Aluno: Bruno Libardoni

Cálculo do PI utilizando o método de aproximação por Monte Carlo

Metodologia

Para realizar o cálculo do PI pela aproximação por Monte Carlo, o autor utilizou a linguagem Java e as respectivas Threads para a resolução do problema. Foram utilizadas quatro classes principais que serão destacadas a seguir:

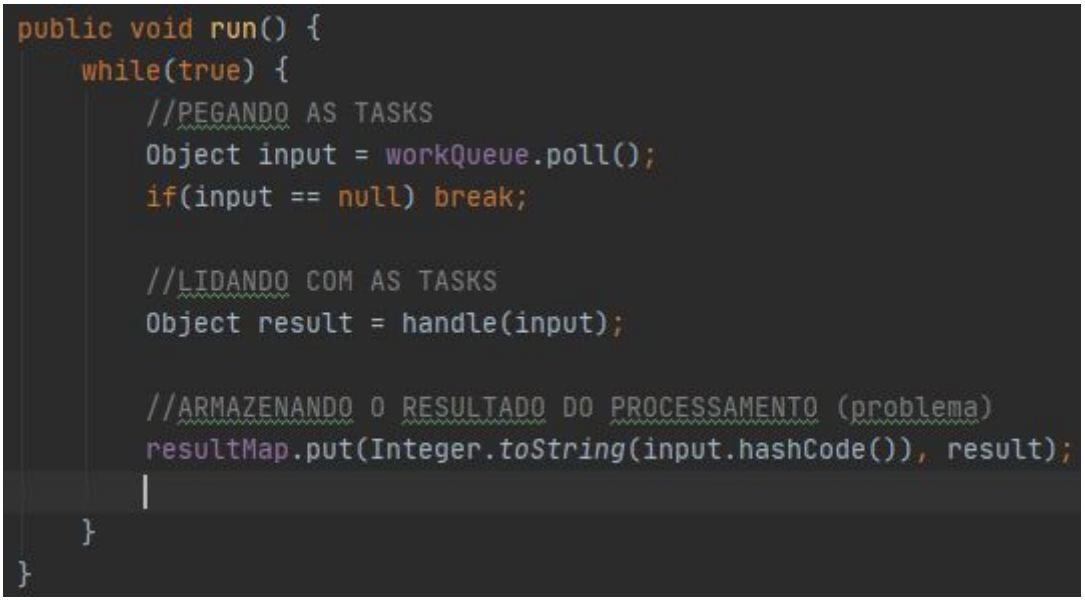
MASTER: Esta classe possui o intuito de iniciar o problema e enviar as tarefas aos respectivos workers. Na classe possui declaração de algumas filas utilizando o Java map (estrutura de dados que permite que seja ligado cada valor a uma chave) como por exemplo: Fila do processamento dos workers e fila da tarefa em si, juntamente com a declaração de uma lista que ficará salvo os resultados do processamento dos workers. Na classe também possui o método Master() que irá atribuir as tarefas aos workers e para cada worker uma thread.

Método Master() da Classe Master.

```
//Parametro o worker e a quantidades de workes q irao "trabalhar"
public Master(Worker worker, int countWorker) {
    worker.setWorkQueue(workQueue);
    worker.setResultMap(resultMap);
    for(int i=0; i<countWorker; i++) {
        threadMap.put(Integer.toString(i),
                new Thread(worker, Integer.toString(i)));
    }
}
```

WORKER: O worker irá receber as tarefas do Master e executá-las. Na classe possui algumas redeclarações como a fila das tarefas e o resultado onde ficará guardado as informações processadas. Na classe possui um objeto Handle que consiste em pegar o resultado final que cada worker irá retornar após a finalização do processamento e salvar em um objeto do tipo Java map todos os cálculos finais do PI que cada worker realizou, para posteriormente enviar para o Master.

Método do worker que pegará o retorno do PI realizado pelo execução e salvará no Objeto Map onde este objeto Map é repassado para o Master para posteriormente na main utilizar o método declarado na main getResultMap para pegar o PI que consta no Master.



```
public void run() {
    while(true) {
        //PEGANDO AS TASKS
        Object input = workQueue.poll();
        if(input == null) break;

        //LIDANDO COM AS TASKS
        Object result = handle(input);

        //ARMAZENANDO O RESULTADO DO PROCESSAMENTO (problema)
        resultMap.put(Integer.toString(input.hashCode()), result);

    }
}
```

TAREFA: A classe Tarefa consiste na lógica da tarefa que será enviada para o master através da Main e do master atribuirá a task para ser executada pelos workers. Esta classe consiste no cálculo do PI que recebe o número de pontos total, fará uma estrutura de repetição para até os pontos total e irá verificar se os pontos estão dentro do círculo através do eixo X e Y e <= 1. Após isso, será feito o cálculo dos pontos que caíram dentro do círculo juntamente com a fórmula de Monte Carlo e retornará o resultado para seu respectivo worker.

Lógica da classe Tarefa para realizar o cálculo do PI por Monte Carlo

```java
public Object handle(Object input) {

    int acertos = 0;
    int pontos = (Integer)input;

    for(int i = 0; i < pontos; i++){
        double x = Math.random();
        double y = Math.random();

        if(x * x + y * y <= 1.0){
            acertos++;
        }
    }
    System.out.println("Total de pontos dentro do circulo: "+acertos);

    return (double) (4.0 * acertos / pontos);
}
```

MAIN: Na classe main consiste basicamente em declarar o objeto Master e repassar as tasks para ele e então ele realizar os devidos repasses das informações aos workers e assim sucessivamente.

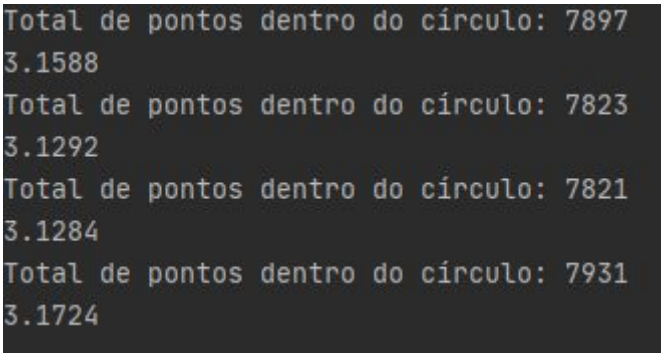
Lógica principal da main para enviar as informações para a Master e a classe realizar a chamada dos workers.

```java
public static void main(String[] args) {
    // FIXADO 4 WORKERS
    Integer work = 4;
    Master master = new Master(new Tarefa(), work);
    for(int i=1; i<=work; i++) { //enviar os pontos para os determinados workers
        master.submit( job: 10000000);
    }
    master.execute(); //executando a tarefa

    Map<String, Object> resultMap = master.getResultMap();
```

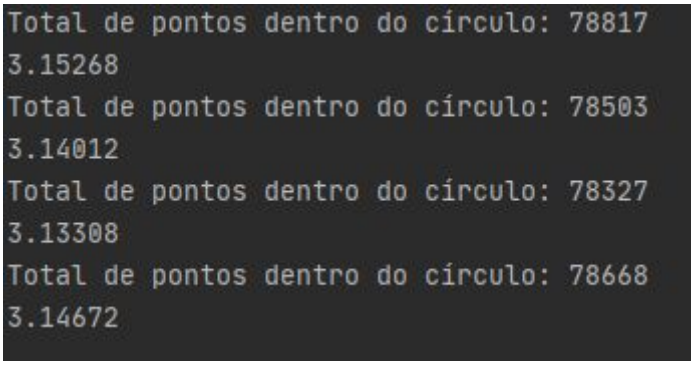
```java
while(true) {
    Set<String> keys = resultMap.keySet(); // são 10000 pontos então sao 10000 indices
    String key = null;
    for(String k : keys) {
        key = k; // colocando as chaves q estao sendo pega do resultmap, e colocando na key
        break;
    }
    Double i = null;
    if(key != null)
        i = (Double) resultMap.get(key);

    if(i != null)
        resul = i; //resultado final

    if(key != null)
        resultMap.remove(key);//removendo os itens que ja foram computados

    if(master.isComplete() && resultMap.size()==0)
        break;

}
```

Resultados

Para 10.000 pontos inseridos primordialmente juntamente com 4 workers consequentemente 4 threads, levou cerca de 4 segundos e os resultados obtidos constam na figura abaixo.



Para 100.000 pontos inseridos primordialmente juntamente com 4 workers consequentemente 4 threads, levou cerca de 7 segundos e os resultados obtidos constam na figura abaixo.



Para 10.000.000 pontos inseridos primordialmente juntamente com 4 workers consequentemente 4 threads, levou cerca de 6 minutos e os resultados obtidos constam na figura abaixo.