Trabalho Prático de Programação Concorrente

Este trabalho prático tem como principal objetivo a utilização das capacidades de programação concorrente em Java para a resolução de um problema computacional intensivo.

Descrição do Problema

Considere o problema de calcular o valor de PI, definido pela divisão do valor da circunferência pelo valor do diâmetro de um círculo. Existem vários algoritmos matemáticos que permitem estimar o valor de PI com base em cálculos. Considere o método de Monte-Carlo[1], e o método de Séries de Gregory-Leibniz[2].

O método de Monte-Carlo consiste em gerar aleatoriamente uma grande quantidade de pontos numa área de largura e altura igual a 1 (as coordenadas x e y variam entre 0 e 1). O valor de PI é o produto de 4 pela divisão entre a quantidade de números gerados com uma distância inferior a 1 unidade a partir da raíz (0,0) e a quantidade total de números gerados.

O método de Séries de Gregory-Leibniz é calculado através da fórmula:

$$\pi = 4\left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right] = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}.$$

Pode ser implementado com o seguinte algoritmo:

```
1        double factor = 1.0;
2        double sum = 0.0;
3        for (k = 0; k < n; k++) {
4           sum += factor/(2*k+1);
5           factor = -factor;
6        }
7        pi_approx = 4.0*sum;
```

Ao paralelizar este algoritmo, é necessário ter em atenção o potencial conflito na variável partilhada "factor", que deverá ser substituída pelo seguinte código:

```
1           sum += factor/(2*k+1);
2           factor = -factor;
```

by

```
1           if (k % 2 == 0)
2              factor = 1.0;
3           else
4              factor = -1.0;
5           sum += factor/(2*k+1);
```

[1] https://www.geeksforgeeks.org/estimating-value-pi-using-monte-carlo/
[2] https://www.mathscareers.org.uk/article/calculating-pi/

Implementação do Problema (3 + 3 + 4 + 4 vals)

1) Desenvolva uma aplicação sequencial (i.e., sem concorrência) para o cálculo de PI usando o método de Monte-Carlo que recebe como entrada a quantidade de pontos a gerar e produz como resultado o valor estimado de PI.
2) Desenvolva uma aplicação sequencial (i.e., sem concorrência) para o cálculo de PI usando o método de Séries de Gregory-Leibniz que recebe como entrada a quantidade de iterações *k* a calcular e produz como resultado o valor estimado de PI.
3) Apresente uma versão concorrente da aplicação com o método de Monte-Carlo, sendo que o programa deverá receber como entrada e produzir como resultado a mesma informação do programa sequencial. Adicionalmente, esta versão concorrente deve permitir também especificar como parâmetro de entrada o número de *threads*.
4) Apresente uma versão concorrente da aplicação com o método de Séries de Gregory-Leibniz, sendo que o programa deverá receber como entrada e produzir como resultado a mesma informação do programa sequencial. Adicionalmente, esta versão concorrente deve permitir também especificar como parâmetro de entrada o número de *threads*.

Para optimizar o desempenho concorrente, tenha em atenção os seguintes pontos de valorização:

- será valorizada a utilização de técnicas de programação que garantam a correcção do programa, necessário para produzir sempre o mesmo resultado determinístico, assim como o desempenho do programa, através da utilização efetiva da concorrência (versus uma linearização da concorrência);
- o programa deve permitir especificar os parâmetros de entrada como argumento da linha de comandos ou leitura na consola;

Análise de Desempenho (1 + 1 + 2 + 2 vals)

1) De modo a medir o desempenho das implementações anteriores, acrescente no código das 4 aplicações anteriores um mecanismo de medição do tempo de execução.
   - durante a medição do tempo de execução não deve haver qualquer output para a consola.
   - o cálculo do tempo de execução na versão paralela deverá incluir a criação das *threads*.
   - a recolha do tempo de execução de cada aplicação deve ser uma média de várias execuções, configurável (por exemplo: 10), em vez de uma única execução.
2) Com base nos programas desenvolvidos nos pontos anteriores, construa uma tabela e um gráfico onde representa no eixo dos YY o tempo de execução do programa, e no eixo dos XX o nº de *threads* (assuma como exemplo: sequencial, PC1, PC2, PC4, PC8, PC10, PC20).
   - Escolha o valor parametrizável da quantidade de pontos a gerar (Monte-Carlo) e de número de iterações k (Séries de Gregory-Leibniz) de modo a que a versão sequencial

demore, pelo menos, 1 minuto a executar. Use o mesmo valor na versão concorrente. Indique o valor escolhido para estes parâmetros.

3) Compare o desempenho da execução concorrente com a execução sequencial, calculando o valor de *Alfa* (percentagem de código sequencial do programa) de acordo com a Lei de Amdahl, tomando como referência os ganhos de desempenho obtidos. Indique o tempo de execução medido para os casos sequencial e concorrente. Faça o cálculo para a melhor das versões concorrente (indicando o nº de *threads* assumido).
   - Indique o nº de processadores presentes na máquina usada para os resultados.
4) Comente os resultados dos tempos de execução para as versões sequencial e concorrentes, nos dois métodos da aplicação, apresentando justificações para os valores obtidos.

Entrega

A entrega deste trabalho prático consistirá num relatório onde deve apresentar uma explicação para cada item anterior, assim como um extrato do código relevante para esse mesmo item. O relatório deverá estar no formato PDF.

Cada programa (sequencial e paralelo) deve estar contido num único ficheiro de código fonte Java.

Todos os ficheiros (relatório + código-fonte) devem ser enviados num arquivo compactado do tipo **ZIP**, com a identificação dos elementos do grupo, número e nome.