

**Trabalho Prático 2 - Programação Distribuída**

Ricardo Dias nº11597

Daniel Correia nº17497

Bruno Rodrigues nº9969

10/07/2019

**Resumo**

Neste Trabalho Prático Programação Concorrente o desafio proposto pelo docente passava por desenvolver um programa em java que fosse capaz de calcular o valor aproximado de pi através de dois métodos diferentes sendo eles método de Monte-Carlo, e o método de Séries de Gregory-Leibniz, sendo que devem existir 2 versões diferentes para cada um dos métodos uma versão sequencial e uma versão concorrente.

Posteriormente será analisado o resultado obtido em ambas as versões.

**Índice**

[1. Introdução 5](#_Toc11705467)

[2. Desenvolvimento 7](#_Toc11705468)

[3. Conclusões 13](#_Toc11705469)

**Índice** **Figuras**

[Figura 1 – Função IsPar. 7](#_Toc11759385)

[Figura 2 - Ciclo sequencial Séries de Gregory-Leibniz 7](#_Toc11759386)

[Figura 3 – getPi. 8](#_Toc11759387)

[Figura 4 - Cálculo de Pi Sequencial Monte-Carlo 8](#_Toc11759388)

[Figura 5 – Método “run” versão concorrente de Séries de Gregory-Leibniz 9](#_Toc11759389)

[Figura 6 - Criação de threads 10](#_Toc11759390)

[Figura 7 - Metodo run do metodo de Monte-Carlo 10](#_Toc11759391)

1. Introdução

Neste Trabalho Prático o desafio proposto pelo docente passava por desenvolver um programa em java, capaz de calcular o valor aproximado de pi através de dois métodos diferentes sendo eles método de Monte-Carlo, e o método de Séries de Gregory-Leibniz utilizando o modelo de actores e modelar a distribuição do cálculo computacional pelos vários actores.

Para o desenvolvimento foi utilizada a ferramenta akka, é um kit para criar um aplicações dirigidas à programação concorrente e distribuída utilizando comunicação por mensagens.

2. Desenvolvimento

Gregory-Leibniz

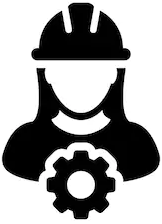
A aplicação recebe como entrada a quantidade de pontos a gerar para estimar o valor final, assim como o número de cálculos concorrentes a suportar. Deve produzir como resultado o valor estimado de PI.

Actor

Um ator é uma unidade fundamental de computação, é uma classe imutável que permite:

1. Enviar mensagens;
2. Alterar estado/comportamento;
3. Criar novo actores.

Na nossa aplicação será responsável por enviar ao coordenador o cálculo do seu resultado, já que eles precisam de menos recursos, é fácil de criar até milhares deles, neste caso o número pretendido de actores e será valor introduzido pelo utilizador.

 Controlador

Será criado 1 controlador, que após receber a ordem da main, cria os atores e os inicia, e após recolher todas as mensagens dos actores, vai devolver à main a resposta final.

 Router

Partes do processo é automatizado pela implementação akka, o router e gerir o envio assíncrono das mensagens.

Dispatcher

Dispatch faz a gestão da atribuição de threads com os actores

2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| main |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Cada actor irá conter estado que pode ser alterado, mas nunca partilhado com outros actores, podem executar cálculo de novos valores e enviar novas mensagens.

Todo o sistema é controlado por um actor coordenador que envia mensagens para os restantes, aguardando pelas suas respostas. Como as mensagens são imutáveis não existem problemas de sincronização.

3)

4)

3. Conclusões

Com este trabalho conseguimos aplicar os conhecimentos adquiridos na unidade curricular Programação Concorrente e Distribuída, é com muito prazer do grupo que conseguimos concluir todas as etapas apresentadas pelo docente neste trabalho, foi com pena nossa de que alguns dos resultados obtidos não tenham sido de acordo com o esperado.

