Computação Paralela e Concorrente

Prof. Carlos Duarte

Agenda

- Paralelismo de Matrizes
- Introdução a Computação Distribuída
- Exemplos
- Multiplicação Distribuída
- Multiplicação Serial vs Paralela
- Comparação

MergeSort

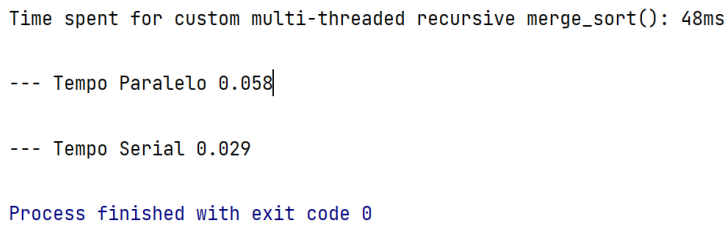
Vimos na aula passada que todo paralelismo tem um custo associado: um ***overhead***

```
Time spent for custom multi-threaded recursive merge_sort(): 48ms

--- Tempo Paralelo 0.058

--- Tempo Serial 0.029

Process finished with exit code 0
```

MergeSort

O paralelismo é vantajoso quando o tamanho do problema é grande o suficiente para compensar o overhead introduzido pelo uso de múltiplas threads. Neste caso, o tempo adicional devido ao overhead fez com que a execução paralela não fosse mais eficiente do que a serial para o tamanho do problema avaliado. Portanto, é essencial considerar o custo-benefício do paralelismo antes de aplicá-lo.

A imagem apresentada aborda o impacto do **overhead** no desempenho ao utilizar paralelismo em uma implementação de ordenação com merge_sort() de forma multi-threaded.



Análise da Imagem:

1.Tempo Paralelo (0.058s):
Esse é o tempo que o algoritmo paralelizado levou para concluir a execução. Embora o tempo seja maior do que o **tempo serial (0.029s)**, é importante notar que há um custo associado ao paralelismo.

2.Tempo Serial (0.029s):
Representa o tempo necessário para executar o mesmo algoritmo de forma sequencial, ou seja, sem dividir o trabalho em threads.

MergeSort

3.Overhead (48ms):
O valor destacado na parte superior da imagem indica o **tempo adicional** necessário para gerenciar threads, dividir tarefas e sincronizar os resultados entre as diferentes threads no ambiente paralelizado. Esse overhead pode ser significativo, especialmente em tarefas menores, onde o benefício do paralelismo não compensa o custo extra.

4.Explicação do Fenômeno:

Custo do Paralelismo: Ao implementar um algoritmo paralelizado, existem custos adicionais como:

- Criação e gerenciamento das threads.
- Comunicação e sincronização entre as threads.
- Divisão e combinação de subtarefas.

Limite do Benefício Paralelo: Quando o tamanho do problema é pequeno, como em arrays menores, o custo do overhead pode ser maior do que o ganho proporcionado pelo paralelismo, resultando em tempos mais longos comparados à execução serial.

Multiplicação de Matrizes

A relação entre paralelismo de matrizes e computação distribuída envolve a execução simultânea de operações de multiplicação ou processamento de grandes matrizes em múltiplos processadores ou máquinas, compartilhando recursos de maneira distribuída.

Isso melhora a performance e a eficiência, especialmente em tarefas que exigem grandes volumes de dados, permitindo que cada nó ou processador execute partes do cálculo de forma independente, aproveitando a capacidade de processamento paralelo da infraestrutura distribuída.

Multiplicação de Matrizes

Vamos analisar a possibilidade de paralelização para operações em matrizes, focando na multiplicação, uma das operações mais comuns e que oferece grande potencial de paralelismo. A multiplicação de matrizes pode ser dividida em partes independentes, o que torna possível a execução simultânea de cálculos para diferentes elementos da matriz resultante. Essa abordagem, quando combinada com técnicas de computação paralela, pode resultar em ganhos significativos de desempenho.

A paralelização de operações em matrizes, como a multiplicação, é possível devido à independência dos cálculos realizados em cada elemento da matriz resultante. Na multiplicação de matrizes A e B, o elemento $C[i][j]$ da matriz resultante é calculado como o somatório dos produtos das linhas de A com as colunas de B.

Multiplicação de Matrizes

Relembrando:

O número de colunas da primeira matriz
tem que ser igual
ao número de linhas
da segunda matriz.

$$A = \begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

$$A \times B = \begin{bmatrix} 2 & 3 \\ 4 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 3 & 0 \\ 2 & 1 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 2 \times 1 + 3 \times 2 & 2 \times 3 + 3 \times 1 & 2 \times 0 + 3 \times 1 \\ 4 \times 1 + 6 \times 2 & 4 \times 3 + 6 \times 1 & 4 \times 0 + 6 \times 1 \end{bmatrix} =$$

$$\begin{bmatrix} 2 + 6 & 6 + 3 & 0 + 3 \\ 4 + 12 & 12 + 6 & 0 + 6 \end{bmatrix} =$$

$$\begin{bmatrix} 8 & 9 & 3 \\ 16 & 18 & 6 \end{bmatrix}$$

Multiplicação de Matrizes

A paralelização deste problema pode ocorrer de duas maneiras:

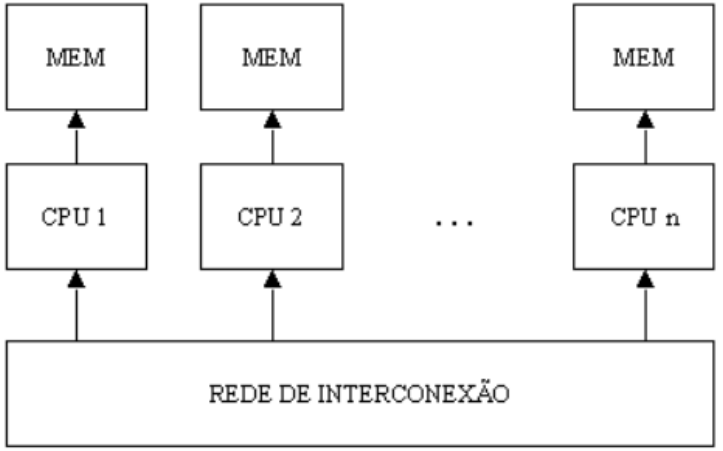
1º Em uma única máquina

2º De maneira ***distribuída***

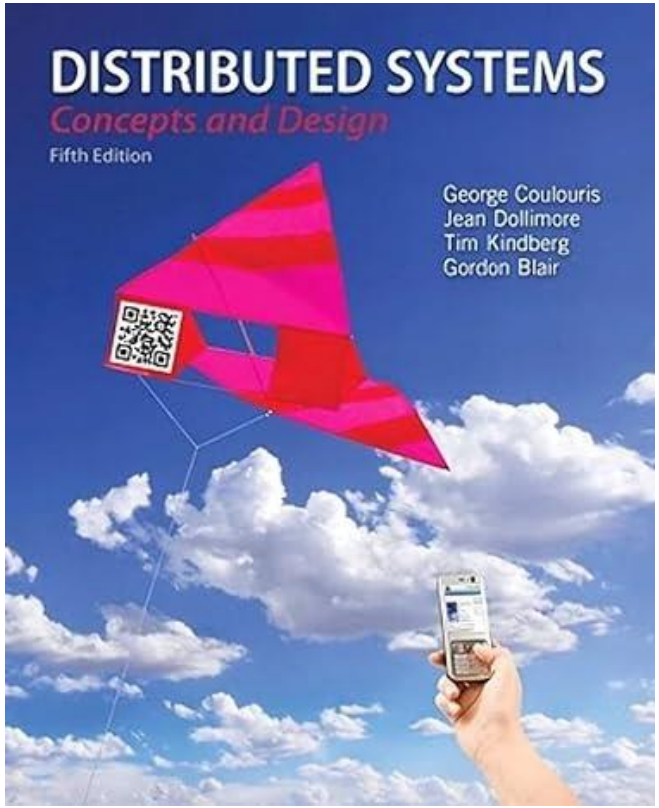
Vamos analisar primeiramente a segunda maneira

Multiplicação de Matrizes - Distribuída

Computação distribuída corresponde ao método de fazer **vários computadores trabalharem juntos** para resolver um problema comum. Isso faz com que uma rede de computadores pareça um único computador



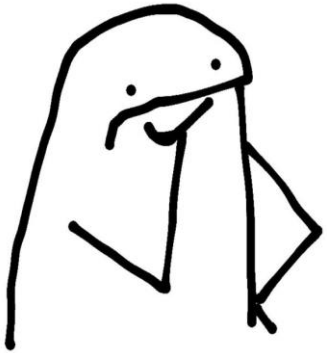
Computação Paralela vs Computação Distribuída



Um ***sistema distribuído*** é aquele em que os ***componentes estão localizados numa rede e coordenam-se suas ações trocando mensagens***.

Por outro, lado, ***sistema*** de processamento ***paralelo*** é aquele que ***várias computações são feitas simultaneamente***

Computação Paralela vs Computação Distribuída



“Todo sistema distribuído é paralelo, entretanto, um sistema paralelo pode não ser distribuído”

A frase significa que **sistemas distribuídos** sempre dividem tarefas entre múltiplos nós fisicamente separados (paralelismo distribuído), enquanto **sistemas paralelos** podem executar tarefas simultaneamente em processadores locais ou núcleos de um único computador, sem a necessidade de distribuição física.

Assim, todo sistema distribuído utiliza paralelismo, mas nem todo paralelismo ocorre em um ambiente distribuído.

Introdução a Computação Distribuída

Na computação distribuída, os elementos computacionais são descritos como nós (*nodes*), que mantém processos de software, capazes de enviar mensagens diretamente uns aos outros, estimulando-os a executar e concluir as operações

Sistemas distribuídos são dinâmicos (dispositivos podem juntar-se ou deixar uma unidade de execução a qualquer momento). Tratam-se de componentes que colaboram para realizar uma tarefa predeterminada, comunicando-se através do envio de mensagens, podendo referenciar máquinas físicas ou processos de software rodando em ***cloud***

Introdução a Computação Distribuída

"Uma coleção de elementos computacionais autônomos, trabalhando em conjunto, mas que aparentam ao usuário final sendo um sistema único" — *Distributed Systems, Steen e Tanenbaum*

Em um sistema distribuído bem elaborado, os usuários acreditam estar lidando com um sistema único para fins de processamento quando na verdade estão na presença destes nodes computacionais. Operando em conjunto, estes componentes entregam uma visão unificada ao usuário final. Sendo o ideal que este usuário sequer note que está lidando com processos e dados dispersos, controlados ao longo de uma rede de computadores distribuída.

Introdução a Computação Distribuída

Mas esse negócio de computação distribuída funciona mesmo?

1. World Wide Web (WWW): Exemplo mais conhecido de um sistema distribuído. Consiste De servidores web distribuídos globalmente que hospedam páginas da web e clientes (navegadores) que solicitam e exibem essas páginas. Os servidores e clientes se comunicam por meio do protocolo HTTP, permitindo o acesso a informações em todo o mundo.

Introdução a Computação Distribuída

Mas esse negócio de computação distribuída funciona mesmo?

2. O Kubernetes: Uma ferramenta popular pque pode criar um sistema distribuído usando uma coleção de contêineres. Os contêineres criam pontos centrais do sistema distribuído, e então o Kubernetes orquestra a comunicação de rede entre os pontos centrais e lida com o escalonamento horizontal e vertical dinâmico dos pontos centrais no sistema.

Introdução a Computação Distribuída

Mas esse negócio de computação distribuída funciona mesmo?

3. Redes de Sensores Sem Fio (Wireless Sensor Networks - WSN): Uma grande quantidade de sensores distribuídos que coletam dados ambientais ou de eventos e os transmitem por meio de uma rede sem fio. Eles são usados em aplicações como monitoramento ambiental, detecção de incêndios florestais e automação industrial.

Introdução a Computação Distribuída

Mas esse negócio de computação distribuída funciona mesmo?

4. Sistemas Peer-to-Peer (P2P): Redes P2P permitem que os computadores se comuniquem diretamente uns com os outros, em vez de dependerem de servidores centralizados. Aplicações P2P incluem compartilhamento de arquivos (como o BitTorrent)

Introdução a Computação Distribuída

Mas esse negócio de computação distribuída funciona mesmo?

5. Exemplos práticos, como as criptomoedas Bitcoin e Ethereum, demonstram como sistemas distribuídos ponto a ponto (peer-to-peer) podem ser eficazes e seguros na troca de dados e execução de transações financeiras sem a necessidade de uma autoridade central.

Introdução a Computação Distribuída

Mas esse negócio de computação distribuída funciona mesmo?

6. Sistemas de Redes Sociais: Redes sociais como Facebook, Twitter e LinkedIn são exemplos de sistemas distribuídos que operam em escala global. Eles incluem servidores distribuídos para armazenar perfis de usuários, publicações e mídia, além de recursos de escalabilidade para atender a milhões de usuários simultaneamente.

Computação Distribuída x Computação Paralela

Característica	Computação Paralela	Computação Distribuída
Definição	Realiza várias operações simultaneamente usando múltiplos processadores.	Usa vários computadores independentes para trabalhar em tarefas distribuídas.
Objetivo	Aumentar a velocidade do processamento de uma única tarefa.	Aumentar a capacidade de processamento distribuindo a carga entre vários sistemas.
Arquitetura	Usa um único sistema com múltiplos núcleos ou processadores.	Consiste em vários sistemas independentes conectados através de uma rede.
Gerenciamento de Tarefas	Tarefas são divididas e executadas simultaneamente em diferentes núcleos.	As tarefas são distribuídas entre diferentes máquinas ou nós, que se comunicam para compartilhar resultados.
Exemplos de Aplicações	- Processamento de imagens - Modelagem científica - Simulações de física	- Sistemas de bancos de dados distribuídos - Redes sociais - Computação em nuvem
Sincronização e Comunicação	Alta comunicação e sincronização entre núcleos/processadores para resultados compartilhados.	Menor sincronização entre máquinas, com troca de dados por meio de rede, podendo ocorrer latência.
Capacidade de Processamento	Depende da capacidade de múltiplos núcleos ou processadores dentro de um único sistema.	Capacidade de processamento é proporcional ao número de nós (máquinas) disponíveis na rede distribuída.
Flexibilidade em Rede	Baixa flexibilidade, pois as máquinas estão em um único sistema, com a comunicação limitada entre núcleos/processadores.	Alta flexibilidade, já que diferentes nós podem ser adicionados ou removidos da rede facilmente, escalando o sistema conforme necessário.

Multiplicação de Matrizes Distribuída

A multiplicação de matrizes pode ser realizada de maneira distribuída, e é uma abordagem comum em **computação paralela e distribuída**, especialmente para grandes matrizes. Em sistemas distribuídos, a tarefa de multiplicar duas matrizes é dividida entre múltiplos nós ou processadores, que trabalham simultaneamente para calcular as submatrizes e combinar os resultados.

Multiplicação de Matrizes Distribuída

- **Como funciona a multiplicação de matrizes distribuída:**

1. **Divisão da Matriz**: A matriz A (de tamanho m × n) e a matriz B (de tamanho n × p) são divididas em submatrizes menores. Cada submatriz será atribuída a um nó ou processador específico para calcular uma parte do produto final.

2. **Distribuição das Tarefas**: Cada nó realiza operações de multiplicação para um subconjunto específico da matriz resultante C (tamanho m × p). Isso é feito dividindo a matriz resultante em partes menores, permitindo que os cálculos sejam feitos de forma paralela.

3. **Combinação dos Resultados**: Após a execução das multiplicações em paralelo, os resultados parciais são coletados e combinados para formar a matriz final C.

Multiplicação de Matrizes Distribuída

- **Benefícios de uma abordagem distribuída:**

- **Aumento da performance**: A multiplicação de matrizes pode ser intensiva em termos de recursos computacionais, e a distribuição do trabalho entre múltiplos nós ou processadores reduz o tempo total de execução.

- **Escalabilidade**: Em sistemas distribuídos, a adição de mais nós pode melhorar a performance ao lidar com matrizes maiores.

- **Eficiência em sistemas grandes**: Para matrizes muito grandes (como as usadas em simulações científicas ou aprendizado de máquina), essa abordagem é essencial, pois o processamento de uma única máquina poderia ser muito lento ou até inviável.

Multiplicação de Matrizes Distribuída

- **Exemplos de uso em computação distribuída:**

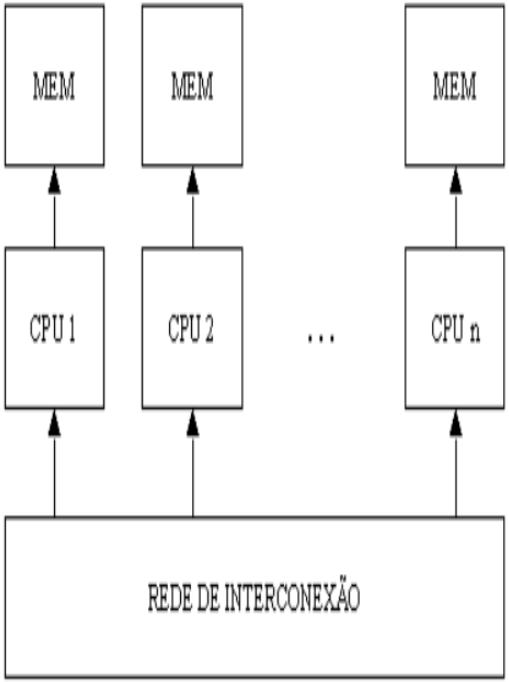
- **Hadoop e Spark**: Ambos são frameworks de computação distribuída que podem ser usados para realizar multiplicação de matrizes em um cluster de máquinas.

- **Plataformas de GPU**: Embora não sejam tradicionalmente "distribuídas" no sentido de múltiplas máquinas, as GPUs - **Unidade de Processamento Gráfico** (do inglês **Graphics Processing Unit**), podem ser vistas como um ambiente de paralelismo em massa onde as operações de multiplicação de matrizes podem ser realizadas em paralelo de forma eficiente.

Multiplicação de Matrizes Distribuída

Existem algoritmos que fazem uso da uso da multiplicação de matriz para ser executado em uma arquitetura de memória distribuída, onde cada processador possui memória local e se comunica com outros processadores por meio de troca de mensagens via rede.

Esse tipo de modelo é típico em algoritmos paralelos e distribuídos, como os utilizados em sistemas de computação em rede, como a arquitetura NOW (Network Of Workstations), que permite a execução simultânea de tarefas distribuídas entre múltiplos nós



Multiplicação de Matrizes Distribuída

Proposta de atividade prática em Python, utilizando a ideia de **multiplicação de matrizes distribuída** com **cliente-servidor**.

Objetivo da Atividade:
Implementar um sistema simples de **multiplicação de matrizes distribuída** em Python utilizando o conceito de **cliente-servidor**. O cliente distribuirá a carga de trabalho entre os servidores (módulos) e, após o processamento paralelo, receberá os resultados de cada servidor para compor o resultado final.

Tecnologias e Ferramentas:

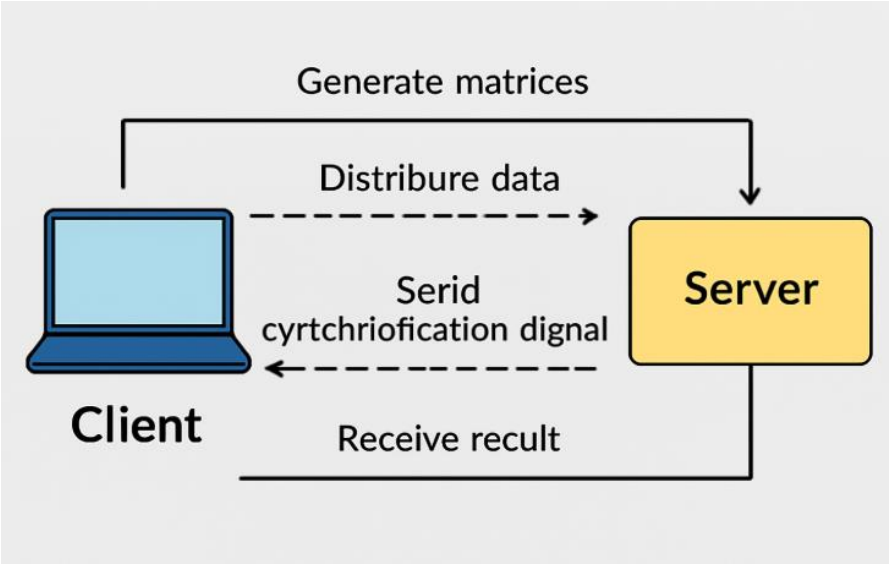
- **Python** para implementação.
- **Socket Programming** para comunicação entre cliente e servidores.
- **Multiprocessing** ou **Threading** para simulação de servidores paralelos.

Multiplicação de Matrizes Distribuída

OBS:

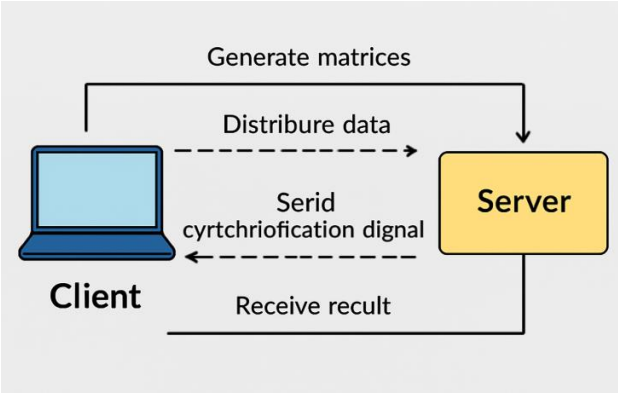
Multiprocessing refere-se ao uso de múltiplos processadores ou núcleos para executar tarefas em paralelo, permitindo que vários processos sejam executados simultaneamente.

Threading envolve a execução de múltiplas threads dentro de um único processo, permitindo a execução concorrente de várias partes de um programa, mas compartilhando o mesmo espaço de memória.



Multiplicação de Matrizes Distribuída

Estrutura da Atividade:



1. Implementação do Cliente (Client.py):
O cliente será responsável por:
1.Gerar as matrizes A e B.
2.Estabelecer a conexão com os servidores.
3.Enviar as submatrizes para os servidores.
4.Receber os resultados parciais de cada servidor.
5.Concatenar os resultados parciais e formar a matriz final.

2. Implementação do Servidor (Server.py):
Cada servidor realizará as seguintes tarefas:
1.Receber a submatriz de A e a matriz B do cliente.
2.Realizar a multiplicação da submatriz de A com a matriz B de maneira paralela.
3.Enviar o resultado parcial de volta para o cliente.

Multiplicação de Matrizes Distribuída

Passo a Passo da Implementação:

1. **Gerar Matrizes A e B no Cliente**:
O cliente gerará duas matrizes A e B de tamanhos adequados para serem multiplicadas. A matriz A será dividida em submatrizes e enviada para os servidores.
2. **Divisão da Carga de Trabalho**:
Cada servidor receberá uma "parte" da matriz A, multiplicando-a pela matriz B e retornando a parte do resultado.
3. **Execução de Multiplicação de Matrizes**:
A multiplicação das matrizes será feita entre os servidores, utilizando **multiprocessing** ou **threads**, para acelerar o processamento.
4. **Resultados**:
Após os cálculos serem feitos pelos servidores, o cliente compilará os resultados parciais de cada servidor para formar a matriz resultante C.

Multiplicação de Matrizes Distribuída

Explicação Passo a Passo:

1. **Divisão das Matrizes**:
    1. A matriz A é dividida em submatrizes iguais, que são enviadas para os servidores (neste caso, 2 servidores).

2. **Envio e Recebimento de Dados**:
    1. O **cliente** envia as submatrizes e a matriz B para os **servidores**.
    2. Cada **servidor** realiza a multiplicação da submatriz com a matriz B.

3. **Concorrência**:
    1. A comunicação entre cliente e servidores é feita via **socket**.
    2. O cliente envia as submatrizes para os servidores em paralelo, aumentando a velocidade do processamento.

Multiplicação de Matrizes Distribuída

4. Cálculo e Composição do Resultado Final:

•Após a multiplicação de cada submatriz no servidor, o cliente recebe os resultados parciais, os concatena e forma a matriz final C.

OBS: Quando a atividade for executada corretamente, o sistema deve exibir a matriz resultante C após a multiplicação das matrizes A e B.

Esse exercício demonstra como é possível implementar um algoritmo de multiplicação de matrizes distribuída, utilizando uma arquitetura cliente-servidor simples com Python. A paralelização e a distribuição do trabalho entre diferentes servidores permitem uma solução mais eficiente para operações com grandes volumes de dados.

Multiplicação de Matrizes Distribuída

```
# Função para enviar a matriz para os servidores e receber o resultado
def main():
    # Matrizes de entrada A e B
    A = np.array([[1, 0, -1], [4, -1, 2], [-1, 2, 4]])
    B = np.array([[-1, 2, -3], [5, -4, 2], [4, 1, 0]])

    # Dividir a matriz A em submatrizes
    num_servers = 2
    submatrices_A = split_matrix(A, num_servers)
```

Cód. Client: Exemplo da construção de Função Matriz.

Multiplicação de Matrizes Distribuída – EXEMPLO 2

Utilizando 4 processadores/máquinas

1. Cliente estabelece conexão com os 4 servidores;
2. Cliente divide e envia a submatriz *A* , com *n*/4 linhas, e a matriz *B* para cada servidor;
3. Servidor recebe a submatriz *A* e a matriz *B* e efetua a multiplicação;
4. Cliente envia sinal de sincronismo para servidor;
5. Servidor recebe sinal de sincronismo e envia submatriz resultante;
6. Cliente recebe e concatena as submatrizes para a matriz resultante.

Multiplicação de Matrizes Distribuída – EXEMPLO 2

$$(1 \quad 0 \quad 2 \quad 3)\begin{pmatrix} -1 & 1 & 2 & -3 \\ -5 & -4 & 2 & -2 \\ 3 & -1 & 0 & 2 \\ 1 & 0 & 4 & 5 \end{pmatrix} = (8 \quad -1 \quad 14 \quad 16)$$

$$(4 \quad -1 \quad 1 \quad 5)\begin{pmatrix} -1 & 1 & 2 & -3 \\ -5 & -4 & 2 & -2 \\ 3 & -1 & 0 & 2 \\ 1 & 0 & 4 & 5 \end{pmatrix} = (9 \quad 7 \quad 26 \quad 17)$$

$$(-2 \quad -3 \quad -4 \quad 2)\begin{pmatrix} -1 & 1 & 2 & -3 \\ -5 & -4 & 2 & -2 \\ 3 & -1 & 0 & 2 \\ 1 & 0 & 4 & 5 \end{pmatrix} = (7 \quad 14 \quad -2 \quad 14)$$

$$(-1 \quad 2 \quad 0 \quad 0)\begin{pmatrix} -1 & 1 & 2 & -3 \\ -5 & -4 & 2 & -2 \\ 3 & -1 & 0 & 2 \\ 1 & 0 & 4 & 5 \end{pmatrix} = (-9 \quad -9 \quad 2 \quad -1)$$

Cliente: Recebe as submatrizes resultantes em C :

$$\begin{pmatrix} 8 & -1 & 14 & 16 \\ 9 & 7 & 26 & 17 \\ 7 & 14 & -2 & 14 \\ -9 & -9 & 2 & -1 \end{pmatrix}$$

Multiplicação de Matrizes Serial

```
1 usage
public static void startSerial() {

    Date startTime = new Date();

    int[][] m1 = MatrixGeneratorUtil.generateMatrix( rows: 20,  columns: 20);
    int[][] m2 = MatrixGeneratorUtil.generateMatrix( rows: 20,  columns: 20);

    int[][] result = multiply(m1, m2);
```

```
for (int i = 0; i < resultRows; i++) {
    for (int j = 0; j < columns2; j++) {
        result[i][j] = 0;
        for (int k = 0; k < resultColumns; k++) {
            result[i][j] += matrix1[i][k] * matrix2[k][j];
        }
    }
}
```

Multiplicação de Matrizes Paralela “Não Distribuída”

```java
public static void startParallel() {

    Date start = new Date();

    int[][] m1 = MatrixGeneratorUtil.generateMatrix( rows: 20,  columns: 20);
    int[][] m2 = MatrixGeneratorUtil.generateMatrix( rows: 20,  columns: 20);

    System.out.println("Matrix 1 : ");
    MatrixGeneratorUtil.print(m1);

    System.out.println("\nMatrix 2 : ");
    MatrixGeneratorUtil.print(m2);

    System.out.println("\nOutput Matrix : ");
    int[][] result = new int[m1.length][m2[0].length];
    ParallelThreadsCreator.multiply(m1, m2, result);
```

Multiplicação de Matrizes Paralela “Não Distribuída”

```java
public static void multiply(int[][] matrix1, int[][] matrix2, int[][] result) {
    List<Thread>threads = new ArrayList<>();
    int rows1 = matrix1.length;
    for (int i = 0; i < rows1; i++) {
        RowMultiplyWorker task = new RowMultiplyWorker(result, matrix1, matrix2, i);
        Thread thread = new Thread(task);
        thread.start();
        threads.add(thread);
        if (threads.size() % 10 == 0) {
            waitForThreads(threads);
        }
    }
}
```

```java
public void run() {

    for (int i = 0; i < matrix2[0].length; i++) {
        result[row][i] = 0;
        for (int j = 0; j < matrix1[row].length; j++) {
            result[row][i] += matrix1[row][j] * matrix2[j][i];

        }

    }

}
```

Multiplicação de Matrizes - Comparação

Para 20 Linhas e 20 Colunas

```
Tempo Serial em  milissegundos: 29
```

```
Tempo Paralelo em  milissegundos: 11
```

Para 2000 Linhas e 2000 Colunas

```
Tempo Serial em  milissegundos: 70414
```

```
Tempo Paralelo em  milissegundos: 54252
```

Para 100 Linhas e 100 Colunas

```
Tempo Serial em  milissegundos: 180
```

```
Tempo Paralelo em  milissegundos: 135
```