Avaliação de desempenho

Exercício 2

2024.1

Alunos: Camila Xavier (CXM) & Vituriano Xisto (VOX)

1) Objetivo:

Realizar uma avaliação comparativa de desempenho das duas versões implementadas no Exercício 01: uma versão sem concorrência e uma versão concorrente usando mutex. Considerar matrizes de tamanhos distintos: 10x10, 100x100 e 1000x1000.

2) Listar os serviços do sistema:

- **Produção de eventos:**

  Criação de goroutines para cada linha da matriz A

```
for i := 0; i < numLinhasA; i++ {
  wg.Add(1)
  go func(i int) {
    defer wg.Done()
    for j := 0; j < numColunasB; j++ {
      for k := 0; k < numColunasA; k++ {
        mu.Lock()
        C[i][j] += A[i][k] * B[k][j]
        mu.Unlock()
      }
    }
  }(i)
}
```

- **Armazenar Eventos:**

  Uso de mutex para proteger a matriz C e WaitGroup para sincronizar a conclusão das goroutines.

Os métodos mu.Lock() e mu.Unlock() são utilizados para garantir que apenas uma goroutine possa acessar a seção crítica por vez. Esta seção crítica é onde a matriz C está sendo atualizada.

```
for i := 0; i < numLinhasA; i++ {
  wg.Add(1)
  go func(i int) {
    defer wg.Done()
    for j := 0; j < numColunasB; j++ {
      for k := 0; k < numColunasA; k++ {
        mu.Lock()
        C[i][j] += A[i][k] * B[k][j]
        mu.Unlock()
      }
    }
  }(i)
}
```

- **Consumir Eventos:**

  Goroutines que consomem os valores das matrizes A e B para calcular os elementos da matriz C.

```
  go func(i int) {
    defer wg.Done()
    for j := 0; j < numColunasB; j++ {
      for k := 0; k < numColunasA; k++ {
        mu.Lock()
        C[i][j] += A[i][k] * B[k][j]
        mu.Unlock()
      }
    }
  }(i)
}
```

3) **Escolher as métricas de desempenho:**

Medição de tempo de execução da aplicação

4) **Listar parâmetros:**

Parâmetro do Sistema	**Valor**
Hardware	1 vCPU, 512 MB Memory, 10 GB Disk
Sistema Operacional	Ubuntu 24.04 (LTS) x64
Linguagem de programação	GO
Fonte de alimentação	Rede elétrica (nuvem)
Processos em execução	Apenas os estritamente necessários à realização do experimento
Interfaces de rede	Desligadas

5) **Escolher fatores:**

Fator	**Nível**
Mecanismo de sincronização	Mutex
Número de produtores/consumidores	30x - 10x10, 100x100, 1000x1000

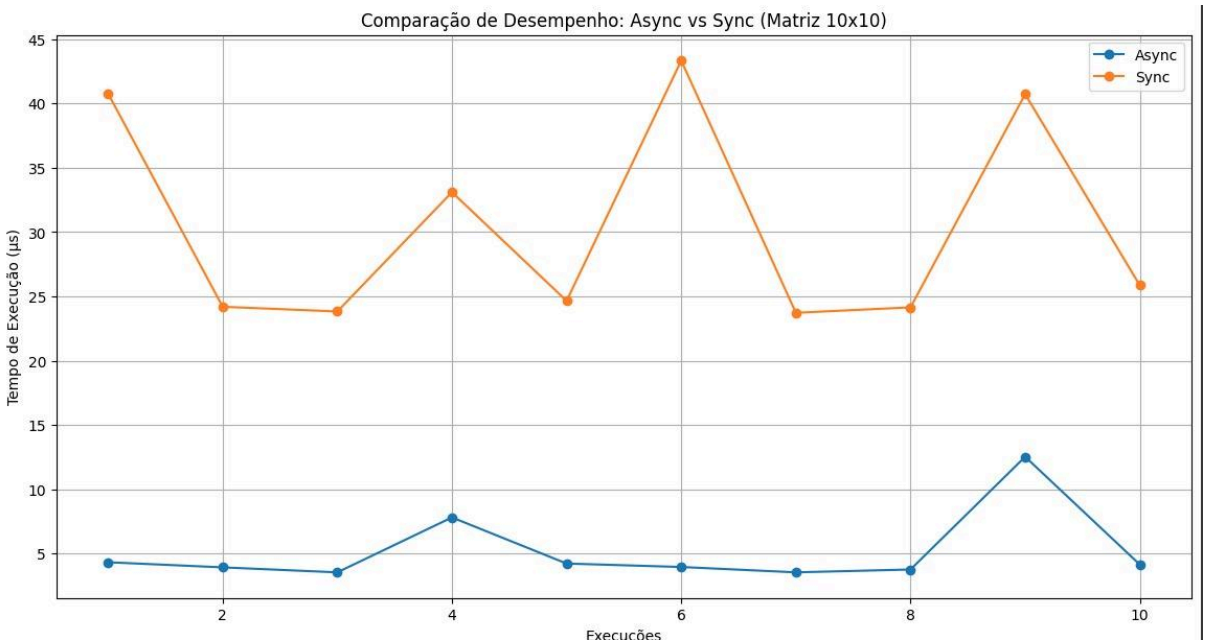
6) **Escolha a técnica de avaliação**

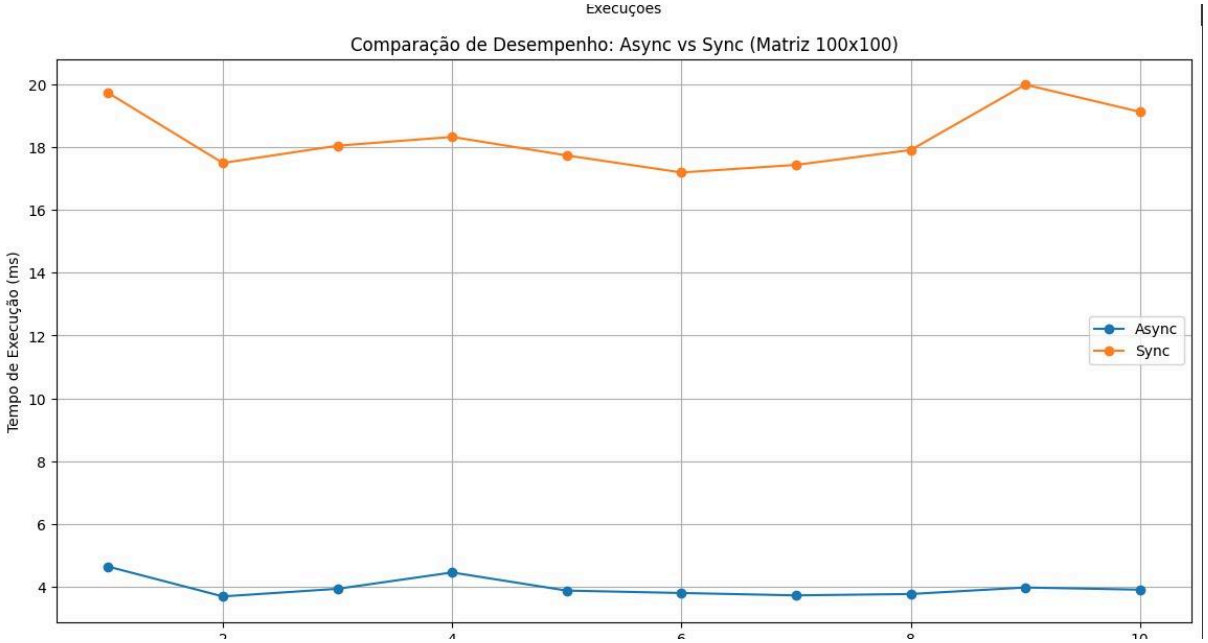
Medição

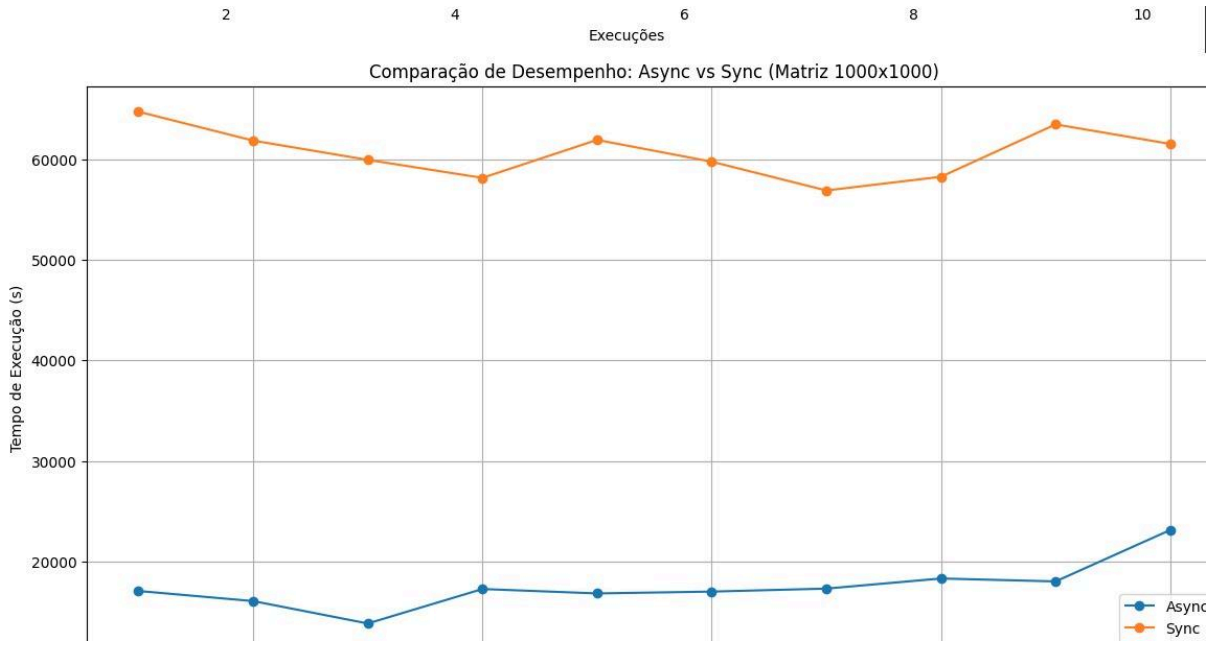
7) **Script do experimento:**

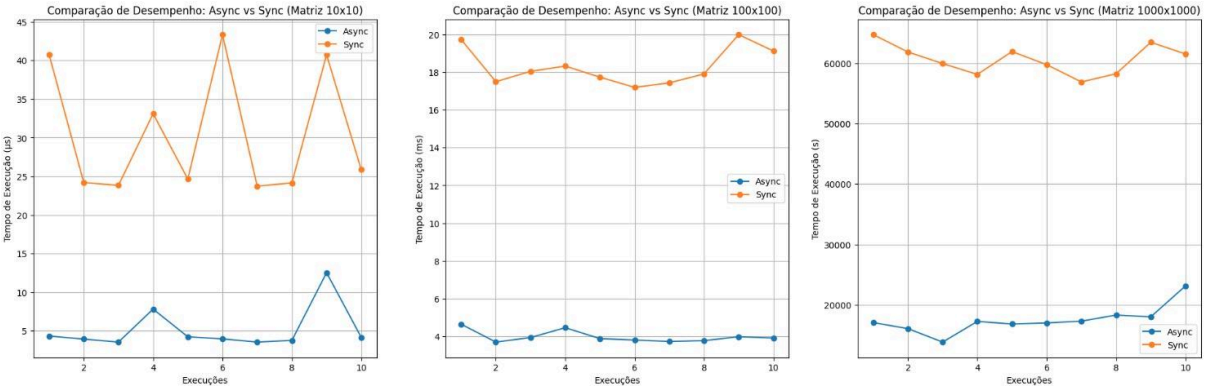
Anexado na atividade

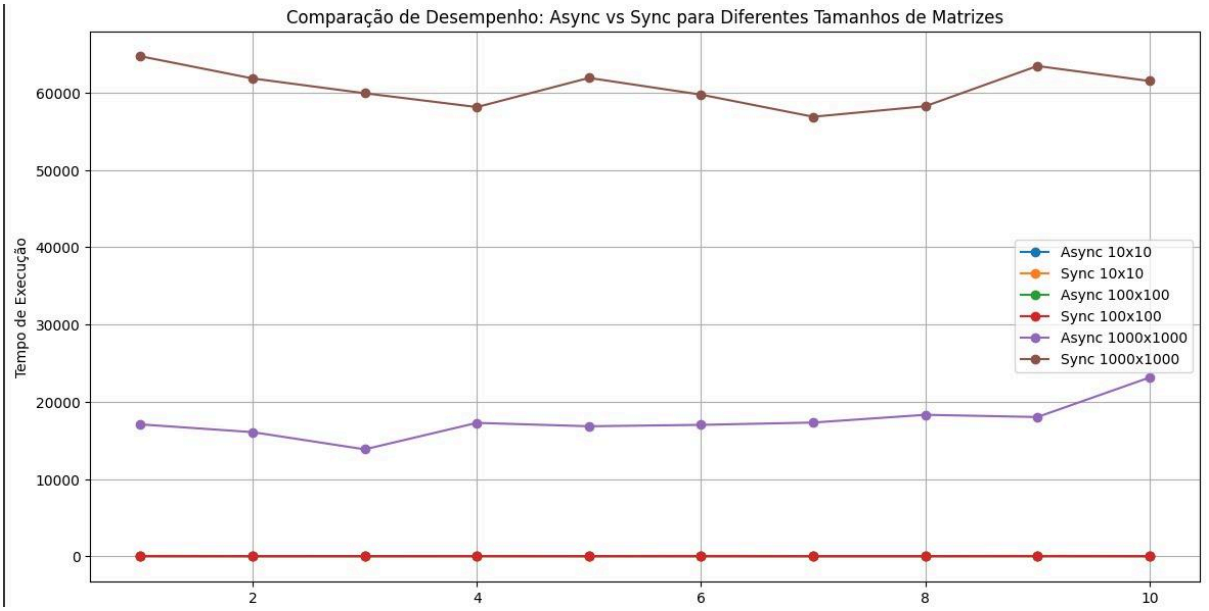
8) Resultados:






2
4
6
8
10
Execuções
Comparação de Desempenho: Async vs Sync (Matriz 1000x1000)
60000
50000
40000
30000
20000
Tempo de Execução (s)
Async
Sync


Comparação de Desempenho: Async vs Sync (Matriz 10x10)
Tempo de Execução (µs)
Execuções
Async
Sync
Comparação de Desempenho: Async vs Sync (Matriz 100x100)
Tempo de Execução (ms)
Execuções
Async
Sync
Comparação de Desempenho: Async vs Sync (Matriz 1000x1000)
Tempo de Execução (s)
Execuções
Async
Sync


Comparação de Desempenho: Async vs Sync para Diferentes Tamanhos de Matrizes
Tempo de Execução
Async 10x10
Sync 10x10
Async 100x100
Sync 100x100
Async 1000x1000
Sync 1000x1000

Para matrizes pequenas (10x10), a abordagem assíncrona é consistentemente mais rápida e estável do que a síncrona, que sofre com a sobrecarga de gerenciamento de concorrência.Para matrizes de tamanho médio (100x100), a abordagem assíncrona ainda se mostra superior, sendo significativamente mais rápida e menos variável em termos de tempo de execução. Para matrizes grandes (1000x1000), a abordagem assíncrona continua a ser superior. A diferença de desempenho é ainda mais pronunciada, mostrando que a sincronização impõe uma sobrecarga muito alta para grandes conjuntos de dados. Vamos tentar analisar para vermos onde estamos errando na abordagem com concorrência para melhorar o desempenho.