

**Trabalho Prático 2 - Programação Distribuída**

Ricardo Dias nº11597

Daniel Correia nº17497

Bruno Rodrigues nº9969

10/07/2019

**Resumo**

Neste Trabalho Prático Programação Distribuída 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, utilizando o modelo de Actores.

**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 biblioteca akka, que permite a utilização do modelo de actores.

2. Desenvolvimento

Gregory-Leibniz

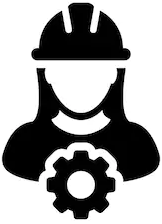
A aplicação recebe como entrada a quantidade de pontos a gerar para estimar o valor final, assim como o número de cálculos concorrentes a suportar. Deve produzir como resultado o valor estimado de PI.

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 controlador 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 o 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.

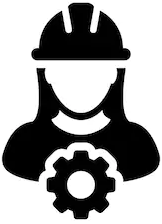
Dispatcher

Dispatch faz a gestão da atribuição de threads com os actores

**2)**

Main

Controlador



Router



Actor

…

…

inbox**.**send**(**controlador**,** **new** CalculaIntervaloMessage**(** N**,** PIcalc**));**

actorPI**.**tell**(new** CalculaPIcalcMessage**(**i**),** getSelf**()** **);**

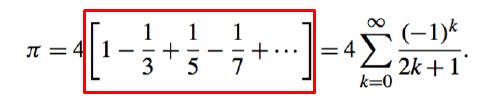
actorIniciador**.**tell**(new** PIMessage**(**pi**),** getSelf**());**

getSender**().**tell**(new** RespostaPIcalcMessage**(**f**),**getSelf**());**

N-Número de valores a considerar.  
PIcalc -Actor.  
i – Valor da interação de 0 a X(N).  
f – Valor calculado de cada Actor.  
pi – Valor final de PI.

**3)**

O Main após dar order de início ao coordenador, este envia a mensagem para cada actor calcular a sua parte do código identificada como concorrente, identificado a vermelho:



À medida que terminam o cálculo é enviado ao coordenador(controlador) o resultado e este trata de agrupar o resultado, e apresentar o resultado final.

**4)**

O Main é a função que permite inicializar o processo importante para instanciar os actores envolvidos e passar a variável de entrada.

Após terminar o processamento o coordenador envia mensagem ao Main que dará depois como concluído o programa.

Monte-Carlo

Ponto importante, a forma como o modelo foi estruturado, foi igual ao método de Séries de Gregory-Leibniz, a principal diferença encontra-se no método utilizado por cada actor para fazer os cálculos, isto deve-se ao facto de ambos terem métodos diferentes de calcular pi.

Sendo neste caso é criado dois números aleatórios, e somando o valor do quadrado de cada um deles, e caso esse valor seja inferior ou igual a um, um valor duma variável passa a 1, caso contrário mantém-se 0, valor este que será depois passado pelo actor para o controlador e no controlador será feita a soma de cada um destes valores.

Valor este (SUM) que será fundamental para o cálculo de pi, pois a fórmula de calcular pi é a seguinte:

pi = 4.0 \*(SUM / Número total de iterações);

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.

Como trabalho futuro, no caso de dar como entrada um grande número, e uma vez que o cálculo individual de cada actor é rápida, parte do tempo final de processamento fica à responsabilidade do coordenador deve tratar para somar todos os resultados. Seria possível atribuir um range a cada actor antes de devolver o resultado.

Em trabalho futuro fica por explorar a implementação de supervisão de actores, para garantir a solidez de código em caso de falha, com operações interessantes como restart,stop e resume de actores.

