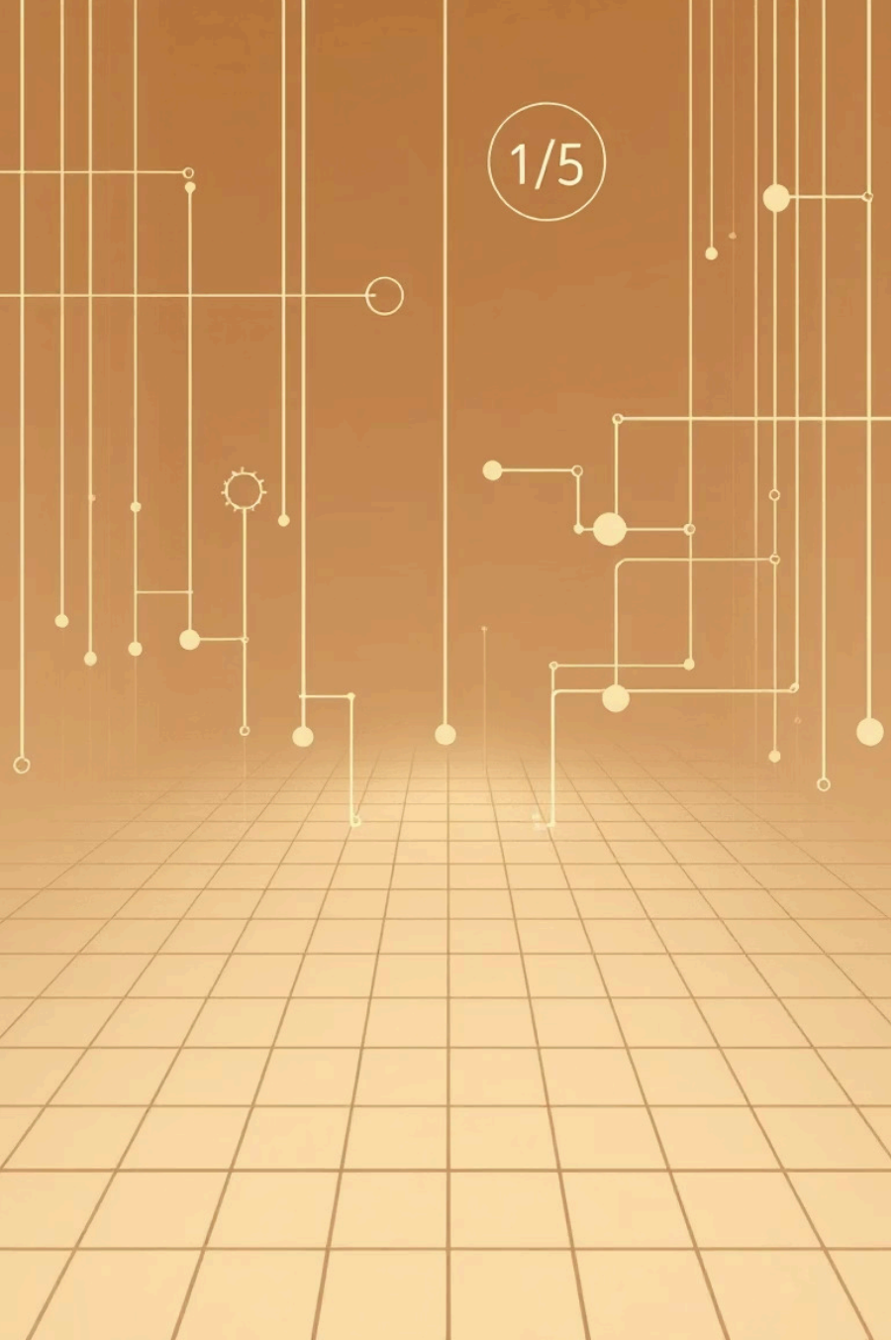


SIMULAÇÃO DE DIFUSÃO DE CALOR EM ARQUITETURAS SEQUENCIAL, PARALELA E DISTRIBUÍDA

Uma análise comparativa de desempenho e escalabilidade.

Amanda Beatriz Machado - RA 2411326

Ana Carolina Moreira da Silva - RA 2418169

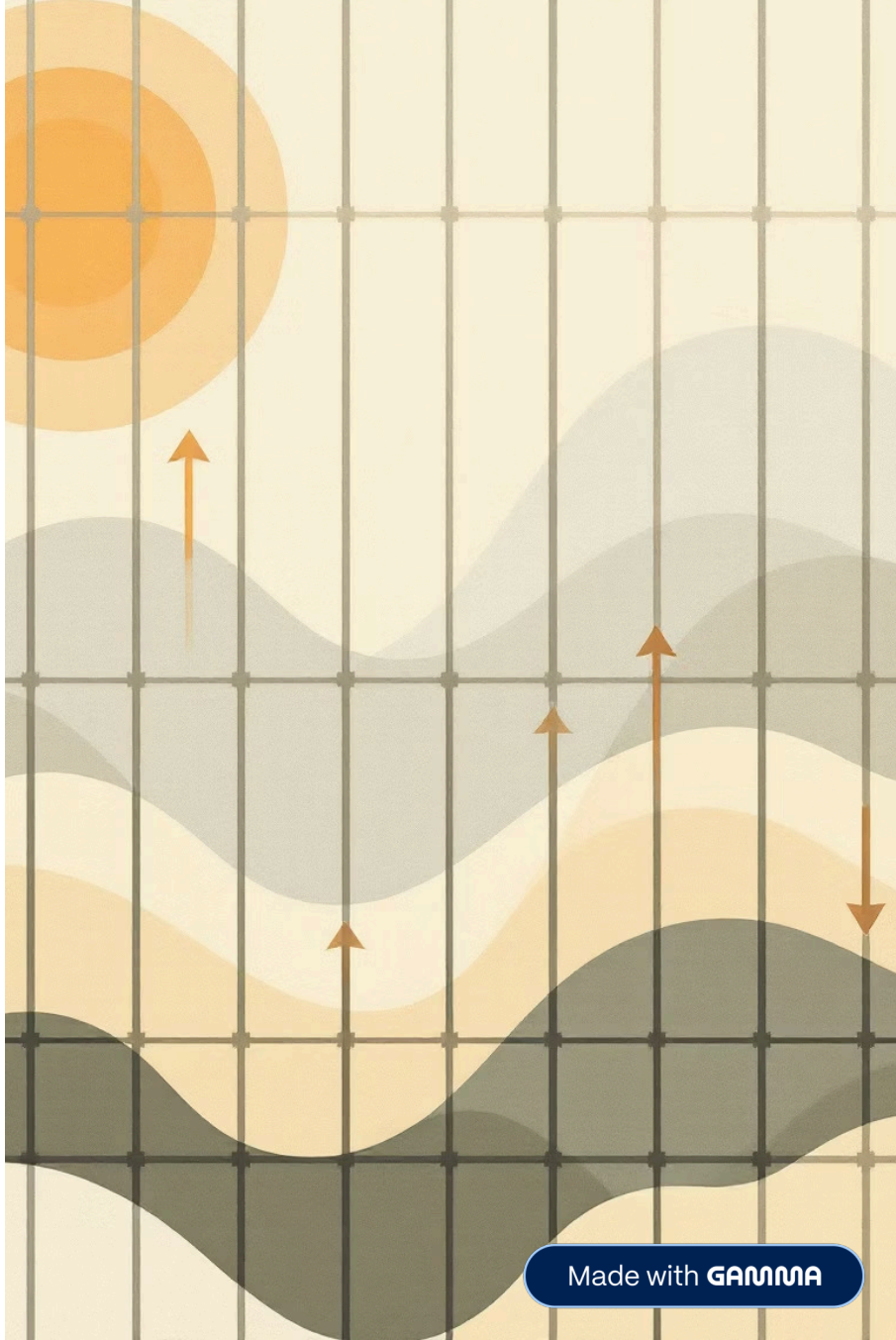
Bruna Naian Moreira Lima Garcia - RA 2454300

Giovana Kaori Parpinelli - RA 2269139

Sumário

Introdução: O Método de Jacobi para Difusão de Calor

O Método de Jacobi é um algoritmo clássico de relaxamento para resolver equações diferenciais parciais, como a equação do calor. Ele simula a propagação de calor em uma malha 2D (matriz) ao longo do tempo, identificando a temperatura de cada célula com base na média de seus quatro vizinhos ortogonais.



Funcionamento do Algoritmo de Jacobi

Geração da Malha

Cria uma matriz N x N com temperatura inicial constante e uma fonte de calor fixa em uma borda (ex: superior).

Cálculo Iterativo

Calcula a nova temperatura da célula interna usando valores da matriz da iteração anterior, evitando conflitos.

Critério de Parada

O processo continua até que a variação máxima de temperatura (max_diff) seja menor que o critério de convergência ou o limite máximo de iterações (T) seja atingido.

Tabelas Comparativas: Desempenho das Abordagens

As tabelas comparativas detalham o desempenho das abordagens Sequencial, Paralela (Threads) e Distribuída (Hosts), mostrando tempos de execução para uma matriz 200x200. Os objetivos de medição incluem a análise da escalabilidade horizontal e vertical.

Tabela Sequencial (Baseline)

TAMANHO DA MATRIZ	TEMPO DE EXECUÇÃO
200x200	111.46
400x400	3032.90
600×600	8780.10
800×800	17057.23

Fonte: autoria própria

Tabela Paralela (Escalabilidade Vertical)

TAMANHO DA MATRIZ	TEMPO (MS) COM 1 THREAD	TEMPO (MS) COM 2 THREADS	TEMPO (MS) COM 4 THREADS
200x200	118.35	185.83	149.01
400x400	3286.09	2333.97	891.51
600×600	N/A	8535.96	1385.37
800×800	N/A	12811.01	3242.53

Fonte: autoria própria

Tabelas Comparativas: Escalabilidade Distribuída

A tabela abaixo demonstra a escalabilidade horizontal da solução distribuída, comparando o tempo de execução com diferentes números de hosts para uma matriz de 200x200.

Tabela Distribuída (Escalabilidade Horizontal)

TAMANHO DA MATRIZ	TEMPO (MS) COM 1 HOSTS	TEMPO (MS) COM 2 HOSTS	TEMPO (MS) COM 3 HOSTS
200×200	32824.40	17361.30	14126.92
400×400	N/A	88516.16	N/A
800×800	N/A	877331.21	N/A

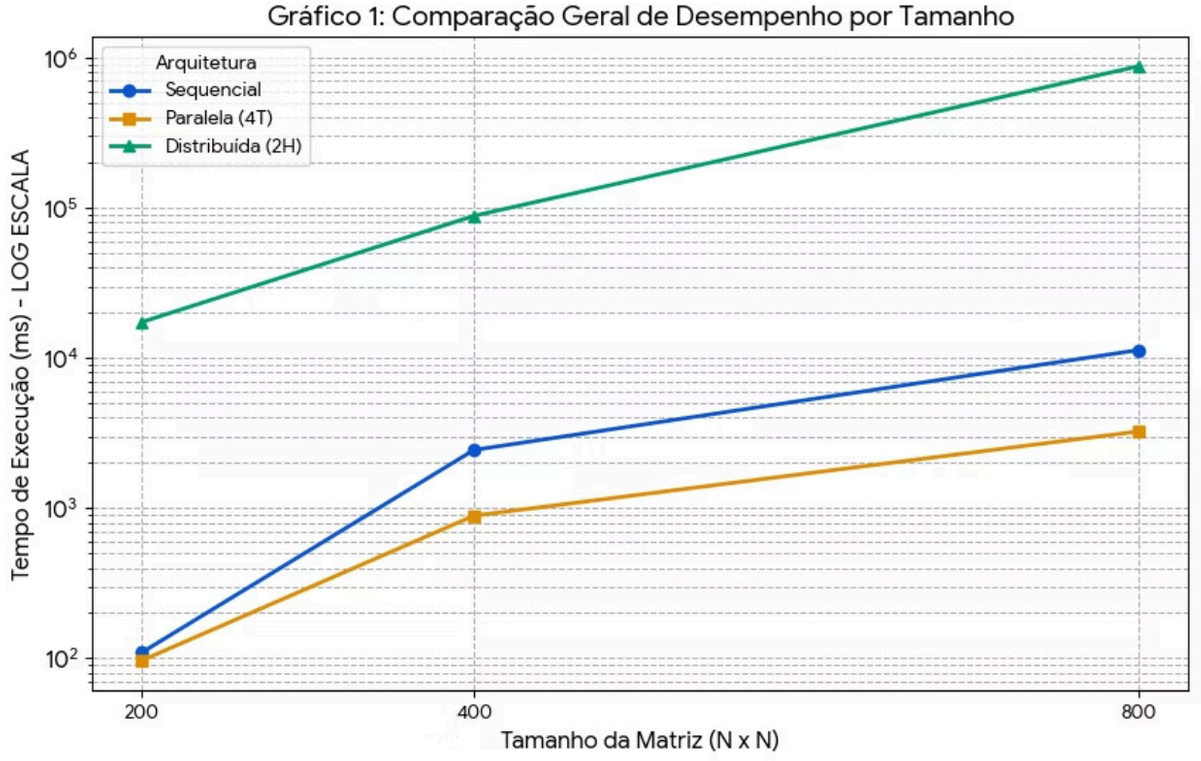
Fonte: autoria própria

Gráficos Comparativos: Visualizando o Desempenho

Os gráficos a seguir ilustram o impacto do overhead e a escalabilidade de cada abordagem. Eles comparam o desempenho geral, o efeito do número de threads e a escalabilidade horizontal com múltiplos hosts.

Comparativo Geral

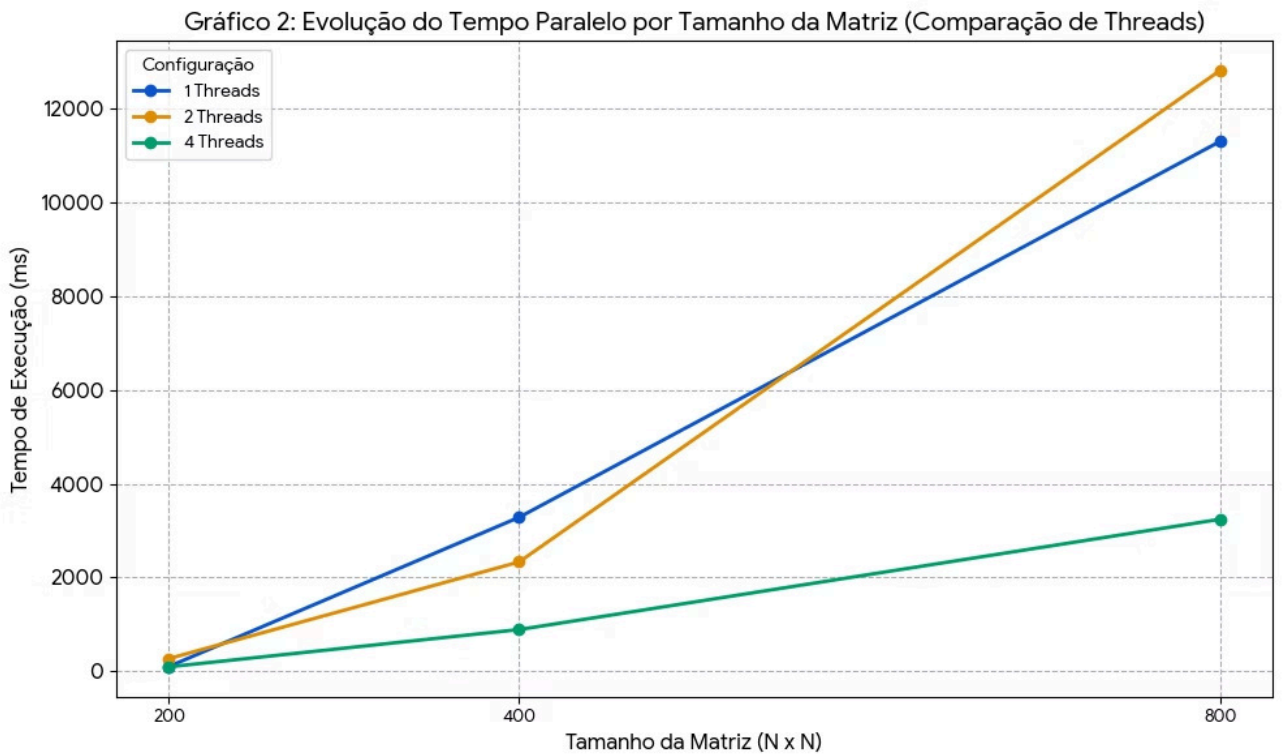
Sequencial vs. Paralela (4t) vs. Distribuída (3h)



Este gráfico compara o tempo de execução das três arquiteturas (Sequencial, Paralela 4T e Distribuída 2H) em função do tamanho da matriz ($N \times N$).

Impacto de Threads

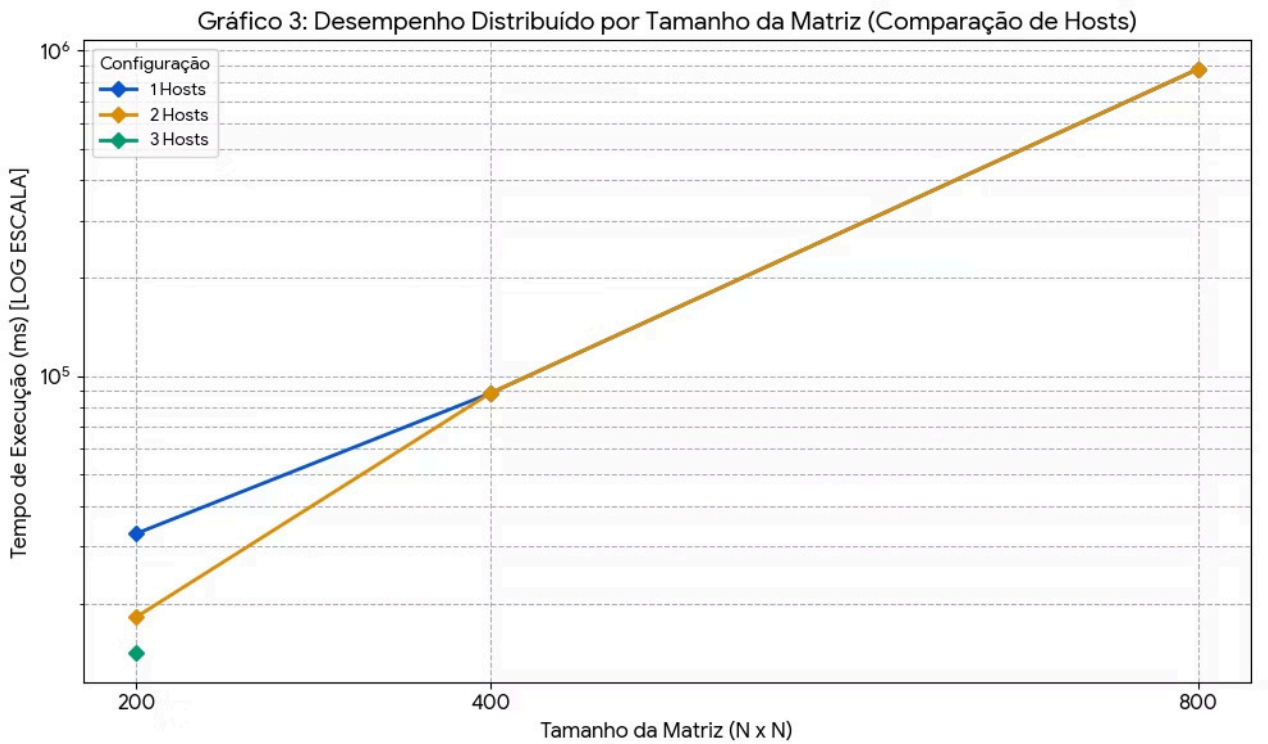
1 vs. 2 vs. 4 Threads (evidência do GIL)



Este gráfico demonstra como a performance varia com o aumento do tamanho da matriz, comparando as diferentes configurações de Threads.

Escalabilidade Distribuída

1 vs. 2 vs. 3 Hosts



Este gráfico usa a escala logarítmica no eixo Y para ilustrar o enorme crescimento do tempo à medida que o tamanho da matriz aumenta, comparando as configurações de Hosts.

Funcionamento das Soluções

Sequencial

Processamento linear da malha. Simples e rápido para matrizes pequenas devido à otimização do NumPy, mas ineficiente para matrizes grandes.

Paralelo (Threads)

Divisão da matriz em fatias para processamento por múltiplas threads. Observou-se alto overhead de criação/sincronização de threads em Python (GIL) e problemas de condição de corrida.

Distribuído (Hosts)

Divisão da malha e envio do trabalho a múltiplos hosts via Sockets. Permite escalabilidade real, mas o overhead de comunicação de rede (pickle, latência) limita ganhos em problemas menores.

Conclusão: Análise das Arquiteturas e Melhorias

1

Sequencial

Ideal para matrizes pequenas e médias (N=200, N=300), servindo como baseline.

2

Paralela

Limitações devido ao GIL do Python e overhead de sincronização, introduzindo incorreção. Indicado para tarefas com pouco I/O ou linguagens sem GIL.

3

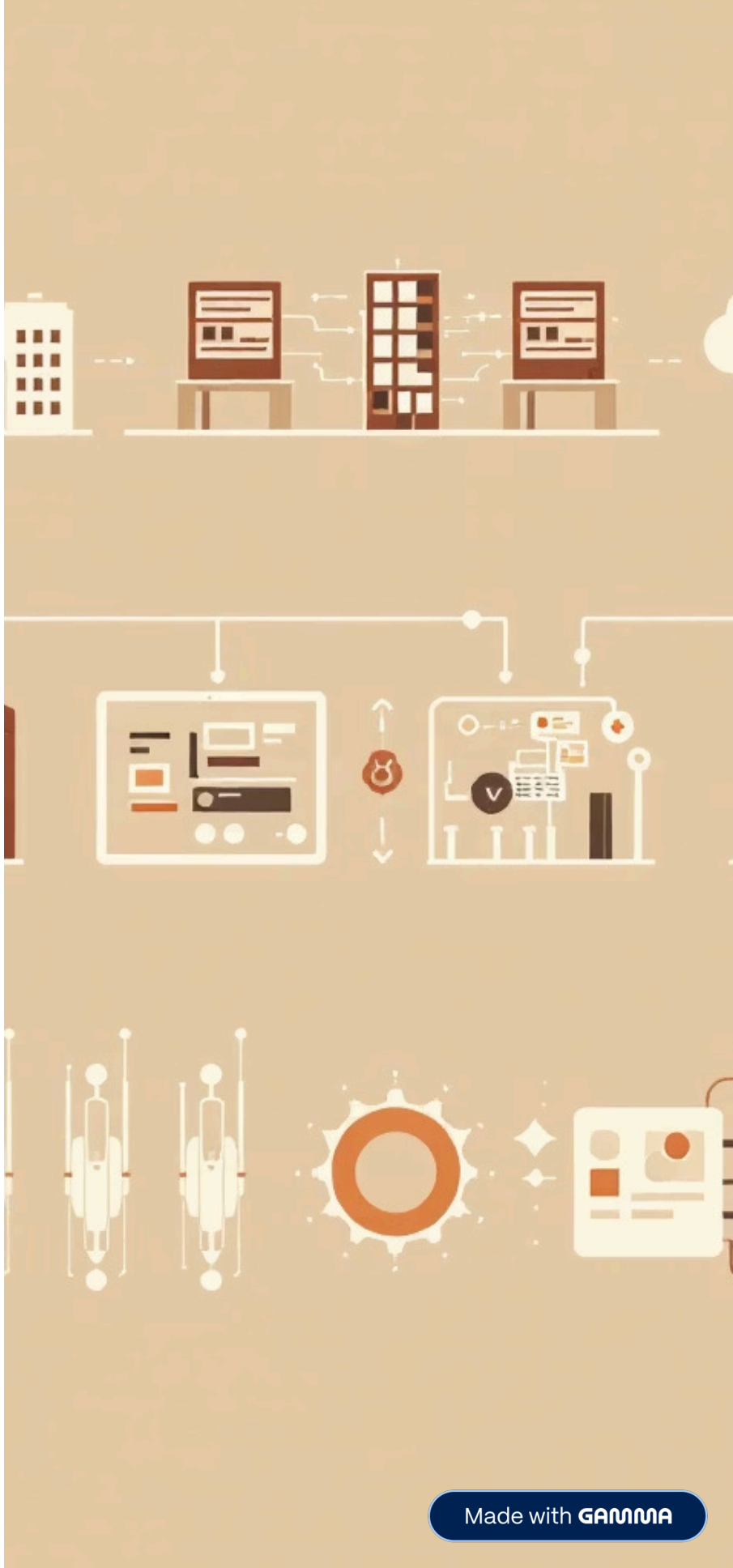
Distribuída

Excelente para escalabilidade em ambientes com múltiplos hosts, reduzindo o tempo de execução para grandes volumes de dados. Overhead de comunicação limita ganhos em problemas menores.

4

Melhorias Sugeridas

Otimizar comunicação do Distributed (pacotes mais leves que pickle) e migrar a versão Paralela para Multiprocessamento para contornar o GIL.



Referências e Perguntas

- TANENBAUM, Andrew S.; VAN STEEN, Maarten. *Sistemas Distribuídos: Princípios e Paradigmas*. [Última Edição]. Pearson Prentice Hall.
- FOSTER, Ian. *Designing and Building Parallel Programs: Concepts and Experience*. [Última Edição]. Addison-Wesley.
- PACHECO, Peter S. *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011.
- CHAPRA, Steven C.; CANALE, Raymond P. *Métodos Numéricos para Engenharia*. [Última Edição]. McGraw-Hill Education.

Obrigado pela Atenção!

Estamos abertos para suas perguntas.