stdint.h>

#define MAX_THREADS 50000

int total_threads;
int points_inside_circle = 0;
int total_points;

void* throw_darts(void* arg) {
    int childID = (uintptr_t)arg;
    int points_per_thread = total_points / total_threads;
    unsigned int seed = time(NULL) * (childID + 1);

    for (int i = 0; i < points_per_thread; i++) {
        double x = (double)rand_r(&seed) / RAND_MAX;
        double y = (double)rand_r(&seed) / RAND_MAX;

        if (sqrt(x * x + y * y) <= 1) {
            points_inside_circle++;
        }
    }

    return NULL;
```

```
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        printf("Uso: %s <total_points> <total_threads>\n", argv[0]);
        return 1;
    }

    total_points = atoi(argv[1]);
    total_threads = atoi(argv[2]);

    if (total_threads > MAX_THREADS) {
        printf("El número de hilos no puede ser mayor a %d\n", MAX_THREADS);
        return 1;
    }

    if (total_points % total_threads != 0) {
        printf("El número de puntos debe ser divisible entre el número de hilos\n");
        return 1;
    }

    // Acá deberán introducir el paralelismo
    throw_darts((void*)1);

    double pi = 4.0 * points_inside_circle / total_points;
    printf("Valor de pi: %f\n", pi);

    return 0;
}
```

Modifiquen el código anterior para que el cálculo de π se realice en paralelo utilizando pthreads y paso de mensajes. Recuerden que cada hilo debe generar una cantidad igual de puntos.

Reporte de resultados

Deberán entregar un reporte con lo siguiente:

1. Implementación del programa en C, siguiendo las siguientes estrategias (un archivo por estrategia):
    1. El programa original, sin modificaciones.
    2. El programa modificado para que el cálculo de π se realice en paralelo utilizando pthreads.
        1. Implementen una cola que soporte los métodos `push` y `pop` para enviar mensajes al hilo principal. Asegúrense de que la cola sea segura para concurrencia. Usen esta cola para enviar los mensajes al hilo principal.
        2. Creen un message queue para enviar los mensajes al hilo principal desde los hilos trabajadores.
        3. Creen un pipe para enviar los mensajes al hilo principal desde los hilos trabajadores.
        4. Investiguen cómo usar Boost.Fiber y realicen el cálculo de π utilizando un `buffered_channel` (es posible que esta parte no la puedan realizar en el laboratorio).
2. Instrucciones detalladas de cómo compilar el programa. Si usaron un Makefile añádanlo.
3. Prueben su programa usando Helgrind o ThreadSanitizer y describan los problemas de concurrencia detectados, porqué sucedieron y cómo los corrigieron. Corríjanlos y vuelvan a probar.
    - Por supuesto, NO corrijan los errores en 1.2.1.
4. Realicen pruebas con diferentes cantidades de puntos y de hilos. Muestren los resultados obtenidos y el tiempo de ejecución en cada caso. Pueden usar el comando time para medir el tiempo de ejecución (por ejemplo, time ./pi 100000000 4).

Sugerencias y observaciones

Observen que deben pasar datos adicionales al thread. Creen una estructura que contenga los datos necesarios y pasen un puntero a esta estructura al thread.

Asegúrense que el mensaje que ustedes transmiten pueda ser escrito de forma atómica en el mecanismo de paso de mensajes que elijan. Para un pipe este debe ser menor que PIPE_BUF, para un message queue debe ser menor que el tamaño del mensaje usado en la creación del message queue, y para un canal de Boost.Fiber debe ser menor que el tamaño del buffer.

Calculo de Pi en otros lenguajes

Go

A continuación muestro un ejemplo de cómo se puede calcular el valor de π en Go utilizando paso de mensajes. En este caso, se utilizan goroutines y canales para enviar mensajes al hilo principal. Si no disponen de un compilador de Go pueden usar https://www.onlinegdb.com/online_go_compiler para probar el código.

```go
package main

import (
    "fmt"
    "math"
    "math/rand"
    "os"
    "strconv"
    "sync"
)

const MAX_THREADS = 50000

func throwDarts(pointsPerThread int, ch chan<- int, wg *sync.WaitGroup) {
    defer wg.Done() // Signal completion upon return
    randSrc := rand.New(rand.NewSource(rand.Int63())) // Create a local random source
    pointsInsideCircle := 0
    for i := 0; i < pointsPerThread; i++ {
        x := randSrc.Float64()
        y := randSrc.Float64()
        if math.Sqrt(x*x+y*y) <= 1 {
            pointsInsideCircle++
        }
    }
    ch <- pointsInsideCircle
}

func collectResults(ch <-chan int, totalThreads int) int {
    totalInside := 0
    for i := 0; i < totalThreads; i++ {
        totalInside += <-ch
    }
    return totalInside
}

func main() {
    if len(os.Args) != 3 {
        fmt.Printf("Uso: %s <total_points> <total_threads>\n", os.Args[0])
        return
```

```
    }

    totalPoints, err := strconv.Atoi(os.Args[1])
    if err != nil {
        fmt.Println("Número de puntos inválido")
        return
    }

    totalThreads, err := strconv.Atoi(os.Args[2])
    if err != nil {
        fmt.Println("Número de hilos inválido")
        return
    }

    if totalThreads > MAX_THREADS {
        fmt.Printf("El número de hilos no puede ser mayor a %d\n", MAX_THREADS)
        return
    }

    if totalPoints%totalThreads != 0 {
        fmt.Println("El número de puntos debe ser divisible entre el número de hilos")
        return
    }

    pointsPerThread := totalPoints / totalThreads
    ch := make(chan int) // Unbuffered channel
    var wg sync.WaitGroup

    wg.Add(totalThreads)
    for i := 0; i < totalThreads; i++ {
        go throwDarts(pointsPerThread, ch, &wg)
    }

    go func() {
        wg.Wait()
        close(ch)
    }()

    totalInside := collectResults(ch, totalThreads)

    pi := 4.0 * float64(totalInside) / float64(totalPoints)
    fmt.Printf("Valor estimado de pi: %f\n", pi)
}
```