**Instituto Federal de Minas Gerais**

**Nome:** Marcus Vinicius Ribeiro Andrade.

**Matrícula:** 0064013.

**Tema:** Documentação do trabalho referente a programação paralela.

**Campus:** Ouro Branco, MG.

**Professor:** Saulo Henrique Cabral Silva.

**Período:** Terceiro**.**

**Turno:** Noturno.

**Introdução**

O processamento paralelo, cada vez mais, vem sendo visto como o único meio para a rápida solução de problemas computacionalmente grandes e intensivos GRAMA et al. (2003).Em decorrência disso, e com a redução no valor dos computadores, de modo geral, e, mais ainda, devido a muitas linguagens de programação hodiernas possuírem condições para que se possa trabalhar de forma paralela, é válido dizer que o uso de concorrência e paralelismo é cada vez mais apreciado, principalmente, por áreas recentes e que são, sobretudo, emergentes no mundo da tecnologia de informação.A saber, as áreas que usam, demasiadamente, programação paralela,são, por exemplo,o caso dos aplicativos que fazem processamento de transações que envolvem muita informação, a mineração de dados, os servidores web, os sistemas gerenciadores de bancos de dados (SGBDs),as inteligências artificiais, a nanotecnologia e muitas outras, que, em comum, partilham da realidade de se trabalhar com uma quantidade massiva de dados e executar, evidentemente, algoritmos bastante complexos.

Logo, diante do exposto, o presente trabalho visa esclarecer e implementar algoritmos de resolução de problemas que, por demais, podem ser bastante onerosos, no que tange ao tempo de execução e processamento de dados.As implementações, escritas na linguagem de programação Java, que, aqui, serão apresentadas ,visam usar o máximo do potencial computacional hospedeiro, isto é, objetivam evitar que o processador fique ocioso mediante a trabalhos pendentes.

Em detrimento disso, é imperioso ressaltar que para atingir o paralelismo nas atividades ,a seguir, conta-se apenas com o uso de threads, mais especificamente, multithreads, herdadas da classe Thread do java.E, primeiramente,antes de descrever, de fato, a implementação, torna-se importante, definir, de forma breve, o que são threads e seu papel no paralelismo.

Então, para que fiquem evidentes alguns conceitos, pode-se caracterizar programação paralela como a divisão da aplicação em partes e a execução simultânea dessas partes, por elementos de processamento, sendo esses, as threads, por exemplo, que são, por definição, um único fluxo do programa GRAMA et al. (2003), elas permitem a execução do programa fora da thread principal, main thread, que é criada ao executar o Java, por exemplo.Essas threads criadas, quando mais de uma, multithreads, permitem a execução paralela do sistema, o que, por sua vez, resulta em inúmeras vantagens e aplicações, como supracitado.

**Implementação Do Primeiro Programa:**

O primeiro programa visa encontrar o maior número primo, pela definição, um número é considerado primo quando é divisível por si próprio e , redundantemente, por um, contido em um arquivo de texto (extensão .txt).Logo, é claro que se trata, não apenas de um problema matemático , mas também, de uma questão sobre manipulação de arquivos. Assim, seguindo o conceito “dividir para conquistar”, tornou-se essencial racionar os dados para obter o melhor processamento paralelo.Em vista disso, como foi realizado a execução em uma máquina de seis núcleos (cores) e 2.90GHz de clock, dividiu-se todos os caminhos dos diretórios, pastas, de forma igualitária, para as threads criadas, no caso, seis, consoante a quantidade de núcleos do computador.

A organização do programa, vale ressaltar, espelha-se um pouco no modelo MVC (Model, Views, Controllers),mas, de forma simples, adaptada para ter um diretório controlador, um de dados, onde serão manipulados locais dos arquivos e , fora desses diretórios, a classe principal, Main.Diante disso, é válido dizer que o diretório Controllers possuí a parte lógica da aplicação e faz, inclusive, a aplicação das threads, o diretório Data contém a manipulação dos arquivos, no que tange ao path(caminho para o arquivo), e a classe Main invoca os métodos necessário para resolver os problemas.

Para uma visão mais abrangente da implementação e das estruturas abstratas de dados utilizadas, o diagrama abaixo ilustra a aplicação , como um todo.

AQUI VEM A ESTRUTURA DE DADOS

1. MAIN
2. CONTROLLER FILTER
3. CONTROLLER PARSER
4. CONTROLLER THREADS

Agora, consoante as estruturas de dados built-in (estruturas já existentes na linguagem), foram utilizadas, principalmente,os métodos da classe File, para manipulação de arquivos, variáveis dos tipos inteiro, string, double e booleana, ArrayList, Vector, List, métodos da classe Path e expressões regulares (regex).

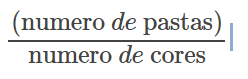
Para explicitar bem o funcionamento de cada método, abaixo está a descrição de cada um dos métodos implementados para resolução do problema.

**Classe Main e seus métodos:**

* **main(String[] args)**

Este método é padrão na linguagem java, é o chamado método principal e, no contexto do algoritmo, ele visa criar as instâncias (objetos) da classe ThreadsController para a criação de multithreads e, ainda, realizar a distribuição racionalizada dos dados, que , no caso, como supracitado são os diretórios.

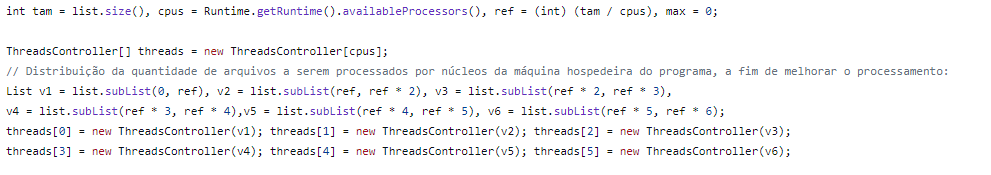
Essa divisão é feita quantitativamente, ou seja, dado um número x de núcleos,



no dispositivo, é feito a divisão de

Dessa maneira, pode-se ocupar o processador de forma equivalente e garantir que o mesmo não fique ocioso, enquanto a tarefa não terminar.Como consequência, têm-se também, o aumento expressivo do desempenho, ou seja, ocorre a redução do tempo de forma considerável.Contudo, o uso do processador torna-se exponencial, inviabilizando, durante o processo, a utilização de tarefas muito pesadas.

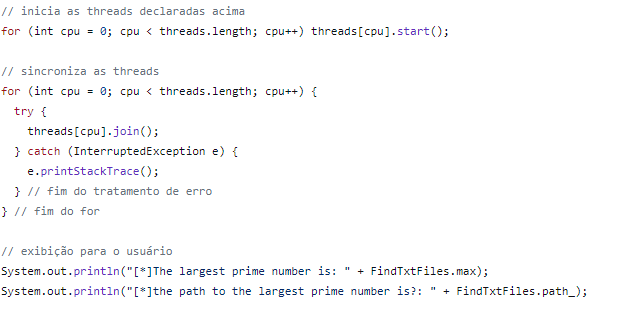
**Distribuição das pastas com os arquivos ,no método main , conforme a quantidade de núcleos**



Feito essas divisões, é chamado os métodos **start()** que dispara a execução da thread, mas, chamando o método run() - que logo será explicado, antes.

E também, é feito um loop, chamando o método **join(),** de cada thread que foi iniciada, para que se espere a thread terminar a execução, mantendo, desse modo, a sincronização.

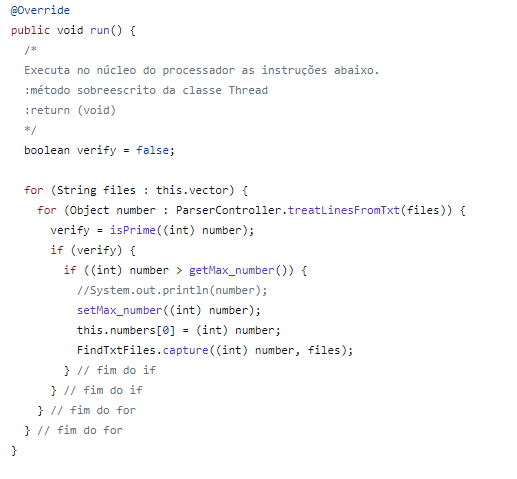
Vale a atenção de que, todos os demais programas, baseiam-se na mesma estrutura, no que tange a utilização das threads, ou seja, sempre se chamará os métodos acima de forma semelhante, como se segue na imagem abaixo:

**Uso comum de multithreads e exibição dos resultados**

**Classe ThreadsController e seus métodos:**

Explicado o método principal do programa, na classe ThreadsController, classe que herda os métodos da classe Thread do java, há a sobreescrita (@override) do método **run()**, que não recebe parâmetro e tem retorno do tipo void.Sumariamente, ela consiste dois loops, como os que percorrem matrizes,mas, no segundo, há a chamada do método estático da classe ParserController e a checagem se o número é primo ou não pelo método isPrime, da classe vigente.

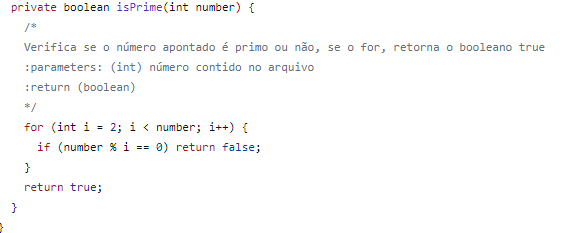
**Método run, sobreescrito, que é chamado pelo método start na classe Main**



* **isPrime(int number)**

Este método verifica se o número inteiro passado como parâmetro é primo ou não, retornando, desse modo, um valor booleano que indica se o é.

Sua construção é bem simples e visa, sobretudo, indicar se o número possui algum divisor, além disso, o método é privado e ,portanto, está disponível somente nesta classe:



**Método isPrime verifica se o inteiro passado é primo ou não, retornando um boolean correspondente**

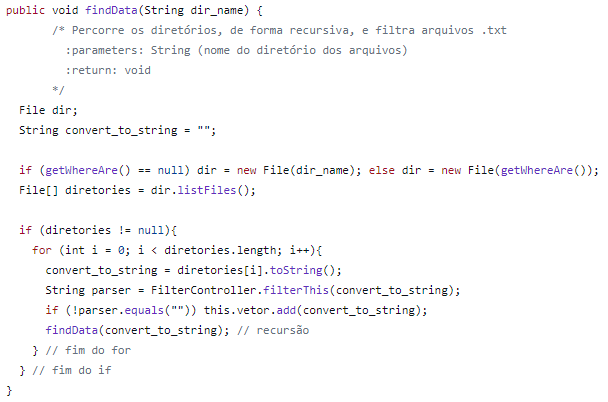
**Classe FindTxtFiles e seus métodos:**

Esta classe percorre o diretório indicado e seus sub-diretórios, de forma recursiva, pois, é de fácil percepção que escrever este método na forma recursiva permite, não apenas redução de código, mas também, ganho no desempenho.

* **findData(String dir\_name)**

O presente método recebe o nome do diretório como parâmetro e tem retorno do tipo void.Mais ainda, enquanto percorre os diretórios ,recursivamente, o caminho de cada arquivo passa pelo método **filterThis()** da classe FilterController, que logo será explicado,e,após serem feitas as modificações necessárias, é adicionado ao vetor do tipo de dado Vector.

Abaixo, está a imagem da implementação do método, na última linha, ele chama a si próprio, criando um ciclo recursivo, até o término da leitura dos arquivos.



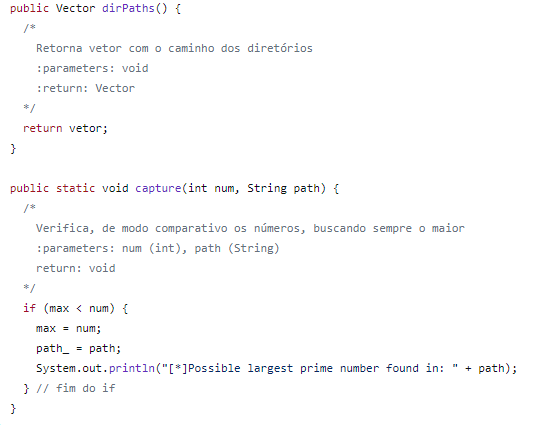
**Método findData que pesquisa arquivos recursivamente**

* **dirPaths()**

Método que retorna um vetor, do tipo Vector, e não recebe parâmetro.Este visa retornar o vetor com os caminhos do diretório.

* **capture(int num, String path)**

Método sem retorno que recebe um número inteiro, o número primo, e uma string com o path do mesmo.Este visa, exclusivamente, determinar o caminho e o maior número primo encontrado.Ou seja, é o método que retorna uma resposta para o problema.



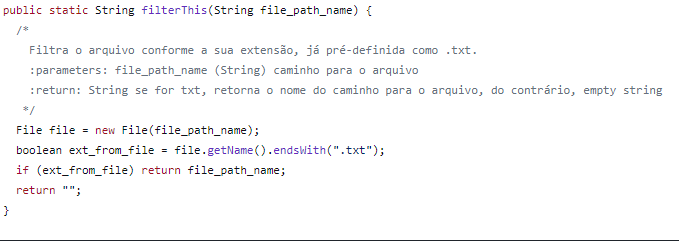
**Classe FilterController e seus métodos:**

Classe que contém o método responsável por considerar apenas arquivos do tipo texto (.txt) como proposto.

* **filterThis(String file\_path\_name)**

Este método é responsável pela separação dos arquivos de diferentes tipos, no repositório, para considerar, apenas, arquivos do tipo texto.Recebe, portanto, como parâmetro, o path do arquivo e analisa, por meio de manipulação de strings, quais arquivos terminam com .txt, retornando os que responde, afirmativamente, a questão.

**Método filterThis seleciona arquivos .txt**



**Classe ParseController e seus métodos:**

Finalmente, eis uma das classes mais importantes do programa, a classe parse contém o método responsável por analisar, minuciosamente, cada linha dos arquivos texto passados para ele, a fim de identificar números, para isso, é feito várias tratativas nos arquivos.

* **treatLinesFromTxt(String path\_and\_file)**