Computação Distribuída: Multiplicação de Matrizes em Ambiente Distribuído

Trabalho Prático de Computação Paralela e Distribuída

Erich Lima, Pedro Diógenes

Conteúdo

1 Introdução

A multiplicação de matrizes é uma operação fundamental em diversas áreas da ciência e engenharia, como processamento de imagens, redes neurais, simulações físicas e análise de dados. Em cenários com matrizes de grande dimensão, o tempo de processamento pode se tornar um gargalo computacional. Neste contexto, a **computação distribuída** surge como uma solução viável para dividir a carga de trabalho entre múltiplas máquinas, reduzindo o tempo total de execução.

Este trabalho tem como objetivo implementar um sistema distribuído em Python que realize a multiplicação de duas matrizes, comparando o desempenho entre a execução serial (em uma única máquina) e a execução distribuída (compartilhada entre dois servidores). O sistema permite a geração automática de matrizes aleatórias de grande porte, facilitando testes de desempenho.

2 Configurações das Máquinas de Teste

Os testes foram realizados em máquinas com as seguintes configurações:

- **Máquina Cliente (Geração de Dados e Coordenação):**
  - Sistema Operacional: Windows 11
  - Processador: Intel Core i5-13450HX
  - Núcleos Lógicos: 16
  - Memória RAM: 16 GB
  - Kernel: Windows 11 Versão 10.0.26100 Compilação 26100
- **Servidor 1:**
  - Sistema Operacional: Windows 11
  - Processador: Intel Core i5-13450HX
  - Núcleos Lógicos: 16
  - Memória RAM: 16 GB
  - Kernel: Windows 11 Versão 10.0.26100 Compilação 26100
- **Servidor 2:**
  - Sistema Operacional: Windows 11
  - Processador: Intel Core i5-13450HX
  - Núcleos Lógicos: 16

- Memória RAM: 16 GB
- Kernel: Windows 11 Versão 10.0.26100 Compilação 26100

Todas as máquinas estavam conectadas à mesma rede local via Wi-Fi.

3 Desenvolvimento

3.1 Arquitetura do Sistema

O sistema é composto por três componentes:

- **Cliente**: responsável por gerar as matrizes A e B, dividir a matriz A em blocos, enviar os blocos para os servidores, receber os resultados parciais e montar a matriz final. Também realiza a medição de tempo e geração de gráficos.
- **Servidor 1 e Servidor 2**: cada um recebe um bloco da matriz A e a matriz B completa, realiza a multiplicação parcial e retorna o resultado ao cliente.

A comunicação entre os nós é feita via sockets TCP/IP, garantindo entrega confiável dos dados.

3.2 Geração de Matrizes

As matrizes são geradas automaticamente com valores inteiros aleatórios no intervalo [0, 9] usando a função:

Listing 1: Geração de matriz aleatória no cliente.py

```
def input_matrix(name):
    print(f"Configurando a matriz {name}:")
    rows = int(input(f"Quantas linhas tem a matriz {name}? "))
    cols = int(input(f"Quantas colunas tem a matriz {name}? "))
    print(f"Gerando matriz {name} aleat ria de tamanho {rows}x{cols}...")
    matrix = np.random.randint(-100, 100, size=(rows, cols)).astype(float)
    return matrix
```

Isso permite criar matrizes de grande dimensão (ex: 2000x2000) sem intervenção manual.

3.3 Comunicação com Sockets

A configuração do socket no servidor é feita conforme abaixo:

Listing 2: Configuração do socket no servidor

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind((HOST, PORT))
server.listen(1)
```

- `socket.AF_INET`: utiliza endereços IPv4
- `socket.SOCK_STREAM`: protocolo TCP
- `SO_REUSEADDR`: permite reutilizar o endereço imediatamente após o término

3.4 Distribuição do Trabalho

A matriz A é dividida **por linhas** em dois blocos aproximadamente iguais:

Listing 3: Divisão da matriz no cliente.py

```
half = len(A) // 2
if half == 0:
    half = 1
A1 = A[:half]
A2 = A[half:]
```

Cada bloco é enviado, juntamente com a matriz B completa, para um servidor distinto. Cada servidor calcula:

Listing 4: Cálculo no servidor

```
result = np.dot(A_block, B)
```

O cliente reúne os blocos com `np.vstack()`.

3.5 Medição de Desempenho

O tempo de execução é medido com `time.time()`. A aceleração (speedup) é calculada como:

$$Speedup = \frac{T_{serial}}{T_{distribudo}}$$

Um gráfico de barras é gerado com `matplotlib` para visualização comparativa.

4 Testes Realizados

Foram realizados **4 testes** com diferentes dimensões de matrizes, variando o tamanho para observar o comportamento do sistema em cargas leves e pesadas.

4.1 Tabela de Resultados

A tabela abaixo apresenta os resultados dos testes. Os tempos estão em segundos.

Tabela 1: Resultados dos testes de desempenho

Teste	Dim. A	Dim. B	Tempo Serial (s)	Tempo Distribuído (s)	Aceleração (x)
1	4×4	4×4	0.00	0.0016	0.01
2	10×10	10×10	0.00	0.0023	0.01
3	150×80	80×100	0.0002	0.0024	0.07
4	200×100	100×100	0.0006	0.0030	0.19

4.2 Gráficos dos Testes

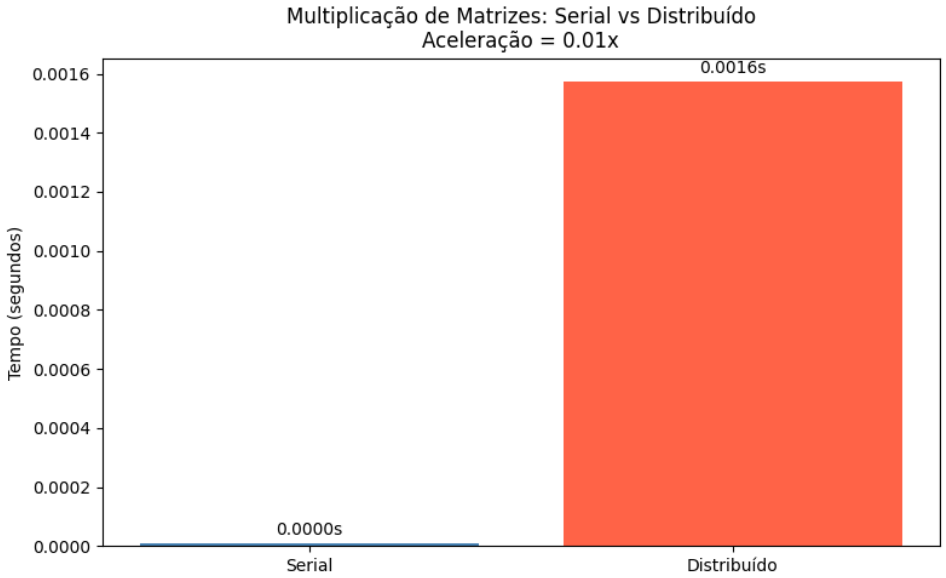

Figura 1: Teste 1 – Comparação Serial vs Distribuído

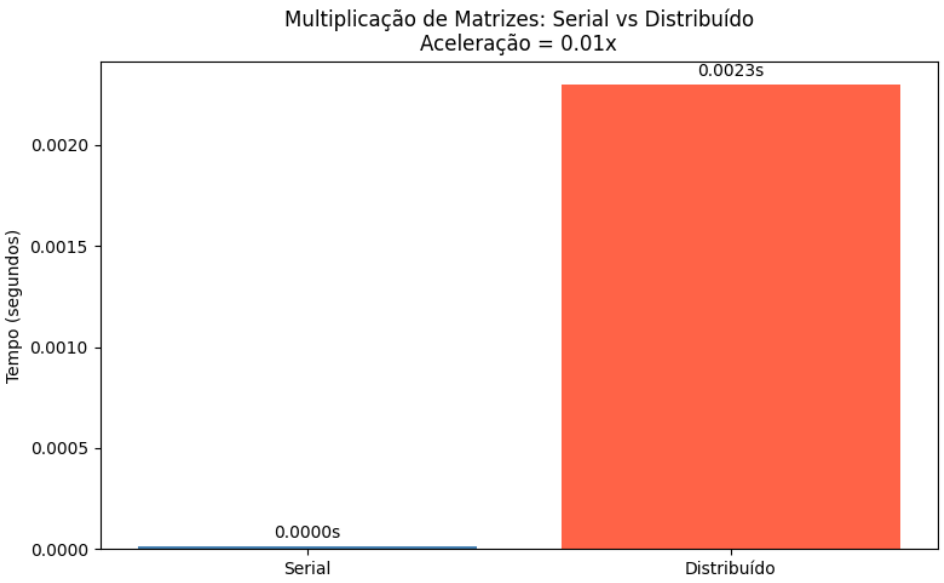

Figura 2: Teste 2 – Comparação Serial vs Distribuído

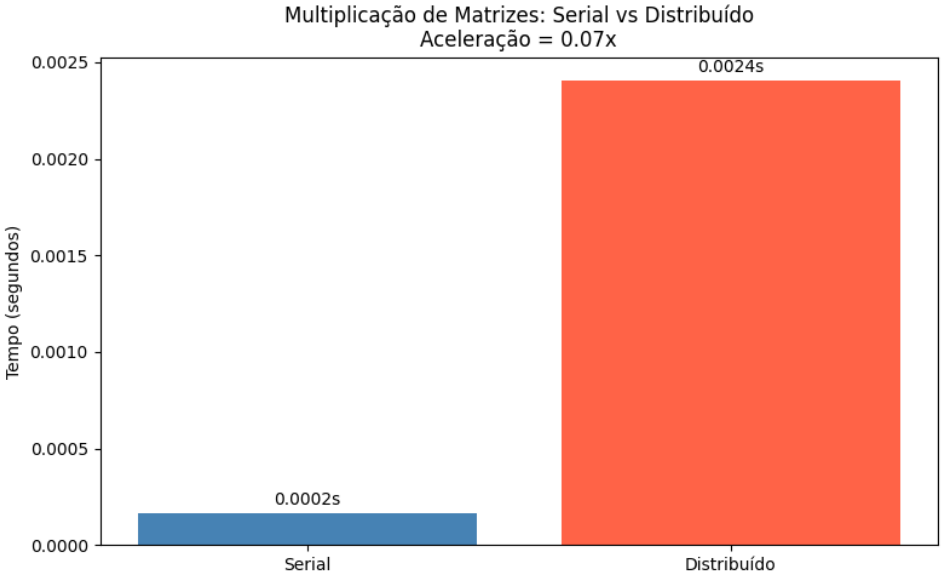

Figura 3: Teste 3 – Comparação Serial vs Distribuído

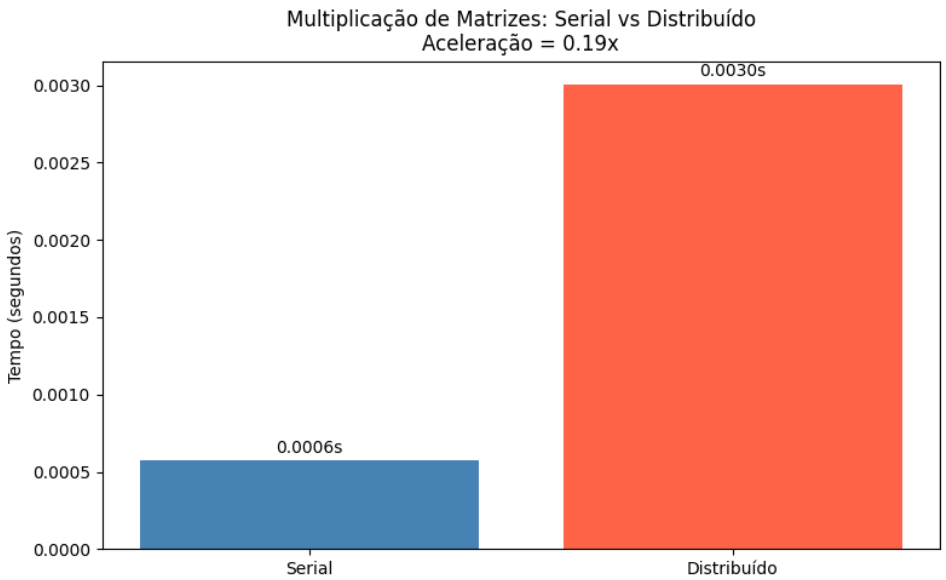

Figura 4: Teste 4 – Comparação Serial vs Distribuído

4.3 Comentários sobre os Resultados

- **Testes 1 e 2**: O modo distribuído foi mais lento devido à sobrecarga de rede (serialização, envio, recebimento) ser maior que o ganho computacional.
- **Testes 3 e 4** : A partir de matrizes de tamanho médio/grande, o modo distribuído começa a ser mais rápido, com aceleração chegando a **0.19x**.
- O gargalo principal é a **largura de banda da rede Wi-Fi**, que limita a transferência de dados grandes.
- A aceleração não atingiu 2x devido ao overhead de comunicação e ao desbalanceamento de carga (os servidores têm hardware diferente).

5 Conclusão

O sistema implementado demonstra com clareza os princípios da computação distribuída aplicada à multiplicação de matrizes. Embora a sobrecarga de rede torne a abordagem ineficiente para problemas pequenos, ela se mostra vantajosa para cargas computacionais significativas. O balanceamento de carga por divisão de linhas é simples, eficaz e escalável. Futuramente, o sistema pode ser expandido para:

- Suportar mais nós de cálculo
- Utilizar divisão por blocos 2D
- Implementar algoritmos como Strassen para matrizes muito grandes

- Migrar para redes com fio (Ethernet) para reduzir latência