INSTITUTO FEDERAL Catarinense Campus Videira	CURSO: **Ciência da Computação**	
	COMPONENTE CURRICULAR: **Programação Paralela e Multicore**	SEMESTRE: **2021-1**
	ALUNO: **Alexandre Raisel**	DATA: 12/03/2021
	PROFESSOR: **Manassés Ribeiro**	

Cálculo do valor de PI (Π) através do método de Monte Carlo utilizando Streams Paralelas Java.

O presente relatório apresenta uma breve introdução sobre o problema abordado, definido como o cálculo de PI através do método de Monte Carlo. Bem como, a forma proposta para resolução do mesmo de forma paralela na linguagem Java.

O PI (Π) é uma constante matemática originada a partir da proporção numérica existente entre o perímetro de uma circunferência e seu respectivo diâmetro. O valor exato de Π dessa constante é desconhecido, pois se trata de uma dízima periódica, mas é possível chegar a estimativas com boa qualidade de precisão na prática.

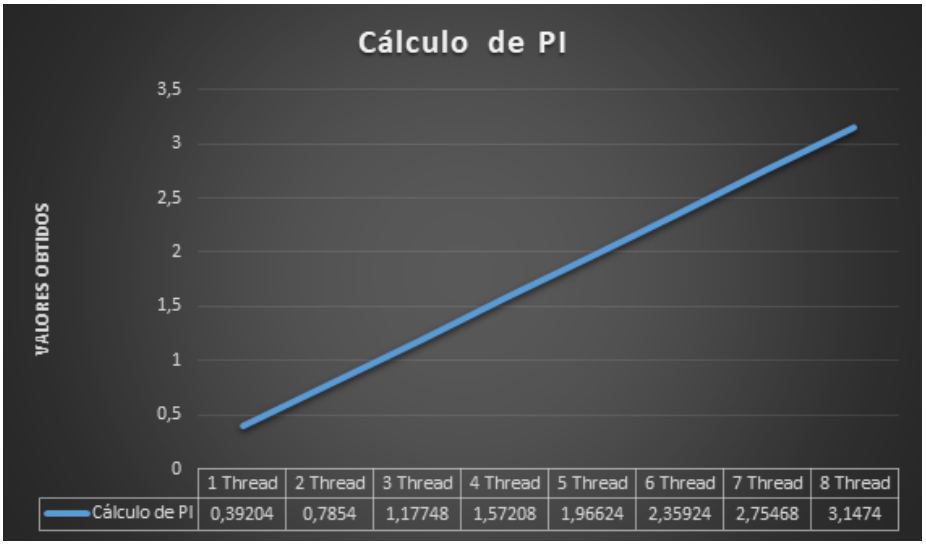
Um desses meios, é o método de Monte Carlo, que utiliza uma alta quantidade de dados estocásticos (pontos) para se obter estatisticamente a resposta de um sistema. Quanto maior a amostra de dados, maior a precisão do resultado. Para o cálculo de Π, é definido um círculo de raio r no interior de um quadrado de comprimento 2r. Utilizando uma relação de proporção entre áreas do círculo e quadrado, chega-se a seguinte fórmula: Π = 4 * M / N. Onde M é os pontos distribuídos dentro do círculo (acertos) e N os pontos totais.

A API Streams Java permite processar um fluxo de dados de forma declarativa, onde há os elementos de entrada, o processamento dos mesmos e a preparação para saída. Essa API oferece uma forma prática de criar e manipular java threads, usados para melhorar o desempenho na resolução de diversos tipos de cálculos que podem ser paralelizados, como é o caso do problema descrito acima. As Streams paralelas são capazes de processar resultados concorrentemente usando múltiplas threads e em seguida combinar as saídas. Para utilizar essa API, basta gerar uma coleção com os dados e definir a quantidade de threads na função *parallel(),* a qual processa a fração da coleção dividida pelo número de threads. Por padrão, o número de threads disponíveis em Stream paralelas é o mesmo que o número de CPUs do seu ambiente.

O sistema desenvolvido buscou apresentar uma comparação de tempo e precisão obtida para duas soluções implementadas, uma paralela (com 8 chamadas de threads executada por 4 *workes*) e outra serial. Foi implementado um método que gera uma quantidade de 100.000 pontos distribuídos aleatoriamente para x e y entre intervalos 0 a 1. A partir dos pontos obtidos é possível calcular a distância entre esses dois valores utilizando o teorema de Pitágoras, e contabilizar o número de acertos (pontos dentro do raio da circunferência). Para evitar problemas de concorrência na variável que contabiliza o total de acertos (todas as threads), foi utilizado um bloqueio de sincronização.

Os processos paralelos são criados após a coleção gerada ser passada para a função de *parallel(),* já o balanceamento se dá pela divisão do total de pontos pela quantidade de threads chamadas (8 processos e 4 *works* com 12500 pontos para cada). O algoritmo sequencial, por sua vez, executou a porção toda com 1 thread. O valor de aproximação obtido para 8 processos e 100.000 pontos foi 3.1474 e todo o processo levou 25 milissegundos, já o algoritmo de sequencial obteve uma aproximação de 3.1466, no entanto, levou 121 milissegundos. Notável diferença de tempo entre os

dois métodos. O gráfico abaixo apresenta o valor de aproximação calculado ao final da operação de cada thread (lembrando que são os dados obtidos após a contabilização do valor acumulado de cada thread).



	1 Thread	2 Thread	3 Thread	4 Thread	5 Thread	6 Thread	7 Thread	8 Thread
Cálculo de PI	0,39204	0,7854	1,17748	1,57208	1,96624	2,35924	2,75468	3,1474

Abaixo foi apresentado o código fonte.

```java
import java.util.stream.Stream;
import java.util.Arrays;
import java.time.LocalTime;

public class PIMC {
    private final static Object lock = new Object(); // evita problemas de concorrência entre thread
    private final static int QuantPontos = 100000;
    private static int total = 0; // valor acumulado

    //estrutura Stream
    private static Stream<Integer> processar(int nunThread) {
        Integer[] Pontos = new Integer[nunThread]; //cria uma coleção Pontos de tamanho especificado (no caso paralelo 8).
        Arrays.fill(Pontos, QuantPontos / nunThread); //Atribui o valor do tipo de dados a cada elemento.
        if (nunThread > 1)
            return Arrays.stream(Pontos).parallel(); //chama o método de paralelismo
        return Arrays.stream(Pontos).sequential(); //método sequencial
    }

    public static void main(String[] args) {
        System.out.println("-------\nSEQUENCIAL\n-------");
        chamaPI(processar(1));

        System.out.println("-------\nPARALELO\n-------");
        chamaPI(processar(8));
    }

    public static void chamaPI(Stream<Integer> stream) {
        long timeInicial = System.currentTimeMillis();
        total = 0;
        stream.forEach(s -> {
            System.out.println(LocalTime.now() + " - QuantPontos: " + s + " - thread: " + Thread.currentThread().getName());
            int n = 0;
            for (int i = 0; i < s; ++i) {
                double x = Math.random();
                double y = Math.random();
                if (x * x + y * y <= 1) { //tamanho do quadrado: 1 x 1
                    n++; // conta os pontos acertados
                }
            }
            synchronized (lock) { // proteção de condição de corrida
                total += n;
                System.out.println("PI = " + total * 4.0 / QuantPontos);
            }
        });
        long timeFinal = System.currentTimeMillis();
        System.out.println("-------------------------------------");
        System.out.println((timeFinal - timeInicial) + " ms");
        System.out.println("PI = " + total * 4.0 / QuantPontos);
        System.out.println("DIFERENÇA EXATA DE PI: " + ((total * 4.0 / QuantPontos) - Math.PI));
        System.out.println("ERRO: " + ((total * 4.0 / QuantPontos) - Math.PI) / Math.PI * 100);
    }
}
```

Resultado Obtido:

```
-------
SEQUENCIAL
-------
22:19:37.029 - QuantPontos: 100000 - thread: main
PI = 3.13088
---------------------------------------
151 ms
PI = 3.13088
DIFERENÇA EXATA DE PI: -0.01071265358979323
ERRO: -0.340994354489346
-------
PARALELO
-------
22:19:37.051 - QuantPontos: 12500 - thread: main
22:19:37.055 - QuantPontos: 12500 - thread: ForkJoinPool.commonPool-worker-1
PI = 0.39176
22:19:37.057 - QuantPontos: 12500 - thread: ForkJoinPool.commonPool-worker-1
PI = 0.78572
22:19:37.070 - QuantPontos: 12500 - thread: main
PI = 1.17712
22:19:37.060 - QuantPontos: 12500 - thread: ForkJoinPool.commonPool-worker-3
22:19:37.073 - QuantPontos: 12500 - thread: main
PI = 1.56748
22:19:37.076 - QuantPontos: 12500 - thread: ForkJoinPool.commonPool-worker-3
PI = 1.96108
PI = 2.35844
PI = 2.74892
22:19:37.062 - QuantPontos: 12500 - thread: ForkJoinPool.commonPool-worker-2
PI = 3.14268
---------------------------------------
37 ms
PI = 3.14268
DIFERENÇA EXATA DE PI: 0.001087346410206802
ERRO: 0.034611311207528056
```