Jogo da Vida

Implementação e Comparação: Sequencial, Paralela e Distribuída

Aluno: Renan Gabriel Bueno

RA: 2454254

Objetivo Geral

Este trabalho visa explorar a implementação do autômato celular Jogo da Vida em três arquiteturas distintas, analisando criticamente o desempenho, escalabilidade e custo computacional de cada abordagem.

Versão Sequencial

Execução linear, ideal para baseline.

Versão Paralela

Utiliza múltiplas threads para processamento simultâneo.

Versão Distribuída

Divisão do trabalho entre múltiplos nós via rede.

As Regras Clássicas do Jogo da Vida

Em todas as implementações, seguimos rigorosamente as regras originais de Conway, que ditam a evolução de cada célula a cada iteração:

→ Sobrevivência

Uma célula viva com **2 ou 3 vizinhos vivos** permanece viva.

→ Nascimento

Uma célula morta com **exatamente 3 vizinhos vivos** nasce.

→ Morte

Em qualquer outra condição, a célula **morre** por subpopulação ou superpopulação, ou permanece morta.

→ Nova Matriz

Cada iteração resulta na **criação completa de uma nova matriz** para representar o próximo estado do Jogo da Vida.

O Jogo da Vida em funcionamento

Versão Sequencial

A implementação sequencial do Jogo da Vida serve como a base fundamental para este estudo. Nesta abordagem, o algoritmo processa cada célula da matriz de forma individual e consecutiva, garantindo a consistência das regras de evolução em cada iteração.

- **Processamento Unifilar:** Todas as operações são realizadas por um único thread, calculando o estado de cada célula uma por uma.
- **Simplicidade e Clareza:** É a forma mais direta de implementar as regras, sendo ideal para validação e depuração.
- **Cálculo Iterativo:** Uma nova matriz é gerada a cada iteração, garantindo que todas as células se atualizem simultaneamente com base no estado anterior.

Esta versão, embora simples e de fácil compreensão, estabelece o ponto de referência inicial para a avaliação de desempenho em comparação com as implementações mais complexas.

Versão Sequencial
A Base para Comparação

Como Funciona

- Processamento em um **único fluxo de execução**.
- Cada célula da matriz é analisada **individualmente**.
- A cada iteração, a matriz é reconstruída **linearmente**.

Vantagens

- Código **simples e direto**.
- Baixa **chance de erros** de sincronização.
- Serve como **linha de base** para comparar outras versões.

Desvantagens

- **Não utiliza múltiplos núcleos** do processador.
- **Baixa escalabilidade** para grandes problemas.
- Matrizes grandes **multiplicam exponencialmente o tempo de execução**.

Versão Paralela com Threads

A Busca por Aceleração Local

Como Funciona

- A matriz é **dividida** entre várias threads.
- Cada thread processa uma **parte das células**.
- As threads **sincronizam** seus resultados a cada nova geração.

Vantagens

- Potencial para **reduzir o tempo total de execução**.
- **Aproveita múltiplos núcleos** da CPU disponível.
- Relativamente **fácil de testar** em ambiente local.

Desvantagens

- **Overhead de criação e sincronização** de threads.
- **Acesso simultâneo à memória** pode reduzir a eficiência.
- O aumento de threads **nem sempre garante** melhor desempenho.

Por Que o Paralelo Não Acelerou Tanto?

Entendendo os Desafios de Desempenho

1

Overhead Maior que o Trabalho

Em matrizes menores (ex: 500x500), o custo de criar e sincronizar threads **supera os ganhos** do paralelismo.

2

Lei de Amdahl

A existência de uma parte **inerentemente sequencial** no algoritmo limita o ganho máximo que pode ser obtido pelo paralelismo.

3

Conflito de Memória

Múltiplas threads acessando e atualizando a mesma estrutura de dados simultaneamente geram **contenção e esperas**.

4

Falso Compartilhamento

Threads acessando dados em **linhas de cache próximas**, mesmo que não sejam os mesmos dados, causam invalidade e recarga de cache.

Os testes revelaram que o paralelismo não gerou ganhos significativos, em muitos casos resultando em desempenho igual ou inferior à versão sequencial, especialmente com o aumento do número de threads.

Versão Distribuída

Escalabilidade Além de Um Único Nó

Como Funciona

- A matriz é **dividida** entre múltiplos computadores ou processos.
- A comunicação e sincronização ocorrem via **Sockets ou Java RMI**.
- Cada nó calcula uma parte e envia o resultado para um **servidor coordenador**.

Vantagens

- Permite **escalar horizontalmente** (adicionar mais máquinas).
- Capaz de processar **matrizes muito maiores**.
- O benefício aumenta **proporcionalmente ao tamanho do problema**.

Desvantagens

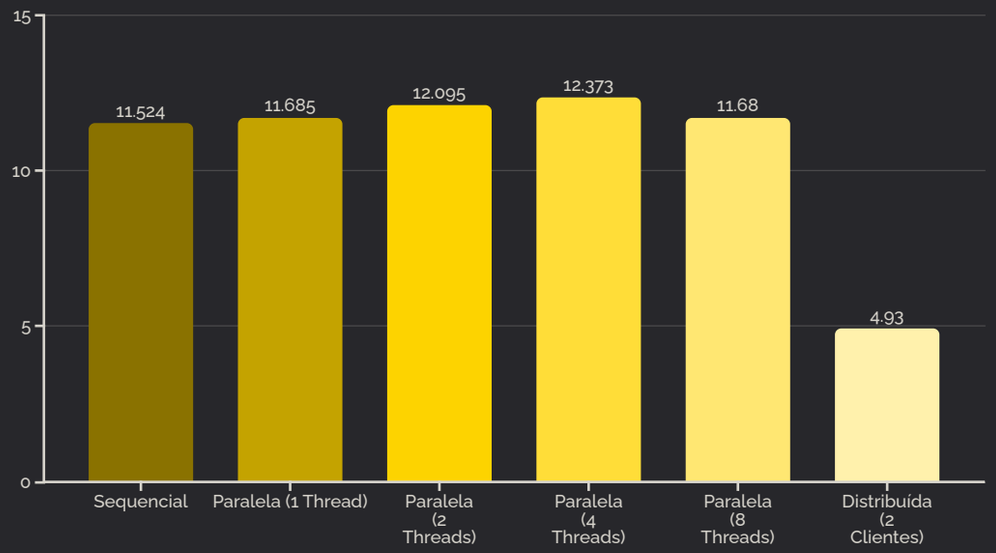
- **Latência de rede** é um fator crítico.
- Sincronização entre nós é **significativamente mais lenta**.
- **Custo de serialização e transferência** de dados é alto.
- Para problemas pequenos, pode ser **mais lenta** que a versão paralela.

Detalhes dos Resultados

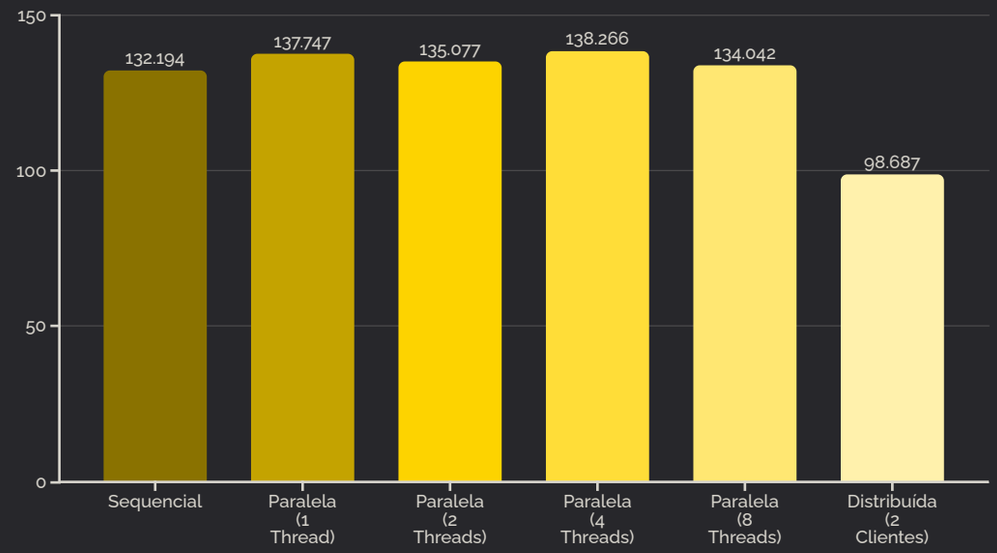
Tempos Médios de Execução por Cenário

Para uma análise mais aprofundada, apresentamos as médias dos tempos de execução para cada configuração de teste.

Tamanho da Matriz: 500x500
30 Interações



Tamanho da Matriz: 2000x2000
20 Interações



Conclusões Finais

Lições Aprendidas sobre Paralelismo e Distribuição

Sequencial como Referência

Simples, confiável e essencial como base de comparação.

Paralelismo Desafiador

Não escalou bem devido a **overhead de threads e disputa de memória**.

Mais Threads ≠ Mais Velocidade

Aumento de threads não garante melhor desempenho; pode até piorar.

Distribuição Vencedora

Trouxe ganhos significativos: **50% em 500x500 e 25% em 2000x2000**.

Gargalo Comum

Sincronização e comunicação são os **maiores obstáculos** em ambas abordagens.

Este exame demonstrou, na prática, as complexidades e os impactos das arquiteturas **sequenciais, paralelas e distribuídas** no contexto dos autômatos celulares. Um trabalho focado na implementação correta, análise experimental detalhada e comparação prática.

Fontes Consultadas

Base Sólida para o Desenvolvimento e Análise

Documentação Oficial Python

Detalhes sobre `threading`, `socket` e `time` foram cruciais para as implementações.

Wikipedia "Conway's Game of Life"

Forneceu a ideia base e as regras clássicas para a implementação sequencial do algoritmo.

Real Python e Stack Overflow

Exemplos e discussões sobre a estrutura de threads com divisão de linhas.

ChatGPT (OpenAI) e Grok (xAI)

Auxílio na correção e otimização do código, especialmente em aspectos de `lock` e sincronização.

"Python Speed" e Medium

Artigos e análises sobre testes de desempenho e a explicação do Global Interpreter Lock (GIL).

Agradecemos a todas as fontes que enriqueceram este estudo, garantindo a robustez teórica e prática das implementações.